UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Triennale in Informatica

Tecniche di Riconoscimento e Offuscamento del Traffico di Rete

Relatore:

Luca Deri

Candidato:

Francesco Pio Magrì

ANNO ACCADEMICO 2024/2025

Indice

Elenco delle figure

1. INTRODUZIONE

L'evoluzione delle infrastrutture e l'aumento esponenziale del traffico di rete hanno accentuato la rilevanza delle problematiche connesse alla privacy e alla sicurezza dei dati trasmessi in rete. In questo contesto, le Virtual Private Network (VPN) e i sistemi proxy, sono diventati strumenti essenziali e sempre più utilizzati per garantire anonimato e confidenzialità agli utenti. Tuttavia, l'adozione diffusa di queste tecnologie ha stimolato parallelamente lo sviluppo di metodi sofisticati per identificarne e monitorarne l'utilizzo, mettendo in discussione l'effettiva sicurezza offerta da tali soluzioni.

Questa tesi affronta in maniera sistematica il problema dell'identificazione e dell'offuscamento del traffico generato da VPN e proxy, con particolare attenzione a OpenVPN e Shadowsocks, ritenuti come due tra le soluzioni attualmente più utilizzate [1]. Attraverso una revisione approfondita della letteratura esistente, si analizzeranno le tecniche di riconoscimento del traffico OpenVPN e Shadowsocks più promettenti e le tecniche di offuscamento più diffuse, verificando se quanto riportato dagli studi più rilevanti trova riscontro empirico.

E' inoltre effettuata una rassegna dei principali lavori che utilizzano algoritmi di Machine Learning per il riconoscimento del traffico VPN e proxy. Tuttavia, tale analisi sarà limitata e superficiale, poichè allo stato attuale il Machine Learning non è ancora in grado di garantire un'identificazione efficace del traffico in tempo reale e su larga scala. La maggior parte degli studi in questo ambito si basa su *dataset* sintetici e su esperimenti condotti in ambienti controllati [48], distanti dalle condizioni operative reali. Come evidenziato da recenti studi, tra le problematiche presenti vi sono la necessità di grandi quantità di dati etichettati, la complessità computazionale dei modelli e le difficoltà nell'adattamento a nuovi tipi di traffico o a cambiamenti nel comportamento della rete, rendendo queste tecniche inadeguate, allo stato attuale, per applicazioni pratiche immediate [13, 3].

Per questa ragione, ho deciso di approfondire alcune tra le tecniche di riconoscimento del traffico OpenVPN e Shadowsocks più promettenti e la loro efficacia in presenta delle tecniche di offuscamento prese in esame. L'obiettivo di questa tesi è valutare se l'ap-

plicazione di queste tecniche di offuscamento renda più complessa l'identificazione delle soluzioni di anonimato prese in esame, e verificare se permangono vulnerabilità sfruttabili mediante tecniche avanzate di filtraggio passivo e *active probing*.

Per OpenVPN, al fine di validare alcune promettenti e innovative tecniche di identificazione, ho realizzato script in Python che implementano le euristiche di identificazione di OpenVPN [71], e ne ho analizzato l'efficacia anche in presenza di traffico offuscato. I risultati ottenuti sono poi stati messi a confronto con quanto atteso sulla base della letteratura analizzata.

La discussione evidenzia eventuali discrepanze tra i risultati ottenuti dai test e quelli previsti dalla letteratura offrendo spiegazioni puntuali circa le cause di tali differenze. Infine, saranno proposte alcune linee guida per lavori futuri, mirati a superare le limitazioni individuate e a perfezionare ulteriormente le tecnologie di offuscamento del traffico di rete

Struttura della tesi

- Capitolo 2: Spiegazione dei fondamenti del traffico di rete (struttura dei pacchetti e flussi di rete) e tecniche di monitoraggio e analisi del traffico di rete (DPI e DFI);
- Capitolo 3: Nozioni sul fingerprinting, modalità di acquisizione dei dati necessari alla classificazione del traffico di rete e obiettivi dell'analisi del traffico;
- Capitolo 4: Presentazione delle tecniche di rilevazione dei protocolli di rete;
- Capitolo 5: Tecnologie VPN e funzionamento alla base dei protocolli OpenVPN e Shadowsocks;
- Capitolo 6: Presentazione delle tecniche più diffuse per l'offuscamento dei protocolli OpenVPN e Shadowsocks;
- Capitolo 7: Presentazione delle tecniche di identificazione dei protocolli OpenVPN e Shadowsocks più innovative, e valutazione della loro efficacia in presenza delle tecniche di offuscamento presentate;
- Capitolo 8: Validazione delle tecniche di identificazione di OpenVPN presentate.

2. TECNICHE DI ANALISI DEL TRAFFICO DI RETE

Le reti di comunicazione rappresentano un'infrastruttura fondamentale per la società odierna. Dall'accesso a servizi cloud alla trasmissione sicura di dati sensibili, le reti interconnettono miliardi di dispositivi in tempo reale, garantendo la condivisione di informazioni su scala globale. Alla base delle reti vi è un processo di trasferimento delle informazioni strutturato, che segue protocolli ben definiti per garantire affidabilità ed efficienza. In questo capitolo si analizzerà come vengono organizzate e scambiate le informazioni nella rete e le tecniche di monitoraggio passivo di pacchetti e flussi, al fine di comprenderne potenzialità e le limitazioni.

2.1 Struttura del traffico di rete

Questa sezione esplora i componenti principali che costituiscono il traffico di rete, illustrando come i dati sono organizzati, trasmessi e gestiti. Queste nozioni risulteranno fondamentali per avere, attraverso l'analisi di pacchetti e flussi, una panoramica dell'attività di rete.

2.1.1 Pacchetti e flussi di rete

Il traffico di rete è organizzato in unità discrete chiamate pacchetti, piccole unità di dati che contengono sia l'informazione stessa (*payload*) sia informazioni utili all'instradamento del pacchetto e alla sua gestione (metadati) [14]. Le informazioni scambiate tra due punti specifici della rete vengono tramutate in pacchetti. Una sequenza di pacchetti costituisce un flusso di rete. Analizzando questi flussi è possibile ricavare informazioni utili sul comportamento della rete, come performance e potenziali anomalie [14].

2.1.2 Metadati e payload

Sebbene la struttura complessiva di un pacchetto dipenda dai protocolli applicativi e di trasporto utilizzati, è possibile definire una struttura generale comune. Ogni pacchetto che transita nella rete è composto da un *header* ed un *payload*.

1. Header: è la porzione di pacchetto che contiene i metadati fondamentali per il corretto instradamento e la gestione dei pacchetti all'interno della rete [22]. Questi includono:

   (a) Indirizzi IP di origine e di destinazione: Determinano da dove proviene il pacchetto e dove deve essere consegnato;

   (b) Numeri di porta: Utilizzati per indirizzare i pacchetti verso le corrette applicazioni sui dispositivi di destinazione;

   (c) Protocollo utilizzato: Indica il protocollo di trasporto (come TCP o UDP) che gestisce la connessione

   (d) Flags vari e controlli di sequenza: Cruciali per la gestione del controllo di flusso e per l'ordinamento dei pacchetti a destinazione.

   Per comprendere la complessità e la quantità delle informazioni contenute nell'header di un pacchetto, la Figura 2.1 mostra la struttura di un pacchetto TCP come definito dallo standard RFC 9293.

2. Payload: è la porzione del pacchetto che contiene i dati effettivi inviati da un utente. Questo può contenere da semplice testo a dati complessi come parti di video, immagini, ecc. È quindi evidente che il *payload* non possa essere trasmesso in chiaro, ma è necessario servirsi di opportune tecniche di crittografia, più o meno complesse, al fine di proteggere i dati da intercettazioni non autorizzate o manipolazioni [22].

2.2 Tecniche di monitoraggio e analisi

Il monitoraggio e l'analisi del traffico di rete sono diventati indispensabili per garantire la sicurezza, l'efficienza e la stabilità delle reti. Con l'espansione delle infrastrutture di rete

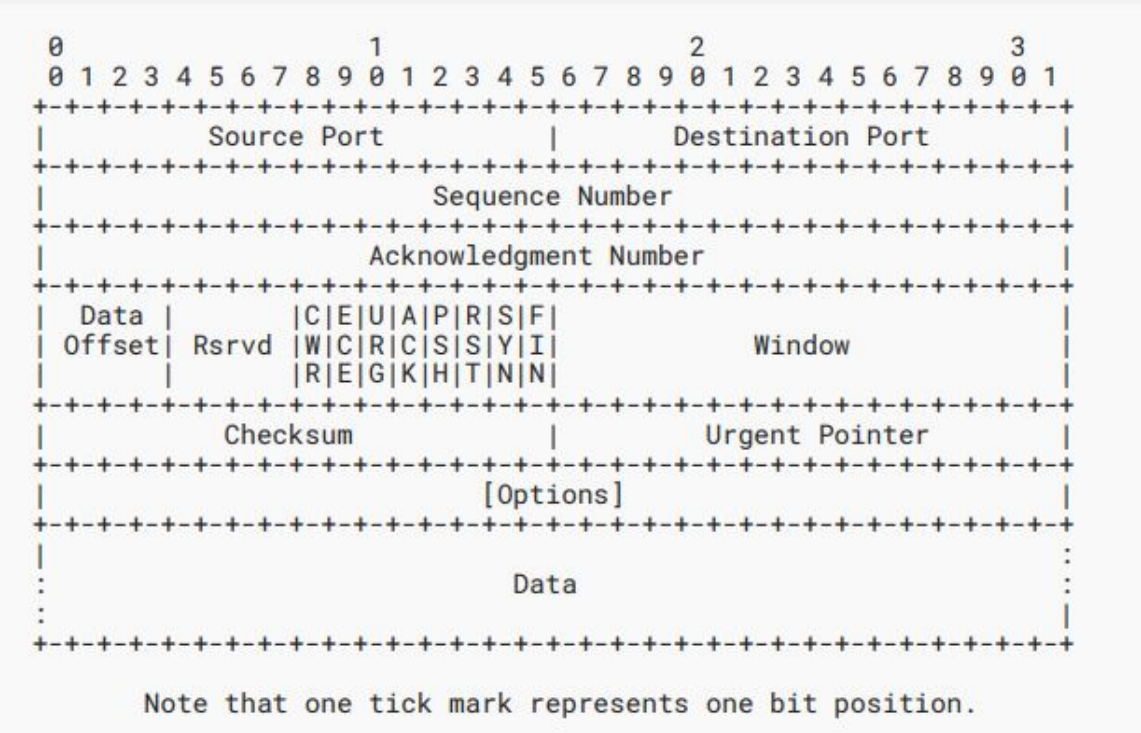

Figura 2.1: Formato header pacchetto TCP [22]

e la crescita delle minacce alla sicurezza, ci si affida sempre di più a tecniche di ispezione del traffico per prevenire, rilevare e rispondere a potenziali problematiche. Tra queste, vi sono la *Deep Packet Inspection* e la *Deep Flow Inspection*, attraverso le quali è possibile avere una visione dettagliata della natura delle informazioni che circolano in rete.

2.2.1 Deep Packet Inspection (DPI)

La *Deep Packet Inspection* è definita come l'analisi dei dati contenuti nel *payload* e nell'*header* di pacchetti che transitano in un determinato punto della rete al fine di comprenderne la natura.

La DPI pone non poche sfide in merito alla velocità di analisi. Infatti, la capacità di processare i dati a velocità superiori alla velocità del traffico di rete è cruciale per evitare perdite di pacchetti [19]. Tra i molteplici utilizzi, abbiamo:

1. Sicurezza di rete: Nel contesto della sicurezza, la DPI può fungere da sentinella. Analizzando dettagliatamente ogni pacchetto, permette di identificare e bloccare il traffico generato da malware [69].

2. Gestione del Traffico**:** La DPI riveste un ruolo fondamentale nella gestione della larghezza di banda e nella prioritizzazione del traffico all'interno delle reti. Attraverso un'analisi approfondita dei dati di rete, la DPI permette agli amministratori di assegnare dinamicamente priorità e risorse di banda in base alla natura e all'importanza del traffico. Questo significa che le applicazioni critiche, come i servizi di comunicazione in tempo reale, possono ricevere la larghezza di banda necessaria per funzionare efficacemente, mentre il traffico meno urgente può essere limitato per evitare congestioni e garantire una distribuzione ottimale delle risorse di rete [44].

2.2.2 Deep Flow Inspection (DFI)

La *Deep Flow Inspection* (DFI) è definita come l'analisi approfondita dei flussi di rete. Questa analisi, a differenza della DPI, non si concentra sul singolo pacchetto e sul suo contenuto, ma su metriche inerenti più pacchetti che transitano in un determinato flusso. Si svolge solitamente per periodi di tempo più lunghi e, non analizzando ogni singolo

pacchetto come la DPI, è computazionalmente più leggera e adatta per grandi volumi di traffico [38]. Tra i molteplici utilizzi abbiamo:

1. Analisi del comportamento: Tramite la DFI è possibile identificare modelli comportamentali di determinati flussi o utenti che utilizzano la rete;

2. Rilevamento di anomalie: La DFI può efficacemente identificare anomalie comportamentali che potrebbero indicare attacchi di rete, come il *Denial of Service* (DoS), o altre attività sospette come le fughe di dati.

Risulta quindi evidente che DPI e DFI abbiano scopi e prestazioni differenti in quanto si concentrano su livelli di granularità diversi del traffico. Per questo motivo, l'utilizzo combinato di DPI e DFI garantisce un'analisi efficiente ed accurata del traffico [6].

In questo capitolo è stata fornita una panoramica sui concetti chiave relativi al traffico di rete. Si è evidenziato come i pacchetti, strutturati in *header* (contenente metadati) e *payload* (contenente le informazioni applicative), rappresentino l'unità minima di trasmissione dei dati . La definizione di flussi di rete ha permesso di contestualizzare l'analisi tipica delle comunicazioni, dalla valutazione delle performance alla rilevazione di anomalie. Infine, l'esame delle tecniche di DPI e DFI ha messo in luce metodologie complementari per il monitoraggio del traffico con la DPI utile per un'ispezione granulare dei singoli pacchetti, mentre la DFI per l'analisi comportamentale su intervalli temporali più estesi.

3. FINGERPRINTING DEL TRAFFICO

Sebbene DPI e DFI siano strumenti potenti per l'analisi diretta dei dati di rete, essi rivelano una vastità di informazioni che richiede un ulteriore livello di sintesi per essere pienamente sfruttata. A tale scopo, si parla di *fingerprinting* del traffico, ossia la scoperta di *pattern* univoci che emergono dall'analisi di flussi o singoli pacchetti, fornendo uno strumento per catalogare e identificare le varie tipologie di traffico con maggiore precisione.

3.1 Cos'è il fingerprinting

Con il termine *fingerprinting*, si fa riferimento al processo attraverso il quale vengono identificate e raccolte le informazioni su un sistema o una rete al fine di creare un profilo di traffico unico, o un' "impronta digitale". Si basa sull'analisi di caratteristiche specifiche del traffico, individuando caratteri unici per classificarlo in modo accurato e dettagliato [41]. Per "caratteri unici" si intendono quelle peculiarità del traffico analizzato, che sono uniche e non si trovano in traffico di altro tipo, e necessitano di essere interpretate in base a diversi livelli di specificità che includono:

- OS fingerprinting: l'identificazione di una caratteristica all'interno del traffico analizzato, che può permetterci di distinguere, ad esempio, il traffico generato da un dispositivo IOS piuttosto che un dispositivo Android [37].

- Application fingerprinting: l'identificazione di una caratteristica all'interno del traffico analizzato, che può permetterci di distinguere, ad esempio, il traffico generato da applicazioni di videoconferenza (es. Skype) dal traffico generato da applicazioni di video streaming (es. Netflix).

- Protocol identification: L'identificazione di quale protocollo viene utilizzato in una comunicazione, attraverso l'analisi di caratteristiche distintive del traffico, come la struttura dei pacchetti, il *timing* e le modalità di *handshake*.

3.2 Modalità di acquisizione dei dati

Quando si parla di *protocol identification*, è fondamentale distinguere come vengono raccolti i dati e che tipo di informazioni vengono analizzate.

1. Modalità di raccolta dei dati:

   (a) Attiva (*probing* attivo): si inviano pacchetti ad un host per provocare una risposta o osservare il comportamento della connessione;

   (b) Passiva (filtraggio passivo): si osserva il traffico senza interagire, limitandosi ad analizzare i pacchetti in transito.

2. Tipo di informazioni analizzate: Si può analizzare il contenuto del pacchetto (*payload*), solo le intestazioni, o caratteristiche aggregate del flusso. La scelta dipende dallo scopo dell'analisi.

3.2.1 Filtraggio passivo

Il filtraggio passivo è una tecnica di analisi di rete che si concentra sull'osservazione e sull'analisi del traffico esistente e non prevede, al contrario del *probing* attivo, alcun invio di pacchetti o richieste a un sistema target [68]. Questa tecnica, nota anche come *sniffing*, opera monitorando i pacchetti che transitano attraverso un punto specifico della rete, per analizzare alcune caratteristiche dei pacchetti come protocolli utilizzati, indirizzi IP e porte, volumi di traffico e tempi di invio e ricezione [41]. A differenza del *probing* attivo, che può essere percepito come aggressivo, il filtraggio passivo è completamente non intrusivo. Tuttavia, questa tecnica può essere meno informativa rispetto al metodo attivo, poiché si limita alle informazioni che possono essere inferite dal traffico osservato. Inoltre, per poter beneficiare di questa tecnica, occorre che venga effettuata in un punto d'osservazione strategico della rete, con un elevato traffico e per periodi di tempo prolungati [9].

3.2.2 Probing attivo

Il *probing* attivo è una tecnica avanzata di analisi di rete che implica l'invio di pacchetti appositamente costruiti verso un sistema target. Questi pacchetti possono essere configurati per testare specifiche vulnerabilità o per provocare una risposta che normalmente non invierebbe, che riveli particolari informazioni su configurazioni, sistema operativo e applicazioni in esecuzione sul sistema target. Le informazioni ricavate contribuiscono alla generazione di un *fingerprint* [68, 41]. Nonostante la sua efficacia, il *probing* attivo è considerato una tecnica invasiva. L'invio di pacchetti non standard o malformati può, infatti, sovraccaricare inutilmente la rete, causare malfunzionamenti o, peggio, essere percepito come un tentativo di intrusione dai sistemi di sicurezza del target, come gli *Intrusion Detection Systems* (IDS) [12]. Pertanto, il *probing* attivo viene generalmente impiegato solo per verificare o confermare sospetti emersi durante il filtraggio passivo [71]. In contesti eticamente e legalmente sensibili, è prassi che tale attività venga eseguita in modo mirato, limitato nel tempo e, idealmente, con il consenso e la collaborazione degli amministratori della rete sondata.

3.3 Obiettivi dell'analisi del traffico

3.3.1 Rilevamento di traffico sospetto o offuscato

Il rilevamento di traffico sospetto o intenzionalmente offuscato rappresenta uno degli obiettivi principali dell'analisi del traffico. Tale rilevamento è una componente essenziale nelle strategie di sicurezza informatica, in quanto consente di identificare in tempo reale attività anomale della rete, che potrebbero celare tentativi di intrusione o forme di attacchi informatici.

Nel traffico di rete standard, la comunicazione segue schemi e protocolli ben definiti; ogni protocollo, infatti, opera secondo regole precise. Quando tali regole non vengono rispettate, o quando vengono introdotte tecniche di offuscamento (che saranno approfondite in seguito), si crea una discrepanza tra il traffico atteso e quello osservato. Questa discre-

panza, se opportunamente identificata, rappresenta un campanello di allarme che non può essere ignorato.

L'analisi di tali anomalie non si limita a un semplice confronto tra traffico legittimo e traffico anomalo, ma assume un ruolo strategico nell'attivazione di meccanismi di difesa. Ad esempio, in scenari in cui il traffico offuscato è impiegato per mascherare la comunicazione tra un server di comando e controllo e i nodi infetti di una botnet, il riconoscimento tempestivo di *pattern* inusuali nel traffico di rete può fornire indizi preziosi per l'attivazione di contromisure difensive.

Questo approccio, quindi, si basa su un continuo monitoraggio e analisi dei flussi di rete, in cui anche piccole deviazioni dagli standard possono costituire un segnale d'allarme.

3.3.2 Censura

L'analisi del traffico di rete, originariamente utilizzato per fini di monitoraggio e sicurezza, si presta anche ad applicazioni nel campo della censura digitale. In questo contesto, la tecnica viene sfruttata per identificare specifici *pattern* di comunicazione associati a determinati protocolli, applicazioni o servizi.

Il meccanismo di analisi del traffico permette di riconoscere con elevata accuratezza il tipo di traffico e, conseguentemente, di attivare misure di blocco o di alterazione della comunicazione. In sostanza, il sistema di analisi diventa uno strumento di filtraggio dinamico, capace di individuare e neutralizzare in tempo reale flussi che non rientrano nelle politiche di accesso consentite.

