
UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

# UNIVERSITÀ DEGLI STUDI DI MODENA

# E REGGIO EMILIA

Dipartimento di Ingegneria "Enzo Ferrari"

Corso di Laurea Magistrale in Ingegneria Informatica (LM-32)

## Adversarial Attack contro Network Intrusion Detection System basati su Machine Learning in Traffic Space

**Relatore:**
Prof. Mirco Marchetti
**Correlatore:**
Ing. Dimitri Galli

**Candidato:**
Giuseppe Falco
Matricola 202612

---

**ANNO ACCADEMICO 2025/2026**

# *Abstract*

**Adversarial Attack contro Network Intrusion Detection System basati su Machine Learning in Traffic Space**

I *Network Intrusion Detection System* basati su *machine learning* (ML-NIDS) raggiungono prestazioni elevate nel riconoscimento del traffico malevolo, ma restano esposti agli *adversarial attack*, il cui obiettivo è evadere il detector: alterare il traffico malevolo in modo che venga classificato come benigno e sfugga al rilevamento. Gran parte della letteratura valuta questa vulnerabilità in *feature space*, perturbando le variabili di flusso già estratte, oppure assume capacità dell'attaccante poco realistiche, come il controllo dell'*exporter* dei flussi o l'interrogazione ripetuta del classificatore. Poiché il sensore di rilevamento risiede su un'infrastruttura protetta, tali ipotesi rischiano di sovrastimare l'efficacia reale dell'evasione.

Questa tesi valuta l'evasione in *traffic space*, introducendo perturbazioni a livello di pacchetto su *packet capture*, prima dell'aggregazione in flussi. Si considerano due strategie, valutate separatamente: il *padding* del payload dei pacchetti e il *ritardo* del loro invio. Il modello di minaccia assume un attaccante *blind*, che modifica soltanto il proprio traffico malevolo nel rispetto dei vincoli di protocollo. La valutazione sperimentale riguarda più famiglie di *botnet* e *malware* tratte dai dataset pubblici CTU-13 e MCFP, traffico TCP e UDP, e tre classificatori: *Decision Tree*, *Random Forest* e *HistGradientBoosting*. A fini di confronto con la letteratura, le stesse perturbazioni sono replicate anche in *feature space*.

**Parole chiave:** ML-NIDS, Adversarial Machine Learning, Evasion, Traffic Space, Packet Capture, Botnet

*Ricorda a te stesso il perché*
*e non dimenticherai mai per chi lo stai facendo.*
*Non dimenticare mai chi sei,*
*il resto del mondo non lo farà.*

*A me stesso, alle mie paure,*
*alla mia determinazione,*
*ai miei sogni.*

*Alla mia famiglia,*
*senza la quale non sarei la persona che sono oggi,*
*va tutto il mio cuore.*

*A Lisa, la mia metà,*
*al futuro che costruiremo insieme,*
*va tutta la mia vita.*

*A tutti coloro che mi sono stati vicini,*
*a voi va la mia gratitudine*
*per aver reso questo viaggio più leggero.*

*A tutti voi,*
*dedico ogni mio traguardo.*

# Indice

# Elenco delle Figure

# Elenco dei Codici

# 1. Introduzione

## 1.1 Contesto e motivazione

I *Network Intrusion Detection System* (NIDS) monitorano il traffico di rete per individuare attività malevole o anomale e segnalano agli operatori di sicurezza gli eventi sospetti [13]. Di fronte all'aumento dei volumi di traffico e alla difficoltà di mantenere aggiornate le regole dei sistemi basati su firme, una parte crescente della ricerca ha adottato tecniche di *machine learning* (ML), dando origine ai ML-NIDS, che analizzano il traffico aggregato in