Una volta individuato un flusso sospetto o appartenente a servizi ritenuti indesiderati, l'intervento può avvenire in vari modi: dalla semplice interruzione della connessione, alla degradazione intenzionale della qualità del servizio, fino al reindirizzamento verso pagine informative o messaggi di blocco. Questo approccio non solo impedisce la libera circolazione dei contenuti, ma agisce anche da deterrente, costringendo gli utenti a ricorrere a metodi di evasione che, a loro volta, possono essere ulteriormente analizzati e neutralizzati.

In questo senso, il sistema più potente di censura risulta essere il *Great Firewall* (GFW) cinese. Il GFW, infatti, utilizza metodologie combinate di filtraggio passivo e probing attivo per bloccare le comunicazioni che utilizzano protocolli offuscati e limitare l'accesso a contenuti esterni [70]. Nei capitoli successivi, viene approfondito il suo funzionamento.

3.3.3 Supporto alle Contromisure e alla Gestione delle Minacce

L'analisi del traffico può altresì concentrarsi sul supporto attivo alle contromisure e sulla gestione delle minacce, divenendo uno strumento dinamico e proattivo per la difesa delle infrastrutture di rete. Ad esempio, qualora venga individuato un flusso anomalo, i dati derivanti dalla classificazione ottenuta tramite filtraggio passivo possono essere utilizzati per attivare contromisure mirate, come il blocco di una connessione, l'aggiornamento dinamico delle regole del firewall, ecc. Inoltre, l'integrazione dei dati classificati, ottenuti tramite filtraggio, con sistemi di monitoraggio centralizzati, quali piattaforme di *Security Information and Event Management* (SIEM), consente la correlazione di vari eventi di sicurezza rilevati. Ciò permette di ricostruire un quadro complesso degli eventi, individuando *pattern* che potrebbero sfuggire a un'analisi frammentaria.[49]

Oltre all'aspetto reattivo, questo processo supporta una strategia difensiva proattiva. L'analisi approfondita dei dati classificati consente di identificare precocemente segnali di compromissione, facilitando l'adozione di contromisure preventive che riducono il tempo di risposta e limitano l'impatto degli attacchi. In questo contesto, l'analisi del traffico di rete si configura come una componente fondamentale per affinare la strategia difensiva di un'infrastruttura di rete in quanto i feedback derivanti dall'analisi dei flussi e dalla loro classificazione, consentono di perfezionare i modelli di rilevamento e di adattarsi alle nuove minacce. Inoltre, i dati raccolti rappresentano una risorsa preziosa per la ricostruzione degli eventi post-incidente e per l'identificazione delle vulnerabilità sfruttate.

4. TECNICHE DI RILEVAZIONE DEI PROTOCOLLI

L'identificazione dei protocolli in un flusso di rete, come detto precedentemente, non dipende unicamente da come i dati vengono raccolti, ma soprattutto da quali informazioni si decide di analizzare.

Quando il contenuto del payload non è disponibile (perchè cifrato o offuscato), l'attenzione si sposta su elementi alternativi come le intestazioni, i metadati o le caratteristiche aggregate dei flussi

Questo capitolo approfondisce proprio tali aspetti, analizzando le principali tecniche che permettono di estrarre valore informativo da ciò che rimane osservabile nel traffico cifrato o offuscato. In particolare, si esploreranno:

- l'analisi statica e dei *pattern*, focalizzata sulle proprietà strutturali dei pacchetti e dei flussi (es. dimensione, timing, sequenza);
- l'analisi dei metadati, che include lo studio di header, indirizzi IP, porte, TTL e flag;
- l'impiego di algoritmi di Machine Learning che permettono di estrarre feature statistiche dal traffico.

4.1 Analisi statica e dei pattern

L'analisi statica e dei *pattern* rappresenta una fase fondamentale nello studio del traffico di rete, in quanto consente di identificare e isolare caratteristiche intrinseche dei flussi di dati che, nonostante le tecniche di offuscamento, possono rivelare informazioni utili per il riconoscimento di protocolli, applicazioni o comportamenti specifici. Questa sezione si focalizza su due aspetti principali: le caratteristiche dei flussi e l'analisi dei metadati. Entrambe le analisi, pur operando su livelli differenti del traffico, contribuiscono a definire un profilo di comportamento utile ad identificare l'utilizzo di protocolli applicativi, anche se offuscati.

4.1.1 Caratteristiche dei flussi

L'analisi delle caratteristiche dei flussi consiste nell'identificare e quantificare una serie di parametri intrinseci ai pacchetti che costituiscono il traffico di rete. Questi parametri includono, ma non si limitano a:

- Dimensione dei pacchetti: Ogni pacchetto trasmesso ha una dimensione che può variare in base al protocollo utilizzato o al tipo di contenuto. L'analisi della distribuzione delle dimensioni dei pacchetti permette di individuare *pattern* caratteristici che possono essere associati a specifiche applicazioni o tecniche di offuscamento [17]. Ad esempio, alcuni protocolli di streaming video potrebbero mostrare una distribuzione bimodale dovuta alla combinazione di pacchetti di controllo e di dati [56].

- Timing e frequenza: Il *timing*, ossia la misurazione degli intervalli temporali tra l'arrivo dei pacchetti, gioca un ruolo cruciale nell'identificazione di protocolli offuscati. In molte implementazioni di offuscamento, viene applicata una randomizzazione o una manipolazione dei tempi di trasmissione per rendere più difficile l'identificazione del flusso [32]. Tuttavia, anche queste variazioni possono rivelare *pattern* distintivi: ad esempio, la presenza di ritardi fissi o variabilità statisticamente rilevabili nel ritardo (*jitter*) possono indicare l'impiego di particolari algoritmi di *buffering* o di controllo della congestione [74].

- Sequenza e ordine dei pacchetti: L'analisi della sequenza dei pacchetti, inclusi numeri di sequenza e ACK (*acknowledgment*), offre ulteriori spunti. Anche in presenza di tecniche di offuscamento, l'ordine e la logica di scambio dei pacchetti tendono a rispettare schemi deterministici definiti dai protocolli sottostanti, il che permette di ricostruire, in modo indiretto, il flusso di comunicazione. Ad esempio, in una connessione TCP, la sequenza di pacchetti seguendo l'handshake e la successiva trasmissione dati possono presentare *pattern* riconoscibili [71].

4.1.2 Analisi dei metadati

L'analisi dei metadati nel traffico di rete si focalizza sull'esame degli *header* e degli attributi informativi che accompagnano ogni pacchetto. Questi metadati, pur non contenendo il *payload* effettivo, offrono una miriade di informazioni utili ad identificare protocolli, poiché spesso includono dati strutturati e ripetitivi che possono essere sfruttati per dedurre il tipo di comunicazione o il protocollo utilizzato. Gli aspetti principali solitamente considerare sono:

- Indirizzi IP di origine e destinazione: Essenziali per identificare la provenienza e la destinazione del traffico. Gli indirizzi IP possono essere molto utili per riconoscere *pattern* di comunicazione sospetti, come ad esempio la comunicazione ripetuta tra un host della rete e un indirizzo esterno, noto per essere associato ad un server di comando e controllo (C2). Individuando gli indirizzi IP malevoli, è possibile inserirli in una blacklist, limitando o eliminando le minacce [27]. Tuttavia, l'efficacia di questa tecnica può essere facilmente compromessa con l'utilizzo di pratiche di *IP Hopping*[1].

- Numeri di porta: Indicano le applicazioni o i servizi utilizzati. L'analisi della frequenza di utilizzo di determinate porte o l'uso di porte non standard, solitamente elevate, può rivelare la comunicazione con servizi malevoli o l'utilizzo di porte. Tuttavia, sebbene porte come la 1194 siano tipicamente associate a protocolli specifici (es. OpenVPN), l'efficacia di questo metodo di rilevamento può essere compromessa. Molti servizi, incluse VPN e malware sono in grado di utilizzare porte comuni considerate legittime, come la 443 o la 53, per mascherarsi meglio [64].

- Time To Live (TTL): Valore che indica il numero di salti (*hop*) residui per il pacchetto, può variare in base al S.O[2] o alla configurazione della rete. Un'analisi del valore del TTL potrebbe rivelare valori anomali, sinonimo di manipolazioni dei pacchetti [72].

[1] È una pratica che consiste nel cambiare frequentemente l'indirizzo IP di un dispositivo

[2] Sistema Operativo

- Flag del protocollo: L'approfondimento dell'analisi dei flag fornisce un potente strumento per il rilevamento precoce di comportamenti malevoli. La capacità di identificare anomalie nella sequenza e combinazione dei flag, unitamente alla loro correlazione con altri parametri del traffico, consente di: prevenire attacchi DoS (es. *SYN flood*) [10, 11], identificare scansioni furtive mirate all'individuazione di falle nella rete [66], rilevare tecniche di evasione o offuscamento impiegate da malware e botnet.

4.2 Approcci basati sul machine learning

Gli approcci basati sul *machine learning* (ML) rappresentano una frontiera avanzata nell'identificazione e classificazione di *pattern*, soprattutto nei casi in cui vengano adottate tecniche di offuscamento che rendono il rilevamento tramite metodi tradizionali meno efficace. Queste tecniche sfruttano algoritmi di apprendimento automatico per analizzare, modellare e riconoscere comportamenti anomali nei flussi di rete, fornendo così una capacità di rilevamento adattiva e in grado di evolversi in funzione di nuove minacce. Di seguito si analizzeranno alcuni tra gli algoritmi di classificazione comunemente utilizzati, e le relative sfide e vantaggi associati a questi approcci.

4.2.1 Algoritmi di classificazione

Gli algoritmi di classificazione sono alla base dei sistemi di ML applicati alla classificazione del traffico di rete, in quanto consentono di categorizzare i flussi di traffico in base a caratteristiche (*feature*) predefinite o apprese. Tra gli algoritmi più utilizzati, abbiamo:

Support Vector Machine

Il *Support Vector Machine* (SVM) è un potente algoritmo di apprendimento supervisionato ampiamente impiegato per affrontare problemi di classificazione e regressione [7]. Il suo principio di funzionamento risiede nella ricerca di un iperpiano ottimale che possa dividere i dati appartenenti a diverse categorie nel modo più efficace possibile. L'obiettivo

è massimizzare la distanza, denominata margine, tra l'iperpiano e i punti dati più vicini a ciascuna classe. Qualora i dati non presentino una separabilità lineare nello spazio di input originale, l'SVM ricorre all'impiego di tecniche del kernel. L'SVM si distingue per la sua efficacia anche in scenari con set di dati di training limitati [29]. Nell'ambito della classificazione del traffico di rete, l'SVM si è dimostrato un approccio promettente per la classificazione di traffico cifrato o offuscato, laddove le tecniche tradizionali, come l'identificazione basata sulle porte o la DPI, risultano insufficienti [75]. Alcuni degli algoritmi di classificazione basati su SVM, si concentrano sull'estrazione di caratteristiche statistiche rilevanti dei flussi come la loro durata, il numero di pacchetti trasmessi, *timing*, frequenza e media della dimensione dei pacchetti. Questi parametri, derivabili direttamente dall'*header* dei pacchetti, consentono di effettuare la classificazione senza la necessità di analizzare il *payload*, rendendo il metodo SVM applicabile anche al traffico cifrato [75]. Un'evoluzione del metodo in questione è dato dall'uso del metodo *Feature Weighted-Degree* (FWD) il quale consente di assegnare un peso specifico ad ogni caratteristica del flusso in base alla sua capacità discriminante, eliminando così l'influenza di dati irrilevanti o ridondanti, migliorando l'efficienza computazionale e l'accuratezza della classificazione [29].

Reti Neurali

La classificazione del traffico di rete con le reti neurali (NN) si basa su algoritmi supervisionati che apprendono da un insieme di dati etichettati, ovvero dati pre-classificati. L'input di questi modelli consiste in flussi di traffico, i cui attributi principali vengono estratti dagli *header* dei pacchetti, come la loro lunghezza, il tempo di arrivo, la destinazione, e altre caratteristiche statistiche come la varianza e la media del *timing* [5, 50]. Un tipo comune di NN utilizzata per la classificazione del traffico di rete è il *Multilayer Perceptron* (MLP) che permette di modellare relazioni non lineari tra le caratteristiche estratte dai flussi di traffico, migliorando a capacità del modello di classificare correttamente il traffico [42]. Un'altra tipologia di NN utilizzata per la classificazione del traffico sono le reti neurali Bayesiane, che combinano i principi delle reti neurali tradizionali con un approccio probabilistico, migliorando la robustezza della classificazione e la gestione dell'incertezza

di alcuni dati [42, 5].

Algoritmi ensemble

Gli algoritmi *ensemble*, come il *Random Forest* (RF), rappresentano una potente tecnica per la classificazione del traffico di rete. Il RF è un metodo che consiste nella costruzione di una ”foresta” composta da numerosi alberi decisionali, dove ciascun albero viene addestrato su un sottoinsieme casuale dei dati di input. La previsione finale, che determina poi la classificazione dei dati forniti, è data dalla media delle previsioni di tutti gli alberi, riducendo così il rischio di *overfitting*[3] [7, 45]. Il vantaggio principale di RF nella classificazione del traffico di rete risiede nella sua capacità di gestire grandi set di dati senza richiedere una complessa elaborazione. Ogni albero decisionale all'interno della foresta è costruito in modo indipendente e le previsioni sono aggregate per ottenere una decisione finale robusta. Questo approccio parallelo consente al *Random Forest* di mantenere alta l'accuratezza anche con una grande varietà di traffico di rete [45]. Una delle varianti di RF, il metodo *Active Build-Model Random Forest* (ABRF), migliora ulteriormente le prestazioni del modello originale riducendo il tempo di elaborazione. Questo approccio elimina gli alberi decisionali con bassa accuratezza, mantenendo solo quelli ”attivi” che contribuiscono significativamente alla classificazione, senza compromettere l'accuratezza complessiva [45].

4.2.2 Vantaggi e sfide

L'uso delle tecniche di ML per la classificazione del traffico di rete offre numerosi vantaggi, ma comporta anche sfide significative. Una delle principali forze di questi approcci è la capacità di analizzare grandi volumi di dati e di identificare modelli complessi nei flussi di rete senza la necessità di accedere direttamente ai contenuti dei pacchetti. Ciò rende gli algoritmi di ML particolarmente utili in scenari in cui i metodi tradizionali, come la classificazione basata su porta o ispezione dei pacchetti, non sono sufficienti, come nel

[3] Fenomeno in cui un modello di apprendimento automatico si adatta troppo fedelmente ai dati di addestramento, includendo anche rumore e anomalie, perdendo così la capacità di generalizzare correttamente su dati nuovi o non visti.

caso del traffico cifrato o quando le applicazioni utilizzano porte non standard [7]. Tuttavia, l'uso di ML nella classificazione del traffico di rete presenta alcune difficoltà. Una delle principali sfide riguarda la necessità di un ampio dataset di traffico etichettato, che sia rappresentativo di tutte le variabili di traffico che il sistema dovrà affrontare. La qualità e la quantità dei dati disponibili influiscono direttamente sull'efficacia del modello di ML. Inoltre, la selezione e l'estrazione delle caratteristiche sono fasi critiche, poiché una scelta errata o un numero eccessivo di caratteristiche potrebbe ridurre la precisione del modello o aumentare il carico computazionale [7]. Altra sfida importante riguarda l'implementazione in tempo reale. Mentre gli algoritmi come *Random Forest* sono relativamente efficienti nel trattare grandi set di dati, altri metodi, come le reti neurali, richiedono notevoli risorse computazionali, specialmente quando si analizzano flussi di rete complessi o quando il modello deve essere addestrato su larga scala. Questo potrebbe rendere difficile l'adozione in ambienti di rete con risorse limitate o in situazioni che richiedono una classificazione immediata del traffico [28, 7].

5. TECNOLOGIE PER LA PRIVACY E L'ANONIMATO

5.1 Privacy e anonimato

Negli ultimi anni, i concetti di privacy e anonimato hanno rivestito, e rivestono ancora, un ruolo di primaria importanza, poiché mirano a tutelare l'identità e i dati personali degli utenti che navigano sulla rete. Sebbene i due termini vengano talvolta utilizzati come sinonimi, in realtà essi rispondono ad esigenze e finalità distinte.

La privacy concerne la facoltà di controllare la diffusione e l'utilizzo delle proprie informazioni personali, mentre l'anonimato si focalizza sulla possibilità di agire o comunicare in rete senza che l'attività di un utente possa essere ricondotta a un soggetto specifico.

L'incremento costante dei servizi online e la diffusione di dispositivi connessi a Internet hanno favorito la raccolta e l'elaborazione di una mole considerevole di dati personali. Questo fenomeno ha evidenziato la necessità di meccanismi tecnici e giuridici atti a salvaguardare la riservatezza degli individui. In questo contesto, l'anonimato rappresenta una soluzione efficace per prevenire forme di sorveglianza, profilazione eccessiva e altre violazioni della privacy.

5.2 VPN

Le *Virtual Private Networks* (VPN) costituiscono un meccanismo largamente utilizzato per creare ambienti comunicativi sicuri e privati, anche su infrastrutture di rete pubbliche. Mediante l'impiego di tecniche di *tunneling* e crittografia, le VPN permettono di instaurare connessioni sicure tra nodi geograficamente distanti, emulando le caratteristiche di una rete privata isolata. Questo approccio consente di proteggere l'integrità e la riservatezza dei dati trasmessi, minimizzando il rischio di intercettazioni e accessi non autorizzati [65].

Inoltre, le VPN si configurano come reti *overlay* che operano su infrastrutture condivise sfruttando protocolli standardizzati. Tali soluzioni consentono alle organizzazioni di connettere sedi distanti, fornendo un accesso sicuro alle risorse aziendali, o a privati di celare la propria identità [36].

Di seguito verrà esaminato OpenVPN, una tra le implementazioni di VPN più utilizzate. Al fine di comprendere le tecniche di individuazione di OpenVPN, è necessario conoscere nel dettaglio il suo funzionamento: dall'instaurazione della connessione al trasferimento di informazioni.

5.2.1 OpenVPN

OpenVPN è una implementazione *open source* estremamente versatile di una VPN ed è uno tra i protocolli più utilizzati dai *provider* vpn commerciali. Può operare su entrambi i protocolli di trasporto, TCP e UDP, ma il suo funzionamento interno resta invariato. Alla base c'è la separazione della connessione in due canali di comunicazione distinti:

- Canale di controllo: Utilizzato per gestire l'autenticazione, la negoziazione delle chiavi e la configurazione della connessione;
- Canale dati: Utilizzato per il transito effettivo del traffico degli utenti.

TLS 1.3

La protezione del canale di controllo in OpenVPN si basa sul protocollo TLS, che, nella sua versione 1.3, fornisce un canale sicuro e robusto contro intercettazioni, manomissioni e falsificazione dei messaggi [52]. La fase di instaurazione della connessione, nota come TLS *handshake* (Figura 5.1) è costituita dalle seguenti fasi:

1. Client Hello: Il *Client Hello* è il primo messaggio inviato dal client al server, che da inizio all'*handshake*. Questo messaggio contiene una serie di informazioni indispensabili per il server, tra cui:
   (a) Versione TLS: Campo fisso tipicamente impostato a 0x0303, corrispondente a TLS 1.2 ma valido anche per la versione 1.3;

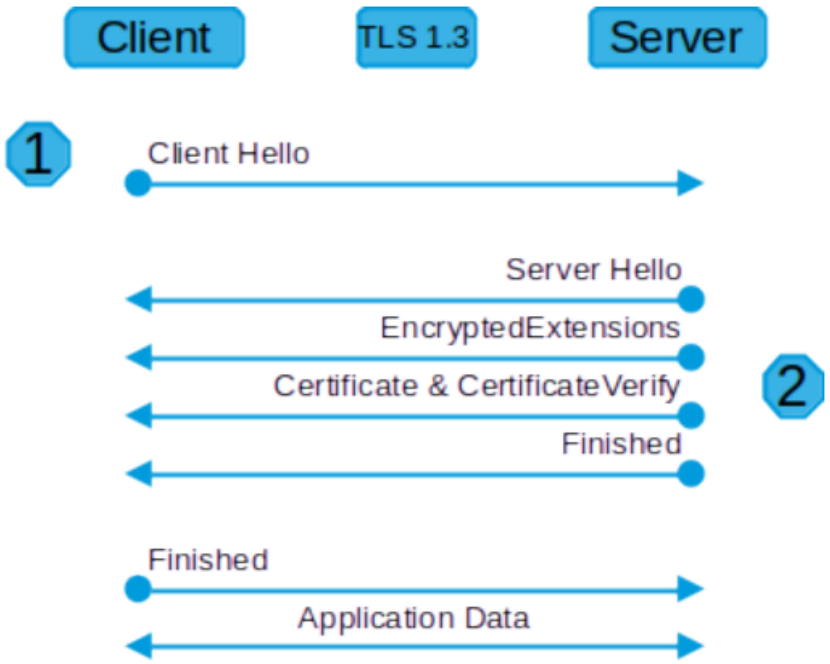

Figura 5.1: TLS Handshake [41]

(b) Valore casuale: Valore di 32 byte, generato in maniera sicura e casuale che verrà utilizzato come seme per la generazione delle chiavi di cifratura;

(c) Lista di Cipher Suites: Campo contenente una lista ordinata (in base alla preferenza del client) di algoritmi per la cifratura e la gestione delle chiavi;

(d) Estensioni: Lista contenente estensioni di vario tipo, a supporto del server per la gestione della connessione;

2. Server Hello: Il server risponde con un messaggio *Server Hello* con il quale conferma la versione da utilizzare, comunica il suo valore casuale e sceglie una *Cipher Suite* da utilizzare, tra quelle elencate dal client.

3. EncryptedExtensions: Questo messaggio cifrato contiene le estensioni che non sono state necessarie per negoziare i parametri crittografici ma che servono a definire ulteriori caratteristiche della sessione (es. parametri di compressione).

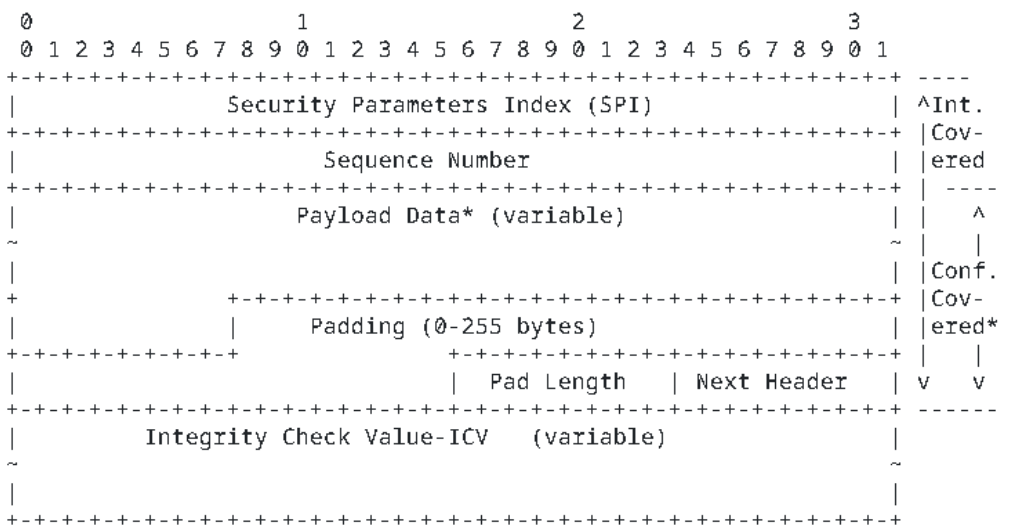

Figura 5.2: Formato ESP [35]

4. Certificate e Certificate Verify: Il messaggio Certificate trasmette la catena di certificati del server, inclusa la sua chiave pubblica, che permette al client di verificare la sua identità; subito dopo, il messaggio *CertificateVerify* contiene una firma, realizzata con la chiave privata corrispondente, che copre l'intero *transcript* dell'*handshake*;

5. Finished: Entrambe le parti si scambiano un messaggio *Finished*, che contiene un MAC (*Message Authentication Code*) cioè un valore calcolato su tutto il *transcript* dell'*handshake*, il quale garantisce l'integrità della comunicazione avvenuta [52].

Protezione dei Pacchetti tramite IPSec

IPSec si occupa della protezione dei pacchetti dati, mediante il protocollo *Encapsulating Security Payload* (ESP). Questo protocollo definisce strutture specifiche per garantire l'integrità, l'autenticazione e la riservatezza della comunicazione. L'*Encapsulating Security Payload* è progettato per garantire confidenzialità delle comunicazioni in quanto assicura la cifratura del *payload* dei pacchetti. Un pacchetto ESP (Figura 5.2), è formato dai seguenti campi:

- Security Parameters Index (SPI): Questo campo server per identificare univocamente la SA[1] a cui il pacchetto è associato;

- Sequence Number: Campo a 32 bit che viene incrementato per ogni pacchetto trasmesso sulla SA, utilizzato per prevenire attacchi di *replay*;

- Payload Data: Campo variabile che contiene i dati originali del pacchetto, i quali devono essere protetti. In modalità tunnel, questo campo comprende l'intero pacchetto IP originale (*header* incluso); in modalità trasporto, contiene solo il *payload* del pacchetto.

- Padding: Questo campo, la cui lunghezza può variare tra 0 e 255 byte, garantisce che la lunghezza totale dei dati da cifrare sia un multiplo della dimensione del blocco richiesta dall'algoritmo di cifratura. Inoltre, può servire per mascherare la lunghezza effettiva del *payload* contribuendo a rendere ancora più riservata la comunicazione;

- Pad Length: Campo a 8 bit che specifica il numero di byte effettivamente utilizzati per il *padding*. Grazie a questo campo, il destinatario potrà rimuovere correttamente il *padding* al momento della decifratura;

- Next Header: Campo di 8 bit che specifica il protocollo del *payload* successivo;

- Integrity Check Value (ICV): Campo variabile, di lunghezza multipla di 32 bit, che contiene il valore di controllo utilizzato per verificare l'integrità del pacchetto. Viene calcolato applicando un algoritmo di hash (MD5, SHA1 o SHA256) sui campi dell'*header* (sia dell'IP che dell'AH) che sono immutabili o prevedibili, impostando a zero il campo ICV stesso durante il calcolo, e includendo eventuale *padding* esplicito necessario per rispettare l'allineamento. Le specifiche dell'algoritmo di hashing utilizzato, definiscono la lunghezza esatta dell'ICV e le regole per il confronto durante la verifica [35].

[1] Security Association (SA): Insieme unidirezionale di parametri crittografici (come algoritmi, chiavi, SPI e durata) condivisi tra due entità, che definisce come autenticare e/o cifrare il traffico tra di esse.

OpenVPN prevede due meccanismi opzionali per rafforzare la sicurezza del canale di controllo: TLS-Auth e TLS-Crypt. Entrambi migliorano la resilienza della connessione contro attacchi attivi e l'accesso non autorizzato, senza alterare nè mascherare i tratti distintivi del traffico OpenVPN

TLS-Auth

TLS-Auth consente di autenticare i pacchetti del canale di controllo ancora prima dell'instaurazione della connessione TLS. Questo avviene attraverso l'utilizzo di una chiave segreta condivisa, generata in fase di configurazione e conosciuta sia dal client che dal server. Tale chiave viene impiegata per calcolare un HMAC (*Hash-based Message Authentication Code*) su ogni pacchetto trasmesso:

$$\mathrm{HMAC}(K,m) = H\Big((K \oplus \mathrm{opad}) \,\|\, H\big((K \oplus \mathrm{ipad}) \,\|\, m\big)\Big)$$

il quale funge da firma. Quando un pacchetto viene ricevuto, il server ne verifica l'autenticità rigenerando l'HMAC e confrontandolo con quello allegato. Solo in caso di corrispondenza, il pacchetto viene considerato valido e processabile.

Sebbene i pacchetti non vengano interamente cifrati, questo meccanismo impedisce ad un server OpenVPN di rispondere a tentativi di connessione provenienti da host non autorizzati, bloccando così attacchi come il *port scanning*, DoS e *active probing* [47].

TLS-Crypt

TLS-Crypt nasce per superare alcune delle limitazioni di TLS-Auth, offrendo una protezione avanzata del canale di controllo, inclusa la fase iniziale dell'*handshake* TLS, che normalmente avviene in chiaro. Questo consente di garantire sia l'autenticazione che la riservatezza, in un'unica soluzione [57, 31].

Il meccanismo prevede l'utilizzo di una chiave simmetrica che viene assegnata a ciascun client in forma cifrata (wrappata) dal server. In fase di connessione, il client invia al server la chiave wrappata e, una volta decifrata, viene utilizzata per proteggere l'intera comunicazione del canale di controllo [31].

Il vantaggio di TLS-Crypt è duplice: da un lato, impedisce a terzi di osservare o alterare i messaggi scambiati nel canale di controllo; dall'altro, riduce la superficie esposta a potenziali interferenze, impedendo l'instaurazione di connessioni da parte di client non in possesso della chiave corretta. Inoltre, a differenza di TLS-Auth, la gestione delle chiavi può essere centralizzata sul server, semplificando la configurazione lato client [62]. Attualmente, TLS-Crypt risulta essere adottato dalla quasi totalità dei *provider* VPN commerciali.

5.3 Proxy

I proxy costituiscono, come le VPN, un meccanismo intermedio nel processo di comunicazione tra client e server, permettendo il reindirizzamento e la manipolazione del traffico. Agendo come punto di passaggio, un proxy riceve le richieste provenienti dal client, le inoltra al server di destinazione e restituisce la risposta al mittente originario, fungendo così da intermediario nel flusso comunicativo.

A differenza delle VPN, i proxy generalmente non instaurano un tunnel completamente cifrato tra i nodi coinvolti nella comunicazione. Di seguito viene esaminato il funzionamento di Shadowsocks, una tra le implementazioni proxy più utilizzate per aggirare blocchi geografici, garantire l'anonimato ed eludere la censura [1].

5.3.1 Shadowsocks

Shadowsocks nasce come un sistema di proxy cifrato basato sul protocollo Socks5 [34]. Originariamente, l'idea con la quale è nato, era quella di semplificare la "navigazione scientifica", ovvero consentire all'utente di aggirare firewall e forme di censura in rete. La semplicità d'uso, la leggerezza e la facilità d'implementazione hanno reso questo protocollo molto diffuso soprattutto in Cina. Uno studio del 2015 condotto presso l'Università di Tsinghua riportava che circa il 21% degli intervistati faceva uso di Shadowsocks per bypassare la censura governativa del GFW [40].

Funzionamento

Per garantire riservatezza, integrità e autenticazione delle comunicazioni, Shadowsocks fa uso di una categoria di meccanismi crittografici, noti come cifrari AEAD (*Authenitcated Encryption with Associated Data*). Questo metodo fa uso di una password condivisa (*Master Password*) nota sia al client che al server, da cui è possibile generare la chiave di sessione con cui verrà cifrato il traffico.

Il flusso di rete generato da Shadowsocks, come mostrato nella Figura 5.3 risulta essere suddiviso in blocchi caratterizzati da una lunghezza pre-indicata ed ogni blocco è cifrato e autenticato grazie ad un tag AEAD. Inoltre, per evitare che la lunghezza del blocco risulti visibile e riconoscibile, tali prefissi vengono a loro volta cifrati e accompagnati da un tag.

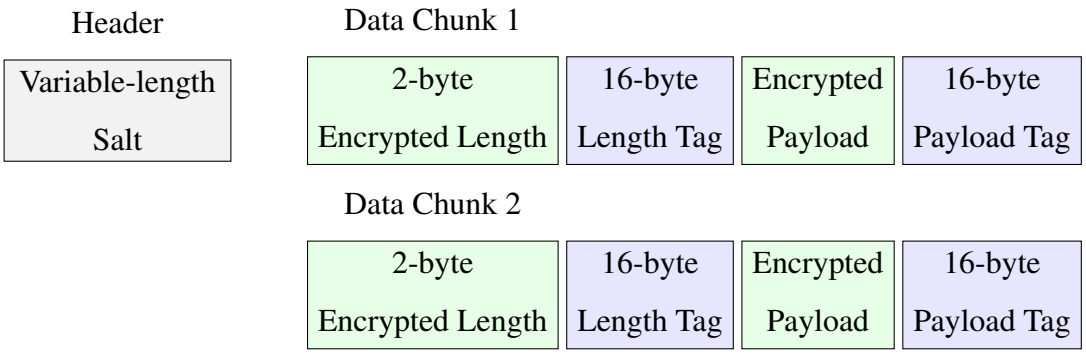

Figura 5.3: Struttura del pacchetto Shadowsocks AEAD.

Tutto il flusso poi, è preceduto da un *salt*, un valore di dimensione variabile (16, 24 o 32 byte) casuale aggiuntivo che, combinato con la *Master Password*, permette di generare una chiave di sessione separata per ciascuna direzione della trasmissione [34].

Grazie a questo potente sistema di cifratura, il traffico risultante appare a tutti gli effetti come un flusso di byte perfettamente casuale e privo di *pattern* sfruttabili. Tuttavia, proprio questa casualità estrema può trasformarsi, come vedremo in seguito, in una vulnerabilità.

6. TECNICHE DI OFFUSCAMENTO

6.1 Offuscamento di OpenVPN

6.1.1 XOR Scramble

La XOR Scramble è una tecnica di offuscamento implementata come *patch* non ufficiale per OpenVPN. La *patch*, mira ad offuscare il contenuto dei pacchetti, rendendo così più complicata l'identificazione del traffico VPN.

Funzionamento

La XOR Patch agisce sul contenuto del pacchetto OpenVPN trasformando il flusso di byte mediante operazioni bitwise [71]. Il processo si articola in tre funzioni principali, implementate nel codice della patch, che operano su un buffer contenente il pacchetto:

- buffer_mask(buf, mask, xormasklen): Questa funzione itera su ogni byte del buffer e applica un'operazione XOR con un byte derivato dalla maschera fornita. La maschera viene utilizzata in modalità ciclica mediante l'uso dell'operazione modulo. Formalmente, per ogni byte in posizione i:

$$\text{buf}[i] = \text{buf}[i] \oplus \text{mask}[i \bmod \text{xormasklen}] \tag{6.1}$$

```
+int buffer_reverse (struct buffer *buf) {
+  int len = BLEN(buf);
+  if (  len > 2  ) {
+    int i;
+    uint8_t *b_start = BPTR (buf) + 1;
+    uint8_t *b_end   = BPTR (buf) + (len - 1);
     .....
```

Figura 6.1: Funzione della Patch XOR con primo byte invariato [71]

- buffer_xorptrpos(buf): Funzione che modifica ciascun byte del buffer effettuando lo xor con un valore derivato dalla sua posizione (indice+1). Formalmente, per ogni byte in posizione i:

$$\text{buf}[i] = b[i] \oplus (i+1) \tag{6.2}$$

- buffer_reverse(buf): Questa funzione inverte l'ordine dei byte del buffer, con la particolarità di escludere il primo byte (contenente l'opcode) dalla trasformazione (Figura 6.1).

L'offuscamento di un pacchetto è quindi dato dall'esecuzione sequenziale di queste quattro funzioni. Nello specifico: buffer_mask, buffer_xorptrpos, buffer_reverse, buffer_xorptrpos.

Sebbene l'utilizzo di questa patch non ufficiale sia fortemente sconsigliato da OpenVPN [63], si è rivelata essere una tecnica utilizzata dalla gran parte dei servizi VPN commerciali [71].

6.1.2 Tecnologie di tunneling per OpenVPN

Il *tunneling* è una tecnica che permette di trasportare un protocollo di comunicazione incapsulato all'interno di un altro protocollo. Questo processo consente il trasferimento di dati, mascherando il protocollo originale sotto un traffico generalmente ritenuto innocuo e largamente utilizzato, come HTTPS, DNS, TLS, ecc. Tale tecnica è ampiamente utilizzata da *provider* VPN commerciali, promettendo l'anonimato e l'irriconoscibilità delle comunicazioni.

Funzionamento

Il *tunneling* opera creando una sorta di "galleria" attraverso cui viaggiano i pacchetti del protocollo originale. Questi pacchetti vengono incapsulati con intestazioni di un protocollo differente, che fungerà da vettore, rendendo il traffico indistinguibile o difficilmente riconoscibile rispetto al protocollo effettivo trasportato. Una volta giunti a destinazione, i pacchetti vengono decapsulati, permettendo la ricostruzione e interpretazione del traffico originale.

Questo processo può avvenire attraverso diverse modalità:

- Encapsulation: il traffico del protocollo originale viene interamente incapsulato all'interno di un altro protocollo;
- Obfuscation: si alterano caratteristiche specifiche del traffico per renderne difficile l'identificazione tramite filtraggio passivo;
- Multiplexing: si mescolano pacchetti di diversi protocolli all'interno dello stesso flusso per complicare l'analisi e l'identificazione del traffico reale.

Applicazione per OpenVPN

Come discuteremo nel capitolo successivo, OpenVPN genera traffico con una firma relativamente riconoscibile dal punto di vista di DPI e DFI. Per mitigare questo problema e superare restrizioni o blocchi, è possibile incapsulare in traffico generato da OpenVPN utilizzando vari strumenti e tecnologie di *tunneling*, tra le più utilizzate abbiamo:

- TLS tunneling (es. Stunnel): È una delle soluzioni più comuni ed efficaci. Incapsula il traffico OpenVPN all'interno di una connessione TLS, apparendo come traffico HTTPS legittimo [58].
- DNS Tunneling e DoH/DoT (DNS over HTTPS/TLS): Queste tecniche sfruttano le query DNS o HTTPS/TLS per trasportare traffico OpenVPN poichè il traffico DNS/HTTPS è comunemente permesso e poco sospetto [59, 33, 2].
- SSLH e Multiplexing di protocolli: utilizza tecniche di *multiplexing* che consentono di instradare connessioni OpenVPN nello stesso flusso di connessioni HTTPS o SSH. La presenza simultanea di traffico di diversi protocolli rende la rilevazione molto complessa [60].

L'impiego di tecniche di *tunneling* come strumento di offuscamento rappresenterebbe, ad oggi, una strategia efficace ed ampiamente utilizzata da molti provider VPN. La scelta della tecnologia di *tunneling* più adatta deve tenere conto di fattori come prestazioni, complessità di configurazione e livello di sicurezza richiesto.

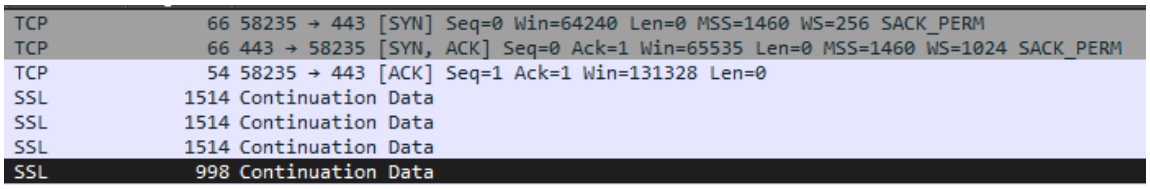

Figura 6.2: Handshake Obfs4 completamente cifrato

6.2 Offuscamento di Shadowsocks

6.2.1 Obfs4

Obfs4 rappresenta uno dei più recenti ed efficaci strumenti di offuscamento (*pluggable transport*) sviluppati nell'ecosistema Tor, un sistema di comunicazione anonima che instrada il traffico degli utenti attraverso una rete globale di nodi volontari, chiamati *relay*, allo scopo di celare tanto l'indirizzo IP sorgente quanto la destinazione finale [51].

Handshake

Un tratto distintivo di obfs4 è il suo *handshake* interamente crittografato. Infatti, sin dai primissimi pacchetti, il protocollo non include intestazioni testuali o byte di riconoscimento in chiaro, ma campi pseudocasuali e valori derivati da algoritmi di cifratura. Questo implica che non è possibile ottenere alcuna informazione utile all'identificazione del protocollo semplicemente esaminando la fase di *handshake* [4].

La Figura 6.2, mostra la fase iniziale di una connessione TCP verso un *bridge* obfs4 in ascolto sulla porta 443. Si noti anzitutto il consueto *3-way handshake* a livello TCP. Successivamente, i pacchetti sono etichettati come "*SSL Continuation Data*": si tratta di dati cifrati che Wireshark, utilizzato per la cattura dei pacchetti, non riesce a interpretare come un normale handshake TLS in quanto mancano i tipici messaggi "ClientHello" e "ServerHello", perché obfs4 avvia la cifratura fin dal primo pacchetto. Questo comportamento differenzia un flusso obfs4 da un convenzionale flusso TLS.

Anti Probing

Un altro elemento chiave di obfs4 è la sua caratteristica *probe-resistant* o *anti-active probing*. Obfs4 contrasta tentativi di probing attivo non rispondendo alle richieste o rispondendo con dati randomici, non lasciando trapelare alcun indizio utile per l'identificazione. Tuttavia, alcuni studi condotti in merito a questo mostrano come obfs4 non sia del tutto immune al rilevamento. Infatti, sarebbe possibile sfruttare proprio la mancata risposta con conseguente timeout della connessione per distinguere obfs4 da altri server legittimi [24].

Padding e Random Fragmentation

Una volta stabilito il canale cifrato, obfs4 ricorre a *padding* e *random fragmentation* per camuffare eventuali correlazioni tra i pacchetti inviati ed il loro contenuto. Il risultato finale è un flusso di byte che appare estremamente casuale, di dimensioni non ricorrenti, perciò privo di firme sfruttabili per l'identificazione. Ciò rende molto complesso, per un sistema di rilevamento, distinguere il traffico obfs4 da altri flussi cifrati [4].

6.2.2 V2Ray Plugin

V2Ray plugin è un'estensione progettata per integrare le funzionalità di V2Ray all'interno di Shadowsocks. In linea generale, il suo approccio all'offuscamento non si discosta troppo da quello di tecnologie come obfs4, se non per la capacità di simulare molteplici protocolli. È proprio questa particolare abilità a consentire di simulare i comportamenti dei principali protocolli di rete.

Simulazione di protocolli multipli

La simulazione di protocolli costituisce uno degli aspetti più interessanti introdotti dal V2Ray plugin, poichè non si limita a offuscare il traffico Shadowsocks, ma cerca di fonderlo facendolo apparire sotto forme di comunicazione comunemente riconosciute come legittime. In sostanza, V2Ray plugin adatta e riplasma i pacchetti Shadowoscks in modo che appaiano come se provenissero da protocolli come TLS, WebSocket, HTTP/2 o QUIC.

Naturalmente, per funzionare in maniera convincente, questa simulazione deve essere completa, o quanto meno abbastanza accurata, così da non innescare meccanismi di allerta nei sistemi di rilevamento. Ecco perché V2Ray non si limita a generare un banale falso *header*: si preoccupa anche di replicare correttamente le peculiarità e le sequenze di scambio dei pacchetti, l'ordine e la struttura dei *frame*, le risposte di errore plausibili, e così via.

Quando si passa a protocolli come HTTP/2 o QUIC, per esempio, si curano aspetti come i *settings frame*, la gestione delle *stream ID*, la segmentazione, oppure, nel caso di QUIC, la presenza di un *handshake* crittograficamente distinto, basato su TLS 1.3 integrato direttamente nel protocollo. Tutti dettagli che, se mancanti o realizzati in modo troppo superficiale, potrebbero essere notati da un DPI ben progettato, sollevando sospetti e portando al blocco.

Questa peculiarità renderebbe quindi, il traffico Shadwosocks difficilmente distinguibile da altri protocolli legittimi, rendendo inefficaci una gran parte dei sistemi di rilevamento basati sul filtraggio passivo del traffico [53].

7. TECNICHE DI RILEVAZIONE

7.1 Rilevazione del traffico OpenVPN

La rilevazione del traffico OpenVPN è un ambito di ricerca in fase embrionale, con un numero estremamente limitato di studi sistematici in letteratura. Attualmente, la fonte principale in questo campo è rappresentata dallo studio di Xue et al. [71], che ha presentato un *framework* per l'identificazione e la classificazione dei flussi OpenVPN. Tale *framework*, mostrato in Figura 7.1 è strutturato in due fasi complementari: una fase passiva di filtraggio del traffico, che combina tecniche basate sul rilevamento degli *ACK* e degli *opcode*, e una fase di *probing* attivo, la quale viene attivata per confermare in maniera dinamica i sospetti generati dalla prima fase.

Il modulo di filtraggio passivo si basa sull'analisi di caratteristiche intrinseche del protocollo, come il comportamento dei pacchetti di riconoscimento (ACK) e la presenza di *opcode* non criptati nell'*handshake*, al fine di identificare *pattern* che siano tipici delle sessioni OpenVPN. Tuttavia, a causa della scarsità di ricerche specifiche, i risultati ottenuti dall'applicazione di ciascuna delle tecniche individuali non sono ancora stati quantificati in modo separato. È noto, invece, che l'intero *framework* implementato da Xue et al. ha mostrato risultati promettenti.

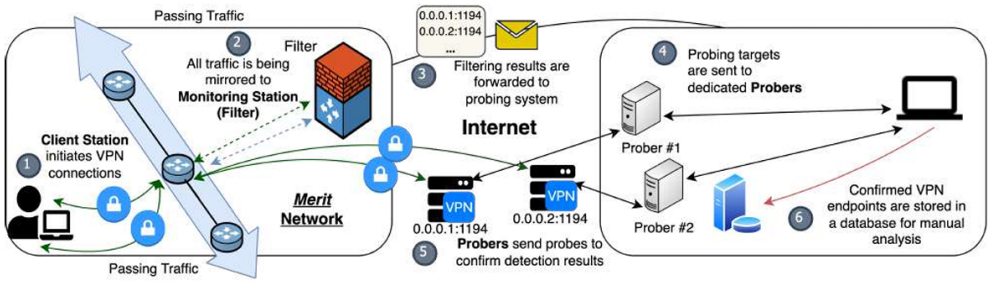

Figura 7.1: Framework per l'identificazione di traffico OpenVPN [71]

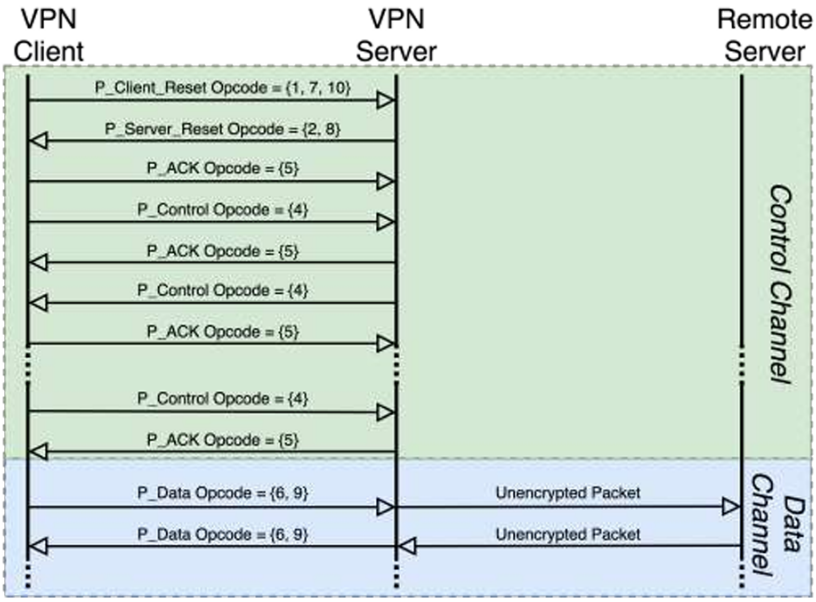

Figura 7.2: Opcode generati durante l'instauramento di una connessione OpenVPN [71]

7.1.1 Rilevamento basato su Opcode

L'identificazione di flussi OpenVPN mediante l'analisi degli *Opcode* rappresenta uno dei metodi più promettenti per il rilevamento passivo di connessione VPN. Ogni pacchetto OpenVPN, in particolare durante la fase di *handshake*, contiene un *header* non criptato, composto da un campo *opcode* e un campo *keyID*. Questi *opcode* definiscono il tipo di messaggio trasmesso ed il loro valore cambia in base alla fase in cui si trova la comunicazione. Lo studio ha identificato gli *opcode* tipici di un *handshake* OpenVPN (Figura 7.2).

Questa tecnica si fonda sull'osservazione che, nonostante le tecniche di offuscamento basate su *XOR* (6.1.1) possano alterare il contenuto dei pacchetti, esse escludono deliberatamente il primo byte del *payload* dall'operazione di inversione. Di conseguenza, l'*opcode* resta invariato, pur essendo sottoposto a un'operazione di *XOR* con chiave fissa, garantendo quindi che il valore dell'*opcode* rimanga mappato in maniera consistente, rendendone possibile l'identificazione e la raccolta dinamica delle variazioni del suo valore durante la fase di *handshake* [71].

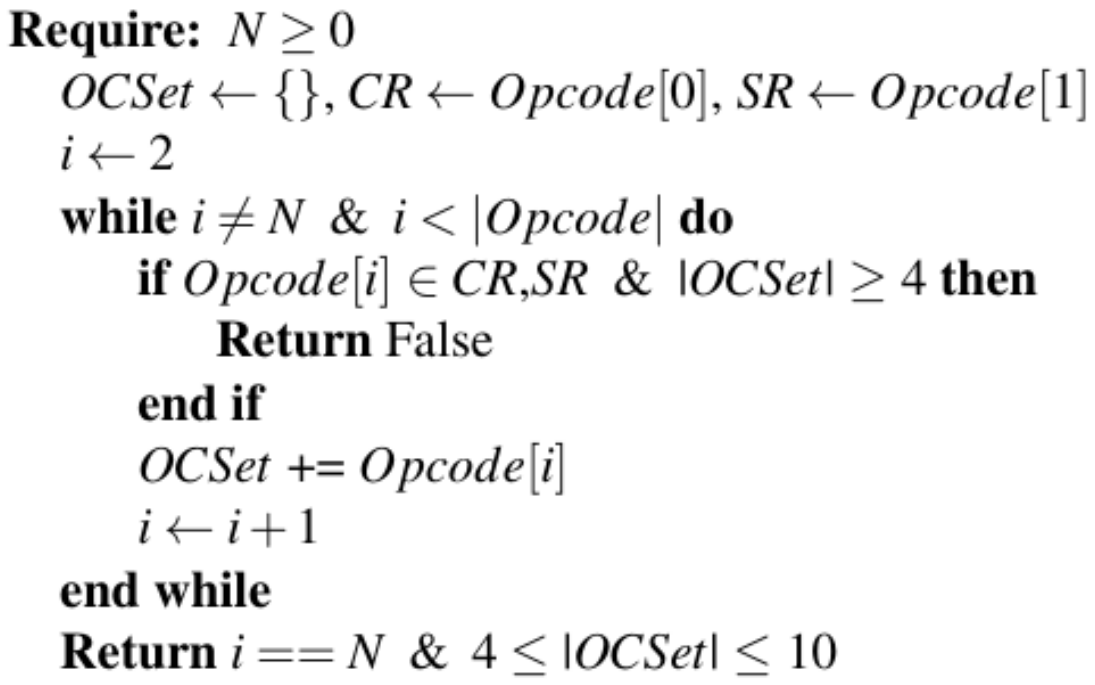

Figura 7.3: Algoritmo Tecnica basata su Opcode

Algoritmo

L'algoritmo, relativamente leggero in termini computazionali, esegue una scansione sequenziale del campo *opcode* per i primi N pacchetti della scansione. In sintesi, il processo prevede:

1. L'inizializzazione di un insieme *OCSet* e la registrazione degli *opcode* dei pacchetti iniziali (quelli relativi a Client Reset e Server Reset vengono salvati rispettivamente in CR e SR)

2. Una scansione successiva che verifica, per ogni pacchetto, se l'opcode rientra nei valori attesi (CR, SR) e, in caso affermativo, verifica se l'insieme OCSet raggiunge almeno quattro elementi distinti

3. Al termine dell'analisi degli N pacchetti, il flusso viene classificato come OpenVPN se il numero di *opcode* unici rientra in un intervallo prestabilito (da 4 a 10)

7.1.2 Rilevamento basato su ACK

Il rilevamento basato su ACK rappresenta un approccio innovativo per l'identificazione di connessioni OpenVPN. Questo approccio sfrutta la fase di *handshake* in stile TLS che OpenVPN utilizza per lo scambio di chiavi e la negoziazione dei parametri di cifratura. La fase di *handshake*, spiegata nella sezione 5.2.1 si traduce nell'invio di pacchetti P_ACK per confermare la ricezione dei pacchetti di controllo (P_Control) inviati dal *peer*.

Algoritmo

L'approccio proposto prevede l'identificazione di un pacchetto ACK tipico all'interno del flusso, generalmente il terzo pacchetto trasmesso. Una volta identificato questo pacchetto, il flusso viene suddiviso in intervalli (o "*bin*") di 10 pacchetti ciascuno. Successivamente si procede al conteggio del numero di pacchetti in ciascun *bin* che corrispondono, in termini di dimensione, al pacchetto ACK identificato.

L'ipotesi alla base di questo algoritmo è che, nei flussi OpenVPN, i primi *bin* contengono un'elevata frequenza di pacchetti di riconoscimento (ACK), mentre nei successivi *bin* tali pacchetti sono quasi del tutto assenti. Tale *pattern* distintivo, che si basa sia sul conteggio che sul *timing* dei pacchetti, permette di differenziare efficacemente le connessioni OpenVPN da altri flussi di traffico che presentano caratteristiche più eterogenee e variabili.

Limitazioni

A seguito di esperimenti mirati a validare questa tecnica (Sezione 8.1) ho scoperto che, contrariamente a quanto sostenuto nello studio [71], questa tecnica non utilizza un'euristica solida che permetta di individuare il traffico OpenVPN senza rischiare un alto FPR o FNR, anche configurazioni in cui il traffico OpenVPN è ulteriormente incapsulato in tunnel crittografati che non aggiungono *padding* casuale (come ad esempio con l'utilizzo di TLS-Crypt o Stunnel), la tecnica in questione perde la sua efficacia.

7.1.3 Tecniche basate sul Machine Learning

La classificazione del traffico di rete e, in particolare, l'identificazione del traffico VPN rivestono un ruolo sempre più cruciale [25, 8]. Le tecniche di crittografia diffuse, in alcuni casi, renderebbero le metodologie tradizionali basate su DPI o sulle firme, meno efficaci [25, 20, 67]. Di conseguenza, la ricerca si sta via via orientando verso l'impiego di algoritmi di *machine learning* come mezzo per affrontare questa sfida, sfruttando caratteristiche comportamentali o statistiche del traffico piuttosto che il suo contenuto [8].

Sono stati proposti ed esplorati diversi approcci basati sul *machine learning* tra le fonti esaminate. Un filone di ricerca si concentra sull'estrazione manuale o semi-automatica di *feature* specifiche, spesso legate al tempo, alle dimensioni dei pacchetti o alle statistiche di flusso, da utilizzare come input per i classificatori [26, 43, 20]. In uno studio pubblicato da Drapper et al. [20] sono state investigate l'efficacia di *feature* temporali per la caratterizzazione del traffico crittografato e il rilevamento VPN, impiegando algoritmi come C5.4 e KNN, con risultati che hanno mostrato accuratezza superiore all'80% [20]. In uno studio comparativo, invece, sono stati valutati diversi algoritmi di ML supervisionato (come regressione logistica, SVM, Naive Bayes, KNN, *Random Forest* e *Gradient Boosting Tree*) per la classificazione del traffico VPN e non-VPN, identificando modelli ottimizzati come *Random Forest* e GBT in grado di raggiungere oltre il 90% di accuratezza utilizzando *feature* temporali [8]. In un altro studio più recente di Miller et al., invece, è stato dimostrata l'efficacia di una rete neurale di tipo *multilayer perceptron* addestrata su *feature* basate sui flussi TCP, incluse statistiche temporali, per distinguere il traffico proveniente da connessione OpenVPN rispetto a quello normale, raggiungendo accuratezze del 92% e 93% rispettivamente per il traffico VPN e non-VPN [43], mentre un altro studio ancora più recente, condotto da Gao et al. propone come *feature* efficace per l'identificazione di traffico VPN offuscato, la sequenza della lunghezza dei pacchetti o dei *payload* e classificatori ottenuti con algoritmi di tipo Random Forest con accuratezza fino al 98.86% e F1-score fino al 98,7% [26].

In contrasto con gli approcci basati su feature ingegnerizzate manualmente, l'applicazione del *deep learning* sta prendendo sempre più piede in questo contesto, data la sua

capacità di apprendere automaticamente rappresentazioni gerarchiche direttamente da dati grezzi o da rappresentazioni a basso livello [39, 25, 67]. Il framework ”Deep Packet” utilizza reti neurali profonde come *stacked autoencoder* (SAE) e *convolutional neural networks* (CNN) per integrare le fasi di estrazione delle feature e classificazione, dimostrando la capacità di identificare il traffico crittografato e distinguere tra VPN e non-VPN [39]. Con le CNN, Deep Packet ha raggiunto risultati notevoli, si parla di un *recall* di 0.98 nell'identificazione delle applicazioni e 0.94 nella caratterizzazione del traffico, superando i metodi precedenti sullo stesso *dataset* (ISCX VPN-nonVPN) [39]. Un altro approccio, propone l'uso di reti neurali convoluzionali unidimensionali (1D-CNN) direttamente sul traffico grezzo (rappresentato dai primi byte di ciascuna sessione). Questo metodo pare abbia ottenuto miglioramenti significativi rispetto allo stato dell'arte, superando i metodi comparati in 11 su 12 metriche di valutazione e validando l'efficacia dell'approccio *end-to-end* nell'apprende *feature* più rappresentative [67]. Un altro studio, si propone invece di mitigare il problema della privacy rispetto ai dati grezzi forniti ai modelli durante la fase di apprendimento. Questo studio, infatti, propone un modello NSA-Net (*NetFlow Sequence Attention Network*), il quale impiega una rete LSTM bidirezionale. Questo sistema ha dimostrato un'elevata capacità di rilevamento del traffico VPN, raggiungendo circa il 98.7% di TPR e si è mostrato efficace anche a bassi tassi di campionamento dei dati [25].

In sintesi, le fonti dimostrano che sia gli approcci basati su *feature* ingegnerizzate (in particolare quelle temporali o basate sulla lunghezza dei pacchetti/payload), sia, soprattutto, le metodologie basate sul *deep learning* (come CNN, SAE, LSTM) si sono rivelate altamente efficaci per la classificazione di rete e l'identificazione del traffico VPN su *dataset* standard come ISCX VPN-nonVPN. Le tecniche di *deep learning*, in particolare gli approcci end-to-end o quelli che sfruttano dati aggregati come NetFlow, offrono il vantaggio di automatizzare l'estrazione delle *feature* e possono mitigare i problemi legati alla privacy o alla gestione di grandi volumi di traffico grezzo, raggiungendo al contempo prestazioni di classificazione molto elevate. Nonostante l'elevata efficienza dimostrata, alcune sfide future includono la gestione di dati sbilanciati, l'identificazione di traffico ”sconosciuto” e l'implementazione di questi algoritmi in ambienti di rete ad alta velocità.

7.1.4 Active Probing di OpenVPN

Il meccanismo di *active probing* per il rilevamento di flussi OpenVPN rappresenta una componente fondamentale di un *framework* di rilevamento. Questo meccanismo agisce come una fase di conferma, successiva a una fase iniziale di filtraggio passivo del traffico. Il suo scopo è quello di ridurre drasticamente il tasso di falsi positivi, garantendo che solo le connessioni che sono effettivamente OpenVPN vengano identificate e bloccate, con un danno collaterale trascurabile.

In questa sezione, verrà esaminata nel dettaglio la tecnica di individuazione basata su *active probing*, illustrata da Xue et al. [71], che sfrutta comportamenti specifici del protocollo OpenVPN a livello di gestione delle connessioni TCP per individuarne i flussi.

Funzionamento

Il funzionamento alla base di questa tecnica è semplice quanto efficace. Il *Prober*, componente responsabile del sondaggio attivo, invia sonde (pacchetti) ai server sospetti OpenVPN identificati dal filtro. Le sonde sono attentamente progettate per scatenare comportamenti specifici nei server OpenVPN, e risultano efficaci anche con server in modalità TLS-Auth o TLS-Crypt, che aggiungono una firma HMAC ai pacchetti del canale di controllo e rendono il server "*probe-resistant*" non rispondendo alle richieste non autenticate.

La peculiarità del *Prober* progettato nello studio [71] è che, al fine di rendere efficace questa tecnica anche verso server "*probe-resistant*", invia un totale di 10 probe ben studiati e analizza la non risposta da parte del server.

Infatti, anche se il server non risponde alle sonde inviate, quest'ultimo chiuderà la connessione in tempi differenti, permettendo di ricavare un *pattern* di comportamento che permetterà al *Prober* di comprenderne la natura del servizio in esecuzione.

Il modo in cui un server OpenVPN può chiudere una connessione, può essere di due tipi:

- Short Close (Chiusura breve): Se il server OpenVPN ha una protezione con HMAC abilitato, nel momento in cui riceverà una sonda di lunghezza completa (16 byte)

tenterà di individuare un'HMAC valida, ma, poichè la sonda non è autenticata, la connessione viene immediatamente interrotta.

- Long Close (Chiusura lunga): Se la sonda inviata fosse più corta di 16 byte, il server attenderebbe i byte mancanti per poter ricostruire il messaggio. Il server chiuderà la connessione dopo un certo *timeout* di *handshake* specifico del server.

La combinazione di *Short Close* e *Long Close* in risposta all'invio di particolari sonde, permette di definire un *pattern* comportamentale univoco di un server OpenVPN. I dettagli implementativi di questa tecnica sono riportati nella Sezione 8.3

Limitazioni

Sebbene questo metodo di rilevamento si sia dimostrato efficace ad individuare server OpenVPN sia nel *framework* realizzato da Xue et al. [71] sia nella fase di validazione (Sezione 8.3), a differenza di tecniche di analisi passiva del traffico, l'invio di pacchetti malformati verso server per indurli in errore possono risultare molto invasivi in quanto, se fatto su larga scala e con un numero elevato di sonde, possono saturare la rete o portare a malfunzionamenti dei server oggetto di *probing*. Pertanto è consigliato un utilizzo limitato di questa tecnica [46].

7.2 Identificazione di traffico Shadowsocks

In questa sezione si affronta l'analisi delle tecniche più promettenti ed efficaci per il rilevamento di protocolli altamente offuscati, con particolare riferimento a Shadowsocks. Le fonti presenti in circolazione, prendono come modello di riferimento il GFW e mirano a comprendere le tecniche alla base del funzionamento di un sistema così complesso quanto efficace.

In questa sezione si propone di analizzare in maniera dettagliata le metodologie adottate, distinguendo due principali approcci: l'analisi passiva basata su metriche di entropia e il sistema di *active probing*. Mentre la prima strategia si concentra sull'osservazione dei

pattern statistici nel *payload* dei pacchetti iniziali, l'*active probing* si affida all'invio di probe mirate per verificare la natura sospetta di una connessione.

Saranno quindi esaminati nel dettaglio i meccanismi di rilevazione, le formule utilizzate per il calcolo delle teoriche e le modalità operative che consentono di bloccare il traffico di Shadowsocks.

7.2.1 Calcolo dell'entropia

In questo paragrafo viene illustrata nel dettaglio la metodologia impiegata dal GFW per identificare traffico cifrato generato da Shadowsocks basandosi su test di entropia, e in particolare sull'analisi della distribuzione dei bit nel *payload* del primo pacchetto dati di una connessione.

Fondamenti teorici e metriche utilizzate

L'idea alla base del test dell'entropia è che i flussi completamente cifrati, come quelli generati da Shadowsock, dovrebbero apparire come sequenze casuali di byte. In una sequenza casuale, ogni byte ha una probabilità uniforme di assumere ciascuno dei 256 possibili valori, e di conseguenza il numero medio di bit con valore 1 in un byte dovrebbe avvicinarsi a 4.

La metrica adottata si basa su una funzione di "*popcount*" che, per un dato byte, conta il numero di bit impostati a 1. Formalmente, per un byte b avremo:

$$\text{popcount}(b) = \sum_{i=0}^{7} 1\{b_i = 1\} \tag{7.1}$$

dove $\mathbf{b}_i$ rappresenta l'i-esimo bit del byte e $\mathbf{1}_{\{b_i=1\}}$ è la funzione indicatrice che vale 1 se $b_i = 1$ o 0 altrimenti. In un *payload* di n byte, la media del numero di bit attivi è data da:

$$\overline{B} = \frac{1}{n} \sum_{j=1}^{n} \text{popcount}(b_j)$$

Nel caso di dati perfettamente randomizzati, ci si aspetta che $\overline{B} \approx 4$. Questo valore, confermato da diversi test condotti da un client in China [70], rappresenta un punto di equilibrio e deviazioni significative di questo valore possono indicare anomalie che, in assenza di

ulteriori informazioni, il sistema interpreta come potenzialmente riconducibili a traffico Shadowsocks. Nello specifico, si è compreso che per il GFW è condizione necessaria ma non sufficiente, che $3.4 \leq \overline{B} \leq 4.6$ [70].

7.2.2 Tecniche basate sul Machine Learning

La maggior parte degli studi basati sull'identificazione del traffico Shadowsocks (o più in generale proxy) che utilizzano *machine learning* sono concentrati sull'estrazione manuale di caratteristiche statistiche dai flussi di rete [30, 16]. Queste caratteristiche includono metriche come il numero totale dei pacchetti, il numero di pacchetti in uscita o in entrata di un flusso, la lunghezza massima o media dei *burst*, o l'analisi del comportamento del traffico DNS o dell'host [30, 18]. In uno studio di Zeng et al. [76] è stato dimostrato come, utilizzando caratteristiche estratte sia dal traffico di rete stesso che dal comportamento DNS dell'host e delle relazioni dei flussi, attraverso l'utilizzo dell'algoritmo *Random Forest* sia possibile identificare traffico Shadowsocks. Un altro studio invece, ha dimostrato l'efficacia di un utilizzo combinato di *active probing* e XGBoost per un'identificazione efficace e leggera in termini computazionali [15]. Tuttavia, una criticità intrinseca di queste tecniche risiede nella necessità di una complessa e laboriosa ingegnerizzazione manuale delle caratteristiche. Questo processo richiede tanto lavoro e influenza significativamente l'accuratezza del modello, limitandone la generalizzabilità [73]. Inoltre, l'estrazione di alcune caratteristiche statistiche richiederebbe l'osservazione di un'ampia porzione se non dell'intero flusso, rendendo questi metodi poco applicabili per l'analisi in tempo reale e l'utilizzo di molte caratteristiche finalizzate ad una maggiore precisione della classificazione del traffico può aumentare il carico computazionale e limitarne la scalabilità in ambienti di rete di grandi dimensioni [16].

Per superare queste limitazioni, la ricerca si sta sempre più orientando verso metodi basati sul *Deep Learning*, grazie alla loro capacità di apprendere automaticamente le caratteristiche direttamente dai dati grezzi [73]. Un approccio comune in questi metodi, consiste nel trasformare i dati di traffico in formati compatibili con i modelli, spesso rappresentandoli come immagini [39]. Per esempio, l'approccio descritto da Shapira et al. consiste nel trasformare coppie di parametri formati da tempo di arrivo e dimensione di

pacchetti di un flusso, in immagini per poi classificarle con l'utilizzo di una rete neurale convoluzionale (CNN) [54]. Tuttavia le immagini generate possono essere di grandi dimensioni e possono richiedere eccessiva potenza computazionale per essere interpretate. Per questo motivo, un altro studio propone un approccio più leggero, introducendo un processo di compressione delle immagini fino al 90% senza degradamento delle prestazioni [30].

7.2.3 Active Probing di Shadowsocks

I meccanismi di aggiramento della censura, come Shadowsocks e obfs4, sono stati sviluppati per resistere a tecniche di rilevamento basate sul *active probing*. Tuttavia, in risposta al crescente utilizzo di misure attive da parte di sistemi di censura come il GFW, studi recenti hanno dimostrato che è possibile identificare questi proxy anche se "*probe-resistant*". In particolare, uno studio condotto da Frolov et al. [23] ha dimostrato che, sfruttando il comportamento TCP di questi server, come la modalità e il tempo di chiusura della connessione, sarebbe possibile definire metriche di *detection* molto efficaci.

Strategia

Il metodo proposto si basa su due osservazioni fondamentali:

- Assenza di risposta: La maggior parte dei proxy progettati per essere *probe-resistant* (quali Shadowsocks e obfs4), per evitare di rivelare informazioni che potrebbero rivelare la loro natura, non invia alcun dato in risposta ai probe. Tale comportamento, si differenzia dalla maggior parte dei server legittimi, che generano errori o risposte applicative quando ricevono pacchetti non conformi.

- Comportamento di chiusura della connessione: La chiusura della connessione avviene con modalità specifiche, misurabili in termini di "*close threshold*". Queste metriche includono il FIN *threshold*, ovvero il numero minimo di byte necessari per far scattare una chiusura regolare della connessione (mediante FIN/ACK), e l'RST *threshold*, il quale indica il quantitativo di byte oltre il quale il server chiude la connessione forzatamente (mediante RST).

L'*active probing* viene quindi implementato inviando una serie di probe di vario tipo: da *probe* standard come richieste HTTP o TLS, a *probe* composti da byte casuali con lunghezze specifiche, catturando quindi il comportamento di chiusura della connessione.

FIN e RST Threshold

Con la collaborazione di sviluppatori di obfs4 i quali hanno fornito dei server di test, e la creazione di appositi server Shadowsocks, i ricercatori hanno elaborato uno strumento di *threshorld detection* che, tramite un approccio di ricerca binaria, ha permesso di determinare il numero di byte necessari affinchè un server chiuda la connessione. I risultati ottenuti, confermati anche da analisi manuale, sono riassunti nella tabella 7.1.

Proxy	**Connection Timeout**	**FIN Threshold**	**RST Threshold**
obfs4	60–180 secondi	8–16 KB	determinato dalla successiva lettura in buffer multipli di 1448 bytes
Shadowsocks	configurabile	50 bytes	51 bytes

Tabella 7.1: Timeout e threshold di chiusura per proxy probe-resistant [23]

Decision Trees

Utilizzando i dati raccolti, è stato possibile definire alberi decisionali che permettano di distinguere, attraverso metriche rilevabili tramite invio di *probes*, tra i comportamenti dei proxy e quelli di server legittimi. In particolare:

- Per obfs4: Il *decision tree* (Figura 7.4) prevede che un probe di 17KB provochi una chiusura tramite RST in meno di 2 secondi, mentre i probe inferiori generino una chiusura tramite FIN in un intervallo temporale compreso tra 30 e 90 secondi. Solo i server che rispettano entrambe le condizioni sono etichettati come obfs4.
- Per Shadowsocks: Il *decision tree* (Figura 7.5) prevede che tutti i probe con dimensione superiore a 50 byte causino chiusura della connessione tramite FIN o RST in meno di 2 secondi, mentre probe più brevi portino a un *Timeout*. Solo i server che rispettano entrambe le condizioni sono etichettati come Shadowsocks.

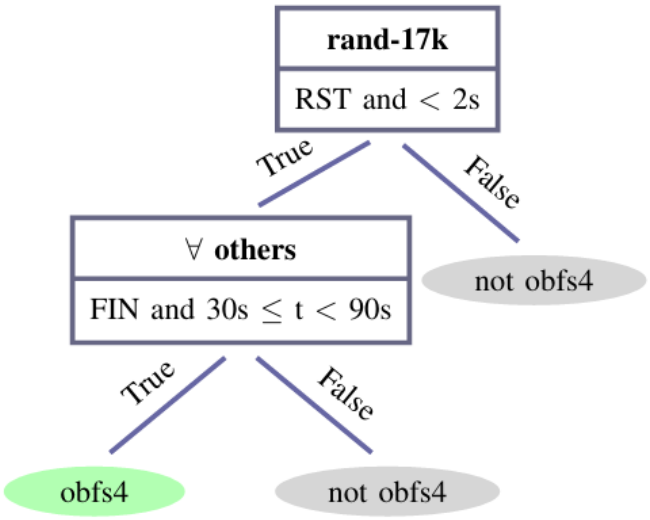

Figura 7.4: Decision Tree di obfs4 [23]

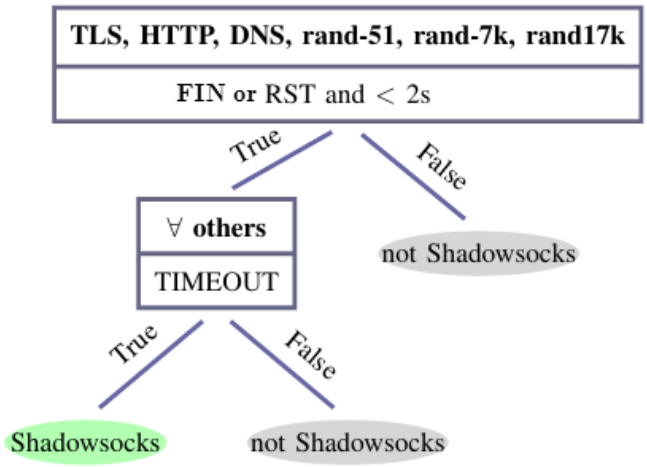

Figura 7.5: Decision Tree di Shadowsocks [23]

Risultati

Per validare l'efficacia degli alberi decisionali costruiti per identificare server Shadowsocks e obfs4, i ricercatori hanno applicato i modelli a due *dataset*: Tap, contenente circa 433.000 *endpoint* raccolti in maniera passiva e Zmap contenente 1.5 milioni di indirizzi IP ottenuti tramite scansione su porte TCP comuni. Tra questi *endpoint*, sono stati scelti solo quelli che condividevano una soglia di chiusura compatibile con obfs4 o Shadowsocks.

I risultati ottenuti mostrano una fortissima capacità selettiva del metodo:

- Per obfs4: su 355 *endpoint* del *dataset* Tap che condividevano una soglia di chiusura compatibile, solo 2 sono stati etichettati come server obfs4, mentre nessuno dei 65 *endpoint* candidati del *dataset* ZMap ha superato i controlli del *decision tree*.
- Per Shadowsocks: su 30 possibili *endpoint* candidati del *dataset* Tap nessuno è stato classificato come possibile server Shadowsocks, mentre tra i 18 possibili candidati del dataset ZMap, 8 sono stati etichettati come possibili server Shadowsocks.

I risultati ottenuti, sono poi stati confermati da analisi manuali, determinando che la precisione del metodo risulta molto elevata. I falsi positivi sono risultati trascurabili: il tasso di classificazioni errate è risultato inferiore allo 0,001%.

In conclusione, l'applicazione dei *decision tree* a contesti reali ha dimostrato che, nonostante l'assenza di risposte esplicite da parte dei proxy *probe-resistant*, la loro "silenziosità" e la gestione specifica delle soglie di chiusura costituiscono, paradossalmente, delle impronte rilevabili. Le metriche come il FIN *threshold* e l'RST *threshold*, combinate con il comportamento nei tempi di chiusura, sono risultate sufficienti per distinguere con precisione server Shadowsocks da server comuni, e in alcuni casi anche server obfs4, pur con numeri molto ridotti. Il metodo, quindi, si conferma efficace e potenzialmente praticabile da un attore censorio, grazie al suo elevato TPR e a un FPR trascurabile.

7.3 Confronto tra i metodi

7.3.1 OpenVPN

Tecnica di rilevamento	**Caso d'uso principale**	**Efficacia con Xor Scramble**	**Efficacia con Tunnel Crittografati**
Basata su Opcode	Funziona bene su OpenVPN senza offuscamenti ma comunque efficace con offuscamenti Xor. Si applica all'handshake iniziale	Alta - Il primo byte del payload non è interessato dal mescolamento	Inefficace - Gli opcode non sono più in chiaro
Basato su ACK e *pattern* temporali	Inefficace - I risultati della validazione sono riportati nella Sezione 8.1		
Active Probing	Particolarmente efficace se il server non filtra correttamente i pacchetti in ingresso	Alta - Questo metodo risulta immune da offuscamento con Xor	Alta - anche per meccanismi di Tunneling che adottano sistemi anti-probing

Tabella 7.2: Confronto metodi e contromisure

7.3.2 Shadowsocks

Tecnica di rilevamento	**Caso d'uso principale**	**Efficacia con Shadowsocks "puro"**	**Efficacia con Obfs4**	**Efficacia con V2Ray**
Test di entropia	Rilevamento passivo del grado di casualità del payload del primo pacchetto	Alta – entropia ≈ 4 rende sospetto il traffico	Bassa – traffico indistinguibile da TLS 1.3 o QUIC	Bassa – headers camuffati, padding variabile e frame criptati mascherano bene il traffico
Active probing	Invio di pacchetti manipolati o casuali per stimolare risposte tipiche da server Shadowsocks	Alta – risposte identificabili e chiusure di connessione prevedibili	Alta – silenzio tipico sospetto e chiusure di connessione prevedibili	Sconosciuta – comportamento personalizzabile, può ignorare, rispondere con dati neutri o simulare HTTP
Machine Learning	Classificazione tramite *pattern* temporali e statistici su traffico cifrato indistinto	Raramente necessario – il traffico è già rilevabile con metodi più semplici	Minaccia emergente – richiede modelli ben addestrati	Variabile – può resistere se configurato per variare header e comportamento del flusso

Tabella 7.3: Confronto tra metodi di rilevamento e tecniche di offuscamento nei proxy

8. VALIDAZIONE DELLE TECNICHE DI IDENTIFICAZIONE DI OPENVPN

8.1 Idenficazione basata sull'ACK

Una delle due componenti del filtro utilizzato nello studio di Xue et al. è data dall'analisi basata sull'ACK. Questa tecnica si fonda sull'osservazione empirica del comportamento del protocollo durante la fase iniziale di *handshake*. Secondo quanto riportato dallo studio, nelle configurazioni (non offuscate), il terzo pacchetto scambiato tra client e server è solitamente un pacchetto ACK applicativo (P_ACK_V1). Questo pacchetto, teoricamente di dimensione fissa, viene usato per confermare la ricezione dei precedenti messaggi di negoziazione (Client Reset e Server Reset).

L'euristica sfruttata dal filtro consiste nell'osservare che:

- I pacchetti ACK hanno una dimensione uniforme;
- Si concentrano nei primi *bin* temporali del flusso
- Tendono a scomparire nella fase di trasmissione dati, dove i pacchetti diventano più variabili e più lunghi.

Nel caso di tunnel crittografati, secondo gli autori, la struttura del pacchetto cambia, ma l'euristica resterebbe applicabile, pur non specificando con precisione in che punto del flusso possa trovarsi l'ACK. In questi casi, infatti, il pacchetto viene incapsulato, e l'ACK potrebbe non essere più il terzo, ma piuttosto apparire in una posizione variabile all'interno del flusso.

8.1.1 Validazione

Per verificare l'efficacia del metodo, è stata condotta una fase di validazione su file .pcap contenenti traffico OpenVPN vanilla su TCP e UDP, oltre a varianti con protezione TLS-crypt.

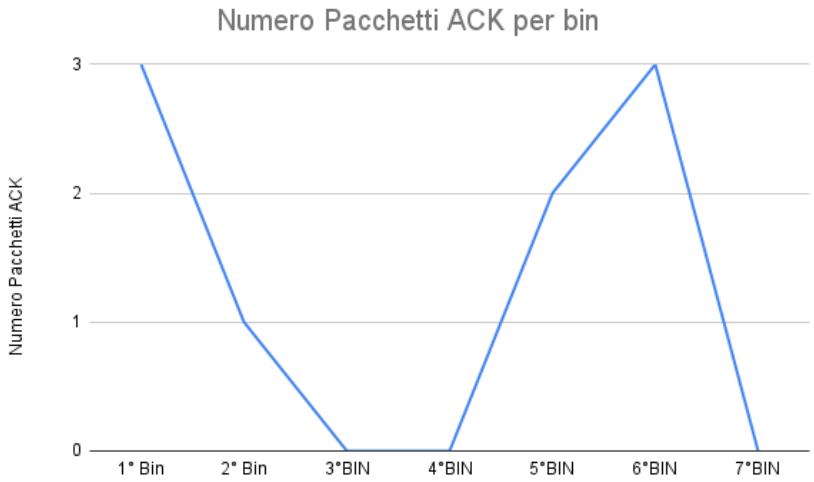

Figura 8.1: Numero di pacchetti ACK per *bin* - OpenVPN Vanilla UDP

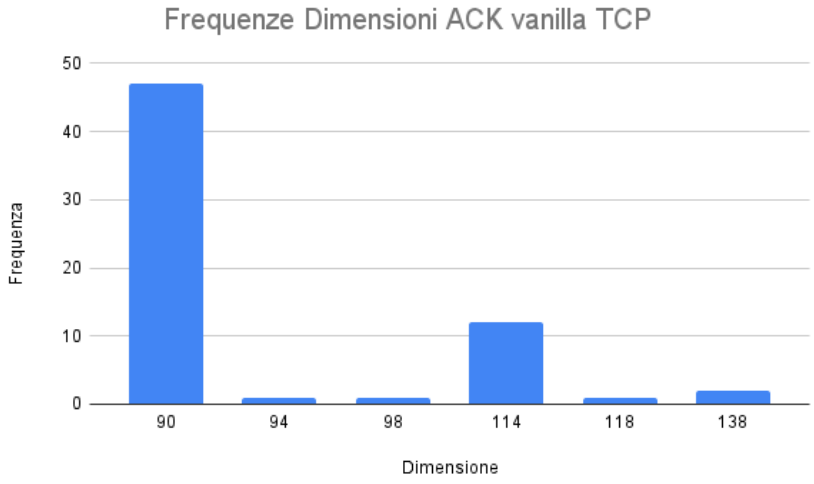

Figura 8.2: Grafico frequenze delle dimensioni dei pacchetti ACK - OpenVPN Vanilla TCP

L'analisi ha evidenziato che, contrariamente a quanto sostenuto dallo studio:

- I pacchetti di tipo ACK non presentano una dimensione unica e costante (Figura 8.2)
- La loro presenza non è limitata alla fase iniziale della connessione, ma può sparire e riapparire anche durante lo scambio di chiavi o in fasi precedenti al trasferimento dati (Figura 8.1)
- In alcune sessioni, i pacchetti ACK sono presenti con dimensioni diverse, o appaiono in sequenza alternata a pacchetti di controllo, rendendo difficile stabilire un *pattern* riconoscibile affidabile.

Anche nel caso di traffico OpenVPN protetto tramite TLS-crypt, si sono osservati dimensioni dei pacchetti variabili tra 110 e 130 byte, incoerenti con l'euristica descritta.

Lo stesso problema è ipotizzabile nei flussi OpenVPN incapsulati in tunnel crittografati (es. Stunnel), per i quali lo studio sostiene che il primo pacchetto ACK possa non essere il terzo pacchetto del flusso, ma possa trovarsi in una posizione diversa (non meglio specificata).

8.2 Implementazione tecnica basata sull'opcode

Partendo dal solo pseudocodice pubblicato nello studio di Xue et. al. [71] (Figura 7.3) ho voluto mettere in pratica la tecnica di identificazione del traffico OpenVPN basata sull'analisi degli opcode spiegata nello studio, al fine di validarne l'efficacia e cercare di migliorarla.

A tal fine, ho sviluppato uno script Python, accessibile tramite GitHub (Appendice A - 9.1), che automatizza l'analisi di file *.pcap*, ispeziona i flussi TCP e UDP e applica l'euristica descritta nello studio di Xue et al. [71]. Lo strumento consente l'estrazione e l'analisi degli opcode presenti nei *payload* dei pacchetti, offrendo un metodo affidabile per identificare il traffico OpenVPN anche in presenza di tecniche di offuscamento leggere, come la *Xor Scramble* (Sez. 6.1.1).

Tuttavia, rispetto allo studio iniziale, si è reso necessario apportare alcune modifiche. Le modifiche apportate riguardano l'introduzione di un filtro pensato per ridurre i falsi positivi dovuti alla presenza di traffico QUIC.

Durante le fasi di validazione è infatti emerso che tale traffico, per via della struttura dei suoi *payload*, veniva in alcuni casi erroneamente classificato come traffico OpenVPN. La soluzione proposta sfrutta una caratteristica intrinseca del protocollo OpenVPN: i primi due pacchetti dell'*handshake* (Client Reset e Server Reset) sono notoriamente di dimensioni ridotte. Di contro, il traffico QUIC tipicamente inizia con pacchetti con dimensione di circa 1300 byte.

Sfruttando questa distinzione, è stato introdotto un controllo che esclude dall'analisi i flussi UDP i cui primi pacchetti superano una soglia di 1000 byte, riducendo sensibilmente il tasso di falsi positivi (FPR).

8.2.1 Script Realizzato

Lo script si compone di tre fasi principali:

1. Parsing e raggruppamento dei pacchetti (analizza_flussi): grazie alla libreria Scapy, i pacchetti vengono estratti dal file .pcap e classificati in flussi distinti (TCP e UDP) sulla base delle tuple IP/porta. Il codice esegue un'analisi preliminare dei pacchetti di rete, organizzandoli in flussi bidirezionali distinti per protocollo (TCP e UDP), memorizzando solo quelli contenenti *payload* e distinguendo, nel caso del TCP, la direzione del traffico (Client $\rightarrow$ Server e Server $\rightarrow$ Client) per facilitare la successiva ricostruzione dei vari segmenti di ogni pacchetto. I pacchetti vengono memorizzati in due dizionari differenti a seconda che siano UDP o TCP (Figura 8.3).

2. Ricostruzione e analisi dei flussi TCP (parse_tcp_stream): Per ogni flusso TCP viene ricostruito il *payload* completo ordinando i frammenti in base al numero di sequenza. Da questo *stream* binario vengono estratti iterativamente i singoli pacchetti (basandosi sui primi due byte che indicano la lunghezza del *payload*), da cui viene calcolato l'*opcode* applicando una maschera *bitwise* $(byte[0]\&0xF8) >> 3$ che permette di estrarre i 5 bit più significativi del primo byte che rappresentano il valore

```python
def analizza_flussi(packets):
    # Crea due dizionari per memorizzare i flussi TCP e UDP
    flows_tcp = defaultdict(lambda: {'C2S': [], 'S2C': []})  # Flussi TCP direzione client a server (C2S) e server a client
(S2C)
    flows_udp = defaultdict(list)  # Flussi UDP

    # Itera su tutti i pacchetti
    for pkt in packets:
        if pkt.haslayer('IP') and pkt.haslayer('TCP') and pkt['TCP'].payload:
            # Se il pacchetto è TCP e ha un payload
            ip = pkt['IP']  # Estrae il layer IP
            tcp = pkt['TCP']  # Estrae il layer TCP
            # Crea una chiave unica per il flusso (src_ip, src_port, dst_ip, dst_port)
            key = (min(ip.src, ip.dst), min(tcp.sport, tcp.dport), max(ip.src, ip.dst), max(tcp.sport, tcp.dport))
            # Determina la direzione del flusso (C2S o S2C)
            direction = 'C2S' if (ip.src, tcp.sport) <= (ip.dst, tcp.dport) else 'S2C'
            # Aggiunge il pacchetto al flusso corrispondente
            flows_tcp[key][direction].append(('TCP', tcp.seq, bytes(tcp.payload)))
        elif pkt.haslayer('IP') and pkt.haslayer('UDP') and pkt['UDP'].payload:
            # Se il pacchetto è UDP e ha un payload
            ip = pkt['IP']  # Estrae il layer IP
            udp = pkt['UDP']  # Estrae il layer UDP
            # Crea una chiave unica per il flusso (src_ip, src_port, dst_ip, dst_port)
            key = (min(ip.src, ip.dst), min(udp.sport, udp.dport), max(ip.src, ip.dst), max(udp.sport, udp.dport))
            # Aggiunge il payload UDP al flusso
```

Figura 8.3: Funzione analizza_flussi

dell'opcode. La stessa maschera viene applicata anche per i pacchetti UDP (Figura 8.4).

3. Classificazione del flusso: ogni flusso (TCP o UDP) avrà il suo set di opcode, con il quale è possibile classificare il flusso secondo i seguenti criteri (Figura 8.5)

   - Il numero di *opcode* contenuti nel set deve essere compreso tra 4 e 10;
   - Non devono verificarsi ripetizioni sospette degli opcode *Client Reset* (CR) o *Server Reset* (SR) dopo i primi due pacchetti;
   - Per i flussi UDP, se entrambi i primi due pacchetti hanno una dimensione superiore a 1000 byte, il flusso viene escluso (euristica anti-QUIC).

Esecuzione dello script

Eseguendo lo script in accordo con quanto riportato nel file Readme[1], l'algoritmo inizierà a scandagliare ogni flusso presente nel file .pcap raccogliendo gli opcode. Al termine

[1] https://github.com/francescomagri02/OpenVPN_Detection/blob/main/Opcode/README.md

```python
def parse_tcp_stream(fragments, flow_key):
    # Ordina i frammenti del flusso TCP in base al numero di sequenza
    fragments.sort(key=lambda x: x[0])
    # Ricostruisce il flusso completo concatenando tutti i payload
    full_stream = b''.join(payload for _, payload in fragments)

    i = 0  # Indice di posizione nel flusso
    pkt_count = 0  # Contatore per i pacchetti elaborati
    while i + 2 <= len(full_stream) and pkt_count < N:
        # Estrae la lunghezza del pacchetto (2 byte)
        length = int.from_bytes(full_stream[i:i+2], byteorder='big')
        start = i + 2  # Punto di partenza del pacchetto
        end = start + length  # Punto finale del pacchetto

        if end > len(full_stream):  # Se il pacchetto è incompleto, interrompi
            break

        # Estrae il payload del pacchetto
        pkt_payload = full_stream[start:end]
        if len(pkt_payload) >= 1:  # Se il pacchetto ha almeno un byte
            # Estrae l'opcode dal primo byte del payload
            opcode = (pkt_payload[0] & 0xF8) >> 3
            print(f"[DEBUG] Flusso {flow_key} - TCP Opcode estratto: {opcode}")
            yield opcode  # Restituisce l'opcode per l'analisi

        i = end  # Aggiorna l'indice per il prossimo pacchetto
        pkt_count += 1  # Incrementa il contatore dei pacchetti
```

Figura 8.4: Funzione parse_tcp_stream

```python
# Verifica se i primi due pacchetti UDP sono troppo grandi (probabile QUIC)
    udp_payload_sizes = [len(payload) for payload in udp_payloads[:2]]
    if any(size > MAX_UDP_PAYLOAD for size in udp_payload_sizes):
        print(f"[INFO] Flusso UDP {key} escluso: primi due pacchetti UDP troppo grandi (probabile QUIC)")
        continue

    # Analizza i pacchetti UDP fino a N
    for payload in udp_payloads[:N]:
        if payload and pkt_count < N:
            # Estrae l'opcode dal primo byte del payload UDP
            opcode = (payload[0] & 0xF8) >> 3
            if pkt_count == 0:
                cr = opcode  # Primo opcode come CR
            elif pkt_count == 1:
                sr = opcode  # Secondo opcode come SR
            else:
                # Se l'opcode è uguale a CR o SR e ci sono più di 4 opcodes, non è OpenVPN
                if (opcode == cr or opcode == sr) and len(opcode_set) >= 4:
                    print(f"[INFO] Flusso UDP {key} classificato come NON OpenVPN (opcode ripetuto CR/SR)")
                    non_openvpn = True
                    break
            opcode_set.add(opcode)
            pkt_count += 1

    is_openvpn = False
    if not non_openvpn and 4 <= len(opcode_set) <= 10:
        is_openvpn = True
```

Figura 8.5: Criteri di classificazione del flusso

dell'esecuzione stamperà in output il riepilogo di tutti i flussi analizzati con l'elenco dei possibili flussi OpenVPN (Figura 8.6).

Si ricorda che, in accordo con il *framework* presentato nello studio di Xue et. al., i flussi contrassegnati come OpenVPN in questa fase, rappresentano solo un sospetto, da confermare in una fase successiva tramite *Active Probing*.

8.2.2 Validazione della tecnica

Per validare l'efficacia dell'algoritmo di individuazione del traffico OpenVPN basato sugli *opcode*, è stata condotta un'analisi su un *dataset* costituito da 10 file pcap contenenti flussi di traffico di vario tipo e realizzato appositamente per questa validazione. Nello specifico:

Nome	Tipologia di traffico
ovpn-auth.pcapng	Traffico OpenVPN con configurazione TLS-Auth catturato da un server OpenVPN attivo in locale
stunnel-capture.pcap	Traffico OpenVPN offuscato con Stunnel catturato da un server OpenVPN attivo in locale
email.pacp	Traffico non-VPN SMTP/POP3/IMAP proveniente dal dataset pubblico ISCXVPN2016
ftp.pcap	Traffico non-VPN FTP proveniente dalla Repository GitHub di nDPI
obfuscated.pcapng	Traffico OpenVPN offuscato con Xor Scramble proveniente dalla Repository GitHub di nDPI
quic.pcap	Traffico non-VPN QUIC proveniente dalla Repository GitHub di nDPI
tcp443.pcap	Traffico OpenVPN TCP su porta 443 ProtonVPN con configurazione TLS-Crypt catturato durante la navigazione
vanilla_TCP.pcapng	Traffico OpenVPN Vanilla catturato da un server OpenVPN attivo in locale
vpnbookfr2.pcap	Traffico OpenVPN TCP catturato durante la navigazione
mixed.pcap	Traffico Misto non-VPN e OpenVPN catturato durante la navigazione

Tabella 8.1: Panoramica dei file pcap utilizzati per la validazione

Per poter effettuare una validazione completa alcuni PCAP includono solo traffico generato da specifiche configurazioni di OpenVPN (vanilla, con tls-auth, offuscato tramite XOR scramble, TCP e UDP su porta standard e non standard, con offuscamento tramite Stunnel), mentre altri, al fine di verificare l'efficacia di questa tecnica su traffico reale, contengono traffico misto non-VPN di diversi protocolli (FTP, SMTP/POP3/IMAP, HTTPS, QUIC) anche proveniente dal *dataset* pubblico ISCXVPN2016.

Il *dataset* realizzato e preso in esame, include tutte le configurazioni di OpenVPN che questa tecnica sarebbe in grado di rilevare, cioè tutte quelle configurazioni che non

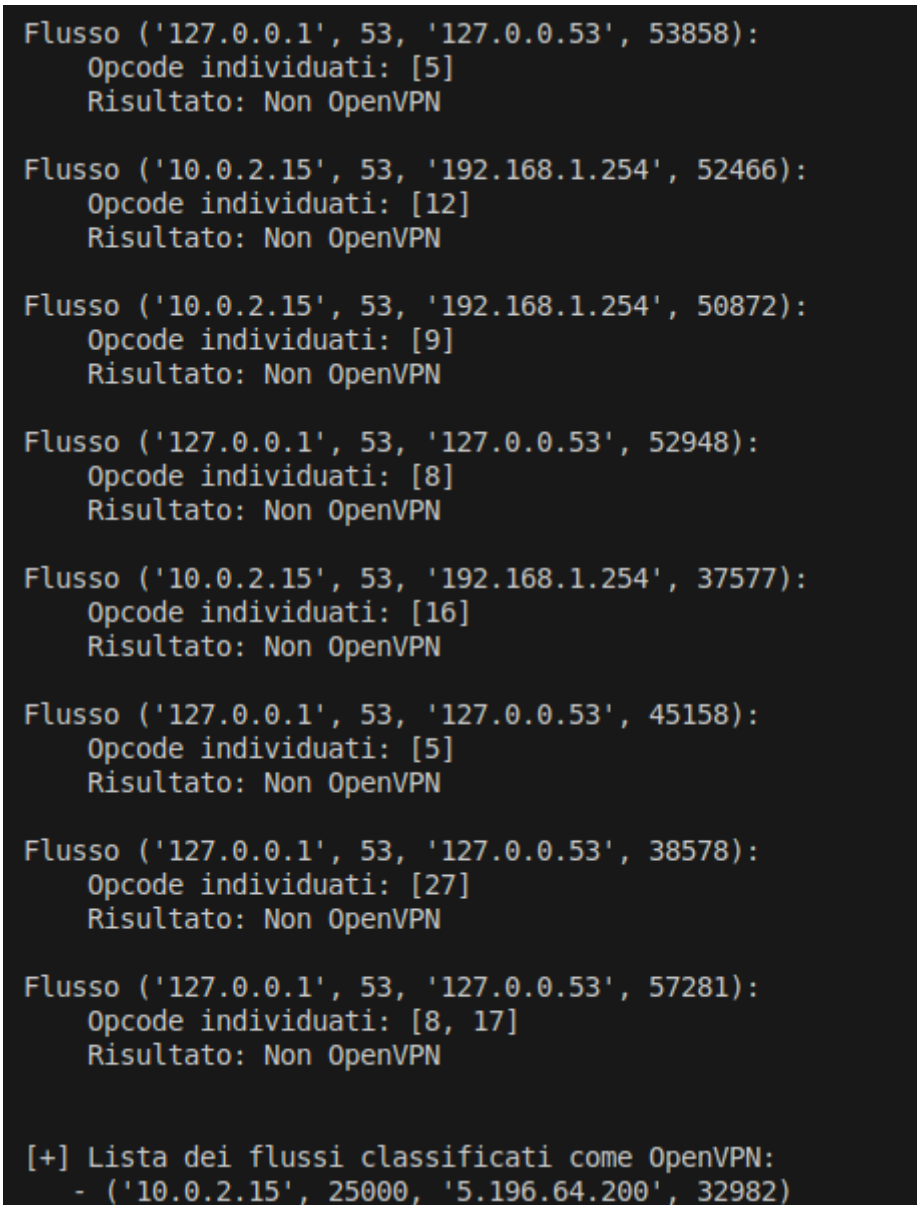

Figura 8.6: Output analisi file .pcap

incapsulano i pacchetti OpenVPN nè variano l'ordine del primo byte del *payload* (es. *Xor Scramble*). Per quanto riguarda il traffico derivante da altri protocolli, di certo il *dataset* preso in esame non risulta essere esaustivo. Ovvero non è composto dal traffico generato da tutti i possibili protocolli della rete, ma solo da una piccola parte che rappresenta più del 70% del traffico web a livello mondiale (es. HTTPS, HTTP, FTP, QUIC, SMTP/POP3/IMAP) [61].

Lo scopo della validazione è valutare la validità della tecnica descritta, e valutare quanto le migliorie apportate al codice abbiano reso lo strumento più performante, attraverso la valutazione dei seguenti punti:

- La capacità del sistema di rilevare correttamente i flussi OpenVPN in tutte le sue varianti (*Recall*);
- la capacità di evitare l'identificazione errata di traffico legittimo come VPN (Precisione);
- L'accuratezza generale;
- L'impatto della soglia di 1000 byte nei pacchetti UDP iniziali per l'eliminazione dei flussi QUIC

Dall'esecuzione dell'algoritmo sul *dataset* predisposto, è emerso che i flussi OpenVPN sono stati correttamente individuati in quasi tutte le varianti, inclusa quella offuscata tramite *XOR Scramble*, ad eccezione di quelli offuscati tramite Stunnel. L'unico dato problematico, che ha reso necessario l'aggiunta di un ulteriore controllo è emerso sul traffico QUIC: in presenza di un numero elevato di pacchetti QUIC, l'FPR era notevolmente alto circa 7 flussi su 10 venivano erroneamente classificati come OpenVPN.

Tuttavia, come mostrato dalla tabella 8.2 e dalle *confusion matrix* 8.7 e 8.8 , con l'aggiunta del filtro l'algoritmo è divenuto più sensibile (*recall* aumentata), più efficace nel complesso (*F1-score* in netto aumento) e più affidabile su scala globale (leggero aumento dell'accuratezza) dimostrando l'efficacia dell'approccio e il miglioramento rispetto l'algoritmo originale.

Algoritmo Originale				Algoritmo Migliorato			
Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score	Accuracy
1	0,5263	0,6896	0,993	1	0,7142	0,8333	0,997

Tabella 8.2: Confronto delle metriche di validazione prima e dopo l'aggiunta del filtro

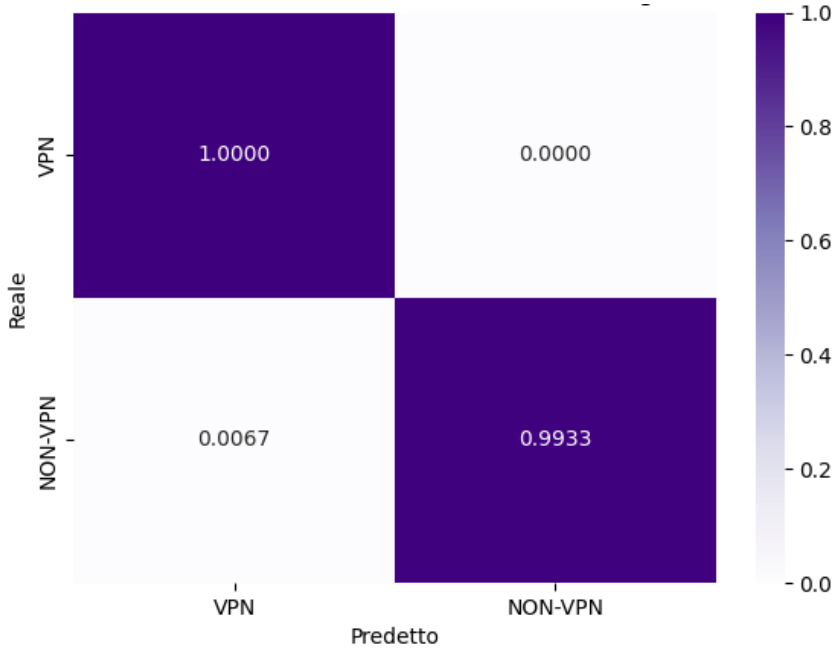

Figura 8.7: *Confusion Matrix* algoritmo originale (senza filtro anti-QUIC)

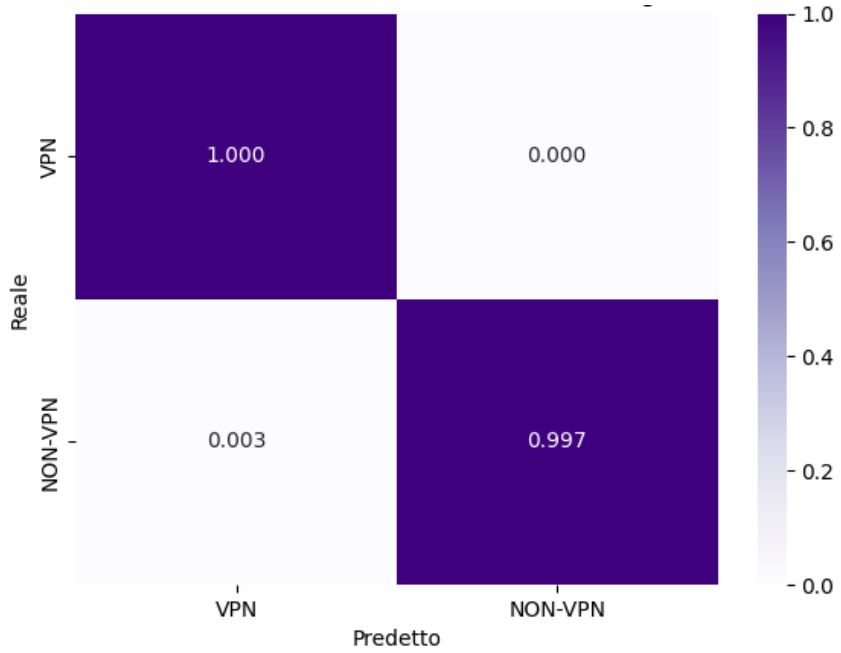

Figura 8.8: *Confusion Matrix* algoritmo migliorato (con filtro anti QUIC)

8.2.3 Confronto dei risultati con lo studio originale

Sebbene lo studio di Xue et al. non fornisca dei dati chiari sulle performance di questa tecnica di identificazione, possiamo stimare questi dati derivandoli dai risultati ottenuti dall'esecuzione di questa tecnica sui flussi generati dalle principali VPN commerciali.

Il confronto con i risultati dello studio di Xue et al.[71] evidenzia che l'algoritmo sviluppato in questo lavoro ha ottenuto prestazioni migliori in termini di *recall* su tutte le configurazioni OpenVPN vanilla e offuscate testate, rispetto a quello originale. In particolare, l'identificazione dei flussi TCP, UDP e XOR ha raggiunto il 100% di *recall*, mentre nello studio originale il valore di *recall* per la medesima tipologia di flussi risulta essere del 98,8%. Per quanto riguarda invece configurazioni OpenVPN offuscate tramite Stunnel, è stato possibile confermare l'inefficacia di questa tecnica.

Tuttavia, è importante sottolineare che il lavoro di Xue et al. è stato condotto su una scala ben più ampia, sia in termini di volumi di traffico analizzato (124.67 terabyte), sia in termini di numero e varietà di *provider VPN* commerciali coinvolti, condizioni difficilmente replicabili in un ambiente di validazione controllato come quello utilizzato in

questo studio.

In sintesi, pur con i limiti di scala e di scenario, i risultati ottenuti indicano che il metodo implementato non solo replica l'efficacia dello studio originale, ma ne estende la robustezza in termini di protocolli ambigui (es. QUIC), rappresentando un contributo concreto all'affinamento di questo tipo di filtro passivo.

8.3 Implementazione tecnica basata sull'Active Probing

Partendo dalla tabella riassuntiva dei *probes* pubblicata nello studio di Xue et al. [71] (Figura 8.9), ho voluto mettere in pratica la tecnica di *Active Probing* presentata. Non essendo riportato nello studio alcun dettaglio implementativo da cui partire, l'obiettivo è stato riprodurre la tecnica in questione cercando di ottenere almeno le stesse performance dello studio originale.

A tal fine, ho sviluppato uno script Python, accessibile tramite GitHub (Appendice A - 9.1), che esegue in automatico il *probing* attivo di uno o più server target, registra eventuali risposte e tempi di chiusura ed elabora un *pattern* comportamentale di chiusura della connessione che permetterebbe di individuare il servizio in esecuzione sul server target.

Lo studio propone un set di pacchetti di *probing* costruiti ad hoc, ciascuno progettato per testare i tempi di chiusura della connessione di un server OpenVPN TCP rispetto a input anomali, malformati o simulanti altri protocolli (es. HTTP, SSH, TLS, ecc.) .

8.3.1 Script Realizzato

Lo script realizzato si compone di quattro funzioni principali:

- probe_single_target (ip, port, payload): Apre un *socket* TCP verso il server con indirizzo IP e porta specificati, impostando un *timeout* per evitare attese indefinite. Stabilita la connessione invia il *payload* al server e misura il tempo trascorso tra l'invio del pacchetto e la ricezione di un'eventuale risposta o chiusura del *socket*. Al termine dell'operazione, restituisce il tempo di chiusura della connessione in secondi, o 0 in caso di errore (Figura 8.10);

ProbeName	Probe Content	Expected Behavior
BaseProbe 1	x00x0ex38.{8}x00x00x00x00x00	Explicit ServerReset or Short Close
BaseProbe 2	x00x0ex38.{8}x00x00x00x00	Long Close
TCP Generic	x0dx0ax0dx0a	Short Close
One Zero	x00	Long Close
Two Zero	x00x00	Short Close
Epmd	x00x01x6e	Short Close
SSH	SSH-2.0-OpenSSH_8.1/r/n	Short Close
HTTP-GET	GET/HTTP/1.0 /r /n /r /n	Short Close
TLS	Typical Client Hello by Chromium	Short Close
2K-Random	Random 2000 Bytes	Short Close & RST

Figura 8.9: Tabella riassuntiva dei probes e del corrispettivo comportamento atteso da un server OpenVPN [71]

```python
def probe_single_target(ip, port, payload):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # Create TCP socket
        s.settimeout(TIMEOUT)                                  # Set timeout
        s.connect((ip, port))                                  # Connects to the destination host
        s.sendall(payload)                                     # Send the payload

        start = time.time()                                    # Start the timer
        try:
            data = s.recv(4096)                                # Waiting for a response (up to 4096 bytes)
            end = time.time()                                  # Record time at reception
            duration = round(end - start, 3)                   # Calculate rounded duration
            if data:
                if args.print_all:                                          # If there is a response
                    print("   Reply " + str(data))
            else:
                if args.print_all:                                              # Socket closed without data
                    print("   No Reply (socket closed)")
        except socket.timeout:
            if args.print_all:                                 # Timeout expired
                print("   Timeout (No Reply)")
            duration = round(time.time() - start, 3)           # Record time anyway
        except socket.error as e:
            if args.print_all:                                # Reception error
                print(f"   Reception Error: {e}")
            duration = round(time.time() - start, 3)
        finally:
            s.close()                                          # Close the socket anyway
        return duration                                        # Return response time
    except Exception as e:
        if args.print_all:                                     # Any other errors (e.g. connection failed)
            print(f"   Connection Failed: {e}")
        return 0.0                                             # No response, returns 0
```

Figura 8.10: Funzione probe_single_target

- send_multiple_probes (ip, port): Questa funzione (Figura 8.11) si occupa di inviare parallelamente ciascun *probe* al server target. Ogni *probe* viene inviato richiamando la funzione *probe_single_target*. L'esecuzione dei *probe* avviene in *multithreading*, sfruttando un *ThreadPoolExecutor* per massimizzare l'efficienza e ridurre i tempi di attesa complessivi. Ogni tempo ricevuto viene classificato tramite la funzione *classify_duration*, che associa a ciascuna misura uno stato (Es. Short Close, Long Close, Error). Nello specifico:
  - Short Close: se il tempo di chiusura è < 2 *secondi*
  - Long Close: se il tempo di chiusura è $\geq$ *Timeout*
  - Error: se il tempo di chiusura è = 0 *secondi*

  I risultati vengono poi stampati in forma tabellare tramite la funzione *print_summary_table* (Figura 8.13);

- probes_from_csv(): permette di estrapolare una lista di indirizzi IP e porte di server da sondare in formato csv e, al termine del processo di *probing*, realizza un file csv contenente per ogni server i tempi di chiusura della connessione di ciascun probe inviato e il potenziale servizio rilevato;

- match_result_to_patterns[2] (result_sequence, patterns): Confronta la sequenza di stati risultante dall'esecuzione delle sonde con una serie di *pattern* attesi, al fine di identificare il comportamento che meglio corrisponde a quello osservato. Dopo aver normalizzato entrambe le sequenze, la funzione calcola il numero di *mismatches* tra ciascun *pattern* e la sequenza osservata. Il *pattern* con il minor numero di *mismatches* viene selezionato come migliore corrispondenza. Se il *match* è perfetto, il nome del *pattern* viene restituito immediatamente; altrimenti, viene indicato anche il grado di tolleranza. Se nessun *pattern* corrisponde, viene restituita l'etichetta *"Unknown"* (Figura 8.12).

[2]Contenuto in matcher.py

```python
def send_multiple_probes(ip, port):
    print(f"\n Testing {ip}:{port}...")
    results = []

    def wrapped_probe(name, payload):
        if args.print_all:
            print(f"-> Sending {name}...")
        t = probe_single_target(ip, port, payload)
        status = classify_duration(t)
        if args.print_all:
            print(f"    Response Time: {t}s -> {status}")
        return {
            "probe_name": name,
            "duration": t,
            "status": status
        }
    # To run each probe in a different thread
    with ThreadPoolExecutor(max_workers=len(probes)) as executor:
        futures = [executor.submit(wrapped_probe, name, payload) for name, payload in probes]
        for future in as_completed(futures):
            try:
                result = future.result()
                results.append(result)
            except Exception as e:
                if args.print_all:
                    print(f"   Probe Error: {e}")

    # Sort the results in the same original sequence as the probes
    results.sort(key=lambda r: [name for name, _ in probes].index(r["probe_name"]))

    print_summary_table(results)

    # Matching results with patterns
    matched_labels = match_result_to_patterns(results, patterns)

    print(f"    Match: {matched_labels}")

    return results
```

Figura 8.11: Funzione send_multiple_probes

```python
def match_result_to_patterns(result_sequence, patterns):
    result_statuses = [res["status"] for res in result_sequence]
    norm_result = normalize_sequence(result_statuses)

    best_match = None
    min_mismatches = len(norm_result) + 1  # initialize with the maximum possible

    for pattern in patterns:
        norm_pattern = normalize_sequence(pattern["expected_sequence"])

        # Skip if sequences are not the same length
        if len(norm_pattern) != len(norm_result):
            continue

        mismatches = sum(1 for a, b in zip(norm_pattern, norm_result) if a != b)

        if mismatches < min_mismatches:
            min_mismatches = mismatches
            best_match = pattern["name"]

    if best_match is not None:
        if min_mismatches == 0:
            return best_match
        else:
            return f"{best_match} (tolerance: {min_mismatches} mismatch)"

    return "Unknown"
```

Figura 8.12: Funzione match_result_to_patterns

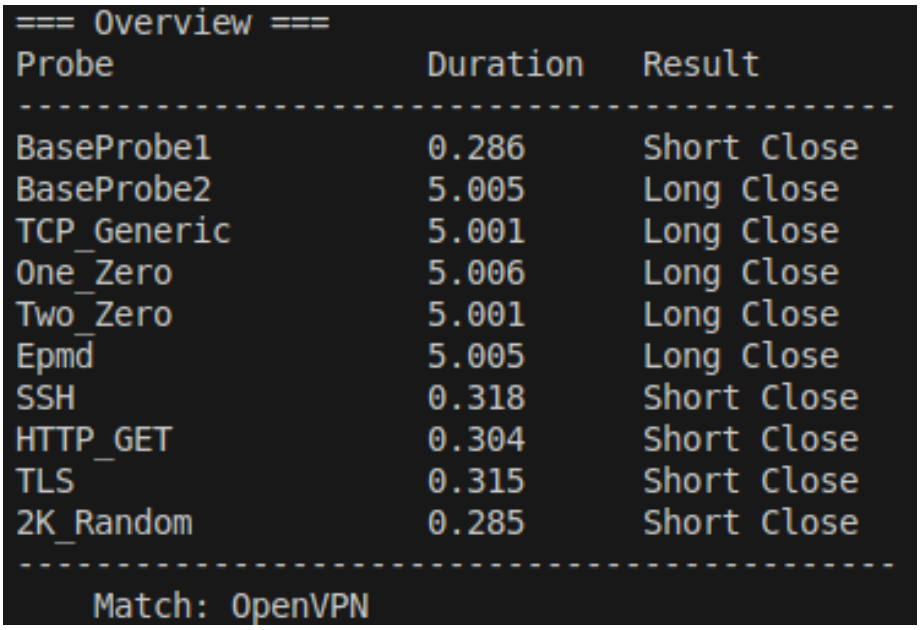

Figura 8.13: Output del probing di un server OpenVPN

Lo script in questione implementa i 10 probe pubblicati nello studio di Xue et. al. ed è in grado di rilevare non solo server OpenVPN, ma ho esteso la rilevazione anche a server FTP, HTTP, HTTPS, PPTP e SMTP con la possibilità di arricchire ulteriormente il ventaglio dei servizi rilevabili, aggiungendo altri *patterns* in formato JSON nella cartella *patterns*.

Esecuzione dello script

Come spiegato nel file Readme[3], le modalità di esecuzione dello script in questione possono essere molteplici a seconda delle esigenze. La Figura 8.13 riporta il risultato dell'esecuzione dello script verso un server target. Nel caso di *probing* tramite file .csv, l'output sarà replicato per ogni server sondato e verrà generato un file .csv di output contenente per ogni server, i tempi di chiusura della connessione per ogni probe ed il potenziale servizio rilevato.

[3]https://github.com/francescomagri02/OpenVPN_Detection/blob/main/Probing/README.md

8.3.2 Validazione della tecnica

La validazione della tecnica in questione è stata effettuata su un *dataset* di 21 server OpenVPN Vanilla presi dalla lista pubblica messa a disposizione dal progetto *VPNGate*, per finalità accademiche e di ricerca.

La validazione ha fatto emergere risultati discordati rispetto allo studio originale. Infatti, la tabella 8.3, mostra come il comportamento rilevato di un server OpenVPN in risposta all'invio dei *probe* in questione differisca rispetto il comportamento atteso pubblicato nello studio.

Inoltre, dall'esecuzione dello script è emerso un punto importante che nello studio originale non è stato trattato, ovvero: nel caso in cui il *probe* inviato non dovesse raggiungere il server a causa di configurazioni particolari o a causa di un firewall che blocca un particolare *probe*, quest'ultimo comprometterebbe l'intera rilevazione.

Per questo motivo, si è reso necessario implementare un meccanismo di confronto tra *pattern* atteso e *pattern* rilevato che fosse più flessibile in caso uno o più *probe* non dovessero ricevere riscontro. Il meccanismo in questione, non solo ritorna il servizio il cui *pattern* atteso corrisponde alla perfezione con quello rilevato, ma se non dovesse trovarne uno, ritorna comunque il servizio con il *pattern* atteso più simile a quello rilevato.

L'applicazione del *pattern* comportamentale rilevato combinata alla miglioria basata sulla somiglianza dei *pattern*, ha portato all'identificazione del 100% dei server OpenVPN presi in esame, dimostrando quindi l'efficacia del *tool* e dell'algoritmo alla base, di identificare server OpenVPN Vanilla.

Tuttavia, per mancanza di un *dataset* di indirizzi IP di server OpenVPN che utilizzino configurazioni con *XOR Scramble*, Stunnel, TLS-Crypt o TLS-Auth non è stato possibile verificare l'efficacia dello script su server con configurazioni non Vanilla.

Inoltre, al fine di verificare il tasso di Falsi Positivi, il *dataset* utilizzato è stato arricchito con 7 server FTP, 5 server HTTP, 8 server HTTPS, 3 server PPTP e 7 server SMTP, ricavati dal motore di ricerca Shodan. L'esecuzione dello script su questa porzione di *dataset* ha fatto emergere non solo un FPR nullo, ma ha anche permesso di rilevare con i medesimi *probes*, dei *pattern* comportamentali peculiari per ogni servizio analizzato,

permettendomi di espandere la collezione dei servizi individuabili dal tool.

Nome del Probe	Contenuto del Probe	Comportamento atteso [71]	Comportamento rilevato
BaseProbe1	x00x0ex38.8x00x00x00x00x00	Short Close	Short Close
BaseProbe2	x00x0ex38.8x00x00x00x00	Long Close	Long Close
TCP Generic	x0dx0ax0dx0a	Short Close	Long Close
One Zero	x00	Long Close	Long Close
Two Zero	x00x00	Short Close	Long Close
Epmd	x00x01x6e	Short Close	Long Close
SSH	SSH-2.0-OpenSSH_8.1/r/n	Short Close	Short Close
HTTP-GET	GET/HTTP/1.0 /r /n /r /n	Short Close	Short Close
TLS	Typical Client Hello by Chromium	Short Close	Short Close
2K-Random	Random 2000 Bytes	Short Close	Short Close

Tabella 8.3: Confronto tra il comportamento atteso di un server OpenVPN in risposta all'invio dei *probes* in questione con il reale comportamento rilevato durante la fase di validazione

8.3.3 Confronto dei risultati con lo studio originale

Considerando i risultati emersi dalla validazione della tecnica di *Active Probing* descritta nello studio di Xue et. al. [71] per il rilevamento di server OpenVPN, sebbene sia emerso un *pattern* comportamentale diverso rispetto a quello pubblicato nello studio, quest'ultimo, unitamente alle modifiche atte a rendere più flessibile la rilevazione, è risultato estremamente efficace essendo stato in grado di rilevare il 100% dei server OpenVPN presenti nel *dataset* preso in esame, mentre, nello studio di Xue et al. [71] è stata raggiunta una percentuale dell'89%.

Sebbene non sia possibile sapere con certezza a cosa sia dovuta questa differenza, viene naturale pensare che tale discrepanza possa essere attribuita a diversi fattori. In primo luogo, è plausibile che il *pattern* comportamentale rilevato da Xue et al. fosse meno preciso rispetto al *pattern* rilevato dal *dataset* da me utilizzato, influenzando dunque l'efficacia del metodo. In secondo luogo, un ruolo determinante potrebbe essere giocato dalla maggiore flessibilità introdotta nell'algoritmo da me sviluppato, il quale, a differenza di quello originale, non richiede una corrispondenza esatta con i *pattern* attesi, ma è

in grado di tollerare alcuni *mismatches*, individuando comunque la sequenza più simile. Questo approccio più "tollerante" permette di rilevare varianti comportamentali non canoniche ma riconducibili allo stesso tipo di server, aumentando così sensibilmente il tasso di rilevamento.

Inoltre, questo metodo, si è rivelato molto efficace nel riconoscimento di altri servizi, riuscendo a rilevare il 100% dei server FTP, HTTP, HTTPS, PPTP e SMTP presenti nel *dataset*, dimostrando di poter andare ben oltre lo scopo per il quale è stato progettato.

8.4 Considerazioni finali

8.4.1 Identificazione basata sull'ACK

La tecnica *ACK-based* descritta da Xue et al. [71], pur essendo interessante a livello teorico, non si è rivelata efficace nè affidabile nella pratica. Le condizioni necessarie al suo funzionamento (dimensione fissa dei pacchetti ACK, comparsa concentrata all'inizio della sessione, scomparsa nella fase di trasferimento dati) non si verificano in modo costante, nè nel traffico vanilla nè in quello offuscato o protetto.

Inoltre, la mancanza di indicazioni chiare per scenari complessi (come OpenVPN incapsulato in tunnel crittografati) rende la tecnica difficile da generalizzare. Questa instabilità ne limita fortemente l'utilità, tanto che strumenti di identificazione del traffico accreditati come nDPI non la implementano, preferendo metodi più robusti basati su caratteristiche strutturali più affidabili, come gli opcode.

8.4.2 Identificazione basata sull'opcode

Sebbene il *dataset* utilizzato per la validazione includa flussi di tutte le varianti di OpenVPN rilevabili dall'euristica e traffico non-VPN, è composto da soli 10 file .pcap per un totale di 1358 flussi.

Nonostante i 10 file .pcap utilizzati per la validazione includano tutte le varianti di OpenVPN individuabili attraverso la tecnica proposta, va sottolineato che la loro quantità limitata dipende da fattori pratici. In particolare, non è possibile escludere che alcuni pro-

vider VPN commerciali adottino configurazioni proprietarie o tecniche di offuscamento avanzate, potenzialmente capaci di eludere i meccanismi di rilevamento implementati.

L'inclusione nel *dataset* di traffico proveniente da una vasta gamma di servizi VPN commerciali sarebbe stata quindi auspicabile per testare la robustezza della tecnica su scenari reali e diversificati. Tuttavia, una simile operazione avrebbe richiesto risorse economiche e logistiche non compatibili con il contesto di questo lavoro, in quanto avrebbe comportato la sottoscrizione di numerosi abbonamenti e la configurazione di molteplici ambienti di test.

Analogamente, l'acquisizione di grandi volumi di traffico non-VPN (ad esempio proveniente da ISP o ambienti enterprise) avrebbe richiesto l'attivazione di collaborazioni con operatori del settore e l'accesso a infrastrutture di monitoraggio ad alto traffico. Anche qualora tale collaborazione fosse stata realizzabile, l'analisi dei dati raccolti avrebbe richiesto risorse computazionali elevate e strumenti di elaborazione su larga scala, al di fuori delle possibilità disponibili per il presente studio.

Possibili test e sviluppi futuri

Risulta quindi fondamentale espandere il *dataset* di validazione includendo un volume significativamente maggiore di traffico, catturato da diverse reti (es. ISP o data center). Questo permetterebbe di rilevare altri possibili protocolli ambigui (come visto per il traffico QUIC) che potrebbero portare ad ulteriori miglioramenti dell'algoritmo, riducendo ulteriormente il tasso di falsi positivi.

Inoltre, sarebbe opportuno includere campioni di traffico provenienti da un'ampia gamma di provider VPN commerciali. Questo permetterebbe di rilevare eventuali configurazioni offuscate di server OpenVPN che sfuggono alla rilevazione, in modo da analizzare le tecniche di offuscamento utilizzate ed affinare ulteriormente l'algoritmo di rilevamento.

8.4.3 Identificazione basata sull'Active Probing

Nonostante il *dataset* utilizzato per la validazione includa 21 indirizzi IP associati a server OpenVPN, esso è composto esclusivamente da server con configurazioni "vanilla", ovvero prive di personalizzazioni avanzate.

La quantità limitata di server testati è dovuta al fatto che l'*Active Probing* è, per sua natura, un'attività potenzialmente invasiva: implica l'invio deliberato di pacchetti verso infrastrutture di terze parti con l'obiettivo di analizzarne il comportamento. Per questo motivo, è stato possibile includere nel *dataset* solo server OpenVPN messi a disposizione per fini di ricerca. Un'eventuale estensione del *dataset* (ad esempio includendo server appartenenti a provider VPN commerciali) avrebbe richiesto la stipula di accordi formali con tali aziende, che avrebbero dovuto fornire accesso a server dedicati per consentire test controllati, nel pieno rispetto delle normative e dell'etica della ricerca.

Possibili test e sviluppi futuri

Un naturale passo successivo consisterebbe nel testare l'algoritmo su un numero significativamente maggiore di server OpenVPN, includendo configurazioni eterogenee come l'uso di TLS-Crypt, TLS-Auth, Stunnel, o altri layer di offuscamento. Questo richiederebbe la collaborazione con *provider* VPN commerciali o l'allestimento di ambienti di test più ampi e diversificati, al fine di valutare la robustezza del metodo in scenari reali e meno prevedibili.

Un altro ambito di sviluppo riguarda l'estensione del *dataset* a un maggior numero di server non-VPN, già parzialmente testati (es. FTP, HTTP, HTTPS, PPTP, SMTP). L'obiettivo è identificare *pattern* comportamentali univoci per altri servizi, così da estendere l'algoritmo non solo al riconoscimento di OpenVPN, ma anche ad altri servizi.

A tal fine, sarà fondamentale ampliare il numero di *probes*, aggiungendo altri *payload* in grado di stimolare comportamenti che possano permettere di individuare anche altri servizi o altre configurazioni di OpenVPN.

9. CONCLUSIONI

Il presente lavoro di tesi ha esplorato in maniera sistematica il dinamico campo delle tecniche di riconoscimento e offuscamento del traffico di rete, con un focus specifico sui protocolli OpenVPN e Shadowsocks. In un contesto come quello attuale caratterizzato da crescente sorveglianza, profilazioni, e restrizioni sull'accesso all'informazione, l'impiego di tecnologie per la privacy e l'anonimato come le VPN e i proxy è divenuto cruciale [55].

Tuttavia l'evoluzione di tali strumenti ha innescato parallelamente lo sviluppo di metodologie sempre più sofisticate volte alla loro identificazione e al blocco da parte di attori quali *provider* di servizi internet, amministratori di rete e regimi censori come il Great Firewall (GFW) cinese [21].

L'analisi condotta ha evidenziato come l'identificazione del traffico si basi su un insieme eterogeneo di tecniche, che spaziano dall'ispezione di pacchetti e flussi (DPI/DFI) all'estrazione di pattern statistici e comportamentali. Metodi passivi, come l'analisi degli opcode o l'applicazione di test di entropia, possono rivelarsi efficaci per identificare configurazioni standard o offuscamenti semplici (Xor Scramble). Tuttavia, come emerso sia dalla letteratura che dai test sperimentali qui condotti, la loro efficacia diminuisce significativamente in presenza di tecniche di offuscamento più avanzate, quali l'incapsulamento del traffico OpenVPN in tunnel crittografati (es. Stunnel) o l'adozione di *pluggable transport* per Shadowsocks che alterano in modo significativo la firma del traffico (es. Obfs4, V2Ray Plugin) e la loro applicazione in contesti in cui si necessita l'analisi di una grande quantità di dati in tempo reale è una delle sfide più grandi.

L'active probing si è affermato come un approccio cruciale, capace di sondare attivamente un endpoint per elicitarne reazioni e dedurne la natura, anche in assenza di risposte esplicite. L'analisi del comportamento di flussi TCP, in particolare le modalità e le tempistiche di chiusura della connessione (FIN/RST threshold), si è dimostrata promettente nell'identificare configurazioni dichiaratamente "probe-resistant" come Shadowsocks e Obfs4, sebbene l'adozione di V2Ray Plugin possa teoricamente, se opportunamente configurata, aggirare il sistema di rilevamento.

Il *Machine Learning* rappresenta una frontiera importante per la classificazione del traffico. Algoritmi basati su *feature* ignengerizzate o approcci di *Deep Learning* che apprendono automaticamente *pattern* complessi hanno mostrato elevate accuratezze su specifici *dataset*. Tuttavia, la necessità di vasti *dataset* etichettati, l'onere computazionale per l'analisi su larga scala e in tempo reale e la difficoltà di adattarsi rapidamente a protocolli o configurazioni nuove rappresentano sfide significative che limitano la loro applicabilità pratica immediata.

La validazione delle tecniche di identificazione del traffico OpenVPN ha rappresentato una fase cruciale di questo studio, permettendo di valutare l'applicabilità e l'efficacia di metodologie esistenti, evidenziandone limiti e opportunità di miglioramento. Dalla validazione è infatti emerso che, il *framework* preso in esame, così come era stato pensato dagli autori, presentava delle notevoli criticità.

Per quanto riguarda l'analisi degli ACK, la validazione ha evidenziato una discrepanza significativa rispetto all'euristica inizialmente descritta dalla letteratura di riferimento. Infatti, la mancanza di costanza e uniformità di pacchetti ACK in un flusso OpenVPN, non permette in alcun modo di ricavare un *pattern* di rilevamento affidabile.

Riguardo l'analisi basata sugli Opcode, invece, è stato apportato un contributo significativo attraverso il perfezionamento dell'algoritmo esistente, diminuendo il tasso di falsi positivi riguardanti flussi di tipo QUIC.

Infine, la validazione della tecnica di Active Probing ha rivelato risultati discordanti rispetto allo studio originale. Infatti, il *tool* realizzato per identificare i server OpenVPN ha permesso di ricavare un *pattern* comportamentale di server OpenVPN differente da quello fornito nello studio, il quale si è rivelato particolarmente efficace. Inoltre, l'algoritmo alla base del *tool* è stato reso più flessibile e completo, in quanto è in grado di rilevare non solo OpenVPN, ma anche servizi di altro tipo.

9.1 Sviluppi futuri

Guardando al futuro, le principali sfide per l'efficacia delle tecnologie di offuscamento e, di conseguenza, per i sistemi di rilevazione, risiedono nella necessità di una costante

adattabilità e sofisticazione.

Dal lato dell'offuscamento:

- Una delle sfide primarie consiste nel sviluppare protocolli e tecniche in grado di mimetizzare in modo convincente il traffico come comunicazione legittima e comune (es. HTTPS, TLS, QUIC, ecc.) Questo non richiede solo l'alterazione superficiale dei pacchetti, ma la simulazione fedele dei *pattern* comportamentali (*handshake*, gestione degli errori, struttura dei flussi) per eludere non solo la DPI ma anche le analisi basate su pattern comportamentali dei flussi.

- Migliorare la resistenza all'*active probing* è fondamentale. Le tecniche future dovranno andare oltre la semplice mancata risposta, cercando di simulare comportamenti di server legittimi o introdurre variabilità nelle risposte e nelle tempistiche di chiusura per rendere inefficaci le analisi basate su soglie precise.

- Mantenere un equilibrio tra *stealthiness*, prestazioni e usabilità rimarrà critico. Tecniche di offuscamento molto complesse possono introdurre latenza e ridurre la velocità di trasmissione, rendendole meno attrattive [55]. Le soluzioni future dovranno essere efficienti dal punto di vista computazionale e facili da configurare e distribuire.

Dal lato della rilevazione, invece:

- Superare le limitazione del *Machine Learning* per renderlo efficace e facilmente scalabile in ambienti di rete reali e ad alta velocità è una sfida aperta. Ciò include affrontare la gestione di *dataset* sbilanciati [73], l'identificazione di traffico "sconosciuto" non visto in fase di addestramento, e lo sviluppo di modelli che si adattino dinamicamente a nuove forme di offuscamento.

- Sviluppare tecniche di analisi comportamentale più robuste, capaci di identificare traffico sospetto indipendentemente dal protocollo specifico utilizzato, diventerà sempre più importante. Specialmente per contrastare quelle tecniche di offuscamento che mirano a simulare comportamenti di protocolli legittimi.

In conclusione, il futuro della privacy e della sicurezza online in contesti censori dipenderà dalla continua innovazione nelle tecniche di offuscamento e dalla parallela capacità dei sistemi di rilevazione di adattarsi a strategie sempre più sofisticate. La ricerca in questo campo rimane quindi di fondamentale importanza.

Appendice A

L'intero progetto è disponibile su GitHub al seguente indirizzo:

`https://github.com/francescomagri02/OpenVPN_Detection`

Il codice sorgente dello script Python inerente l'individuazione del traffico OpenVPN basata sull'analisi degli opcode è disponibile al seguente link

`https://github.com/francescomagri02/OpenVPN_Detection/tree/main/Opcode`

Il codice sorgente dello script Python inerente l'individuazione dei server OpenVPN basata sull'Active Probing è disponibile al seguente link

`https://github.com/francescomagri02/OpenVPN_Detection/tree/main/Probing`

Bibliografia

[1] Alice, Bob, Carol, Jan Beznazwy, and Amir Houmansadr. How china detects and blocks shadowsocks. In *Proceedings of the ACM Internet Measurement Conference*, pages 111–124, 2020.

[2] Ahmed Almusawi and Haleh Amintoosi. Dns tunneling detection method based on multilabel support vector machine. *Security and Communication Networks*, 2018(1):6137098, 2018.

[3] Nour Alqudah and Qussai Yaseen. Machine learning for traffic analysis: a review. *Procedia Computer Science*, 170:911–916, 2020.

[4] Yawning Angel. obfs4 protocol specification. `https://github.com/Yawning/obfs4/blob/master/doc/obfs4-spec.txt`.

[5] Tom Auld, Andrew W Moore, and Stephen F Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on neural networks*, 18(1):223–239, 2007.

[6] Ahmad Azab, Mahmoud Khasawneh, Saed Alrabaee, Kim-Kwang Raymond Choo, and Maysa Sarsour. Network traffic classification: Techniques, datasets, and challenges. *Digital Communications and Networks*, 10(3):676–692, 2024.

[7] Ahmad Azab, Mahmoud Khasawneh, Saed Alrabaee, Kim-Kwang Raymond Choo, and Maysa Sarsour. Network traffic classification: Techniques, datasets, and challenges. *Digital Communications and Networks*, 10(3):676–692, 2024.

[8] Sikha Bagui, Xingang Fang, Ezhil Kalaimannan, Subhash C Bagui, and Joseph Sheehan. Comparison of machine-learning algorithms for classification of vpn network traffic flow using time-related features. *Journal of Cyber Security Technology*, 1(2):108–126, 2017.

[9] Robert Beverly. A robust classifier for passive tcp/ip fingerprinting. In *International workshop on passive and active network measurement*, pages 158–167. Springer, 2004.

[10] Mitko Bogdanoski, Tomislav Shuminoski, and Aleksandar Risteski. Analysis of the syn flood dos attack. *International Journal of Computer Network and Information Security*, 5(8):1, 2013.

[11] Mitko Bogdanoski, Tomislav Shuminoski, and Aleksandar Risteski. Analysis of the syn flood dos attack. *International Journal of Computer Network and Information Security*, 5(8):1, 2013.

[12] Sergey Bratus, Cory Cornelius, David Kotz, and Daniel Peebles. Active behavioral fingerprinting of wireless devices. In *Proceedings of the first ACM conference on Wireless network security*, pages 56–61, 2008.

[13] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Hyojoon Kim, Renata Teixeira, and Nick Feamster. Traffic refinery: Cost-aware data representation for machine learning on network traffic. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(3):1–24, 2021.

[14] Nevil Brownlee, Cyndi Mills, and Greg Ruth. *Rfc2722: Traffic flow measurement: Architecture*. RFC Editor, 1999.

[15] Jiaxing Cheng, Ying Li, Cheng Huang, Ailing Yu, and Tao Zhang. Acer: detecting shadowsocks server based on active probe technology. *Journal of Computer Virology and Hacking Techniques*, 16:217–227, 2020.

[16] Jin Cheng, Yulei Wu, Junling You, Tong Li, Hui Li, Jingguo Ge, et al. Matec: A lightweight neural network for online encrypted traffic classification. *Computer Networks*, 199:108472, 2021.

[17] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5–16, 2007.

[18] Ziye Deng, Zihan Liu, Zhouguo Chen, and Yubin Guo. The random forest based detection of shadowsock's traffic. In *2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, volume 2, pages 75–78. IEEE, 2017.

[19] Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. ndpi: Open-source high-speed deep packet inspection. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 617–622. IEEE, 2014.

[20] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pages 407–414, 2016.

[21] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. Analyzing china's blocking of unpublished tor bridges. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*, 2018.

[22] W Eddy. *Rfc 9293: Transmission control protocol (tcp)*. RFC Editor, 2022.

[23] Sergey Frolov, Jack Wampler, and Eric Wustrow. Detecting probe-resistant proxies. In *NDSS*, 2020.

[24] Sergey Frolov, Jack Wampler, and Eric Wustrow. Detecting probe-resistant proxies. In *NDSS*, 2020.

[25] Peipei Fu, Chang Liu, Qingya Yang, Zhenzhen Li, Gaopeng Gou, Gang Xiong, and Zhen Li. Nsa-net: A netflow sequence attention network for virtual private network traffic detection. In *Web Information Systems Engineering–WISE 2020: 21st International Conference, Amsterdam, The Netherlands, October 20–24, 2020, Proceedings, Part I 21*, pages 430–444. Springer, 2020.

[26] Ping Gao, Guangsong Li, Yanan Shi, and Yang Wang. Vpn traffic classification based on payload length sequence. In *2020 International Conference on Networking and Network Applications (NaNA)*, pages 241–247. IEEE, 2020.

[27] Ibrahim Ghafir and Vaclav Prenosil. Blacklist-based malicious ip traffic detection. In *2015 Global Conference on Communication Technologies (GCCT)*, pages 229–233. IEEE, 2015.

[28] Deepali Gobare, Ankita Patil, Sahil Pawar, and Ravi Tarate. Network traffic analysis using random forest algorithm. *International Research Journal of Modernization in Engineering Technology and Science*, 2024.

[29] Honglin He. A network traffic classification method using support vector machine with feature weighted-degree. *Journal of Digital Information Management*, 12(2), 2017.

[30] Yanjie He and Wei Li. A lightweight deep learning approach for encrypted proxy traffic detection. *Security and Communication Networks*, 2025(1):4469975, 2025.

[31] OpenVPN Inc. ”tls-crypt-v2.txt”. `https://github.com/OpenVPN/openvpn/blob/v2.6.13/doc/tls-crypt-v2.txt`, 2022.

[32] Mohamad Jaber, Roberto G Cascella, and Chadi Barakat. Can we trust the inter-packet time for traffic classification? In *2011 IEEE international conference on communications (ICC)*, pages 1–5. IEEE, 2011.

[33] Hrithik Jha, Iishi Patel, Gang Li, Aswani Kumar Cherukuri, and Sumaiya Thaseen. Detection of tunneling in dns over https. In *2021 7th International Conference on Signal Processing and Communication (ICSC)*, pages 42–47. IEEE, 2021.

[34] Qingbing Ji, Zhihong Rao, Man Chen, and Jie Luo. Security analysis of shadowsocks (r) protocol. *Security and Communication Networks*, 2022(1):4862571, 2022.

[35] Stephen Kent. IP Encapsulating Security Payload (ESP). RFC 4303, December 2005.

[36] Shashank Khanvilkar and Ashfaq Khokhar. Virtual private networks: an overview with performance evaluation. *IEEE Communications Magazine*, 42(10):146–154, 2004.

[37] Martin Laštovička, Martin Husák, Petr Velan, Tomáš Jirsík, and Pavel Čeleda. Passive operating system fingerprinting revisited: Evaluation and current challenges. *Computer Networks*, 229:109782, 2023.

[38] Guo Lei, Wang Yadi, Yao Qing, Zhu Ke, and Yi Peng. Flow characteristic selection algorithm based on dynamic information in deep flow inspection. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, volume 2, pages 1216–1219. IEEE, 2011.

[39] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3):1999–2012, 2020.

[40] Zhen Lu, Zhenhua Li, Jian Yang, Tianyin Xu, Ennan Zhai, Yao Liu, and Christo Wilson. Accessing google scholar under extreme internet censorship: A legal avenue. In *Proceedings of the 18th ACM/IFIP/USENIX middleware conference: industrial track*, pages 8–14, 2017.

[41] Ivan Nardi Luca Deri. A deep dive into traffic fingerprints using wireshark. In *SharkFest'24 Europe*, 2024.

[42] Ang Kun Joo Michael, Emma Valla, Natinael Solomon Neggatu, and Andrew W Moore. Network traffic classification via neural networks. Technical report, University of Cambridge, Computer Laboratory, 2017.

[43] Shane Miller, Kevin Curran, and Tom Lunney. Multilayer perceptron neural network for detection of encrypted vpn network traffic. In *2018 International conference on cyber situational awareness, data analytics and assessment (Cyber SA)*, pages 1–8. IEEE, 2018.

[44] Milton L Mueller and Hadi Asghari. Deep packet inspection and bandwidth management: Battles over bittorrent in canada and the united states. *Telecommunications Policy*, 36(6):462–475, 2012.

[45] Alhamza Munther, Rozmie Razif, Shahrul Nizam, Naseer Sabri, and Mohammed Anbar. Active build-model random forest method for network traffic classification. *International Journal of Engineering & Technology*, 6:0975–4024, 2014.

[46] Antonio Nappa, Zhaoyan Xu, M Zubair Rafique, Juan Caballero, and Guofei Gu. Cyberprobe: Towards internet-scale active detection of malicious servers. In *In Proceedings of the 2014 Network and Distributed System Security Symposium (NDSS 2014)*, pages 1–15. The Internet Society, 2014.

[47] Openvpn.net. Reference manual for openvpn 2.4. `https://openvpn.net/community-resources/reference-manual-for-openvpn-2-4/`.

[48] Jaroslav Pešek, Dominik Soukup, and Tomáš Čejka. Active learning framework to automate networktraffic classification. *arXiv preprint arXiv:2211.08399*, 2022.

[49] Oskars Podzins and Andrejs Romanovs. Why siem is irreplaceable in a secure it environment? In *2019 open conference of electrical, electronic and information sciences (eStream)*, pages 1–5. IEEE, 2019.

[50] Ashis Pradhan. Network traffic classification using support vector machine and artificial neural network. *International Journal of Computer Applications*, 8:8–12, 2011.

[51] The Tor Project. Tor: Overview. `https://2019.www.torproject.org/about/overview.html.en`.

[52] Eric Rescorla. The transport layer security (tls) protocol version 1.3. Technical report, 2018.

[53] Shadowsocks. V2ray-plugin. `https://github.com/shadowsocks/v2ray-plugin/`.

[54] Tal Shapira and Yuval Shavitt. Flowpic: A generic representation for encrypted traffic classification and applications identification. *IEEE Transactions on Network and Service Management*, 18(2):1218–1232, 2021.

[55] Mohammad Shehab and Lial Raja Alzabin. Evaluating the effectiveness of stealth protocols and proxying in hiding vpn usage. *Journal of Computational and Cognitive Engineering*, 2022.

[56] Meng Shen, Yiting Liu, Siqi Chen, Liehuang Zhu, and Yuchao Zhang. Webpage fingerprinting using only packet length information. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[57] Fabio Streun, Joel Wanner, and Adrian Perrig. Evaluating susceptibility of vpn implementations to dos attacks using adversarial testing. In *Network and Distributed Systems Security Symposium 2022 (NDSS'22)*. Internet Society, 2022.

[58] Stunnel. Stunnel. `https://www.stunnel.org/index.html`.

[59] Dennis Tatang, Florian Quinkert, Nico Dolecki, and Thorsten Holz. A study of newly observed hostnames and dns tunneling in the wild. *arXiv preprint arXiv:1902.08454*, 2019.

[60] Icegate VPN Team. Building a resilient vpn service. 2024.

[61] Keysight Technologies. Http/2 and http/3 dominating internet traffic in 2025, 2025.

[62] Florian Tschorsch. Openvpn tls-crypt-v2 key wrapping with hardware security modules.

[63] Tunnelblick. OpenVPN-xorpatch. `https://tunnelblick.net/cOpenvpn_xorpatch.html`.

[64] Silvio Valenti, Dario Rossi, Alberto Dainotti, Antonio Pescapè, Alessandro Finamore, and Marco Mellia. Reviewing traffic classification. *Data Traffic Monitoring and Analysis: From Measurement, Classification, and Anomaly Detection to Quality of Experience*, pages 123–147, 2013.

[65] Ramachandran Venkateswaran. Virtual private networks. *IEEE potentials*, 20(1):11–15, 2001.

[66] Hung Nguyen Viet, Quan Nguyen Van, Linh Le Thi Trang, and Shone Nathan. Using deep learning model for network scanning detection. In *Proceedings of the 4th international conference on frontiers of educational technologies*, pages 117–121, 2018.

[67] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE international conference on intelligence and security informatics (ISI)*, pages 43–48. IEEE, 2017.

[68] Stephen Watkins, George Mays, Ronald M Bandes, Brandon Franklin, Michael Gregg, and Chris Ries. *Hack the stack: Using snort and ethereal to master the 8 layers of an insecure network*. Elsevier, 2006.

[69] Li Wei, Liu Hongyu, and Zhang Xiaoliang. A network data security analysis method based on dpi technology. In *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 973–976. IEEE, 2016.

[70] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, and Eric Wustrow. How the great firewall of china detects and blocks fully encrypted traffic. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2653–2670, 2023.

[71] Diwen Xue, Reethika Ramesh, Arham Jain, Michaelis Kallitsis, J Alex Halderman, Jedidiah R Crandall, and Roya Ensafi. Openvpn is open to vpn fingerprinting. *Communications of the ACM*, 68(1):79–87, 2025.

[72] Ryo Yamada and Shigeki Goto. Using abnormal ttl values to detect malicious ip packets. *Proceedings of the Asia-Pacific Advanced Network*, 34(0):27, 2013.

[73] Jingan Yang. The application of deep learning for network traffic classification. *Highlights Sci. Eng. Technol*, 39:979–984, 2023.

[74] Peng Yang, Juan Shao, Wen Luo, Lisong Xu, Jitender Deogun, and Ying Lu. Tcp congestion avoidance algorithm identification. *IEEE/ACM Transactions On Networking*, 22(4):1311–1324, 2013.

[75] Ruixi Yuan, Zhu Li, Xiaohong Guan, and Li Xu. An svm-based machine learning method for accurate internet traffic classification. *Information Systems Frontiers*, 12:149–156, 2010.

[76] Xuemei Zeng, Xingshu Chen, Guolin Shao, Tao He, Zhenhui Han, Yi Wen, and Qixu Wang. Flow context and host behavior based shadowsocks's traffic identification. *IEEE Access*, 7:41017–41032, 2019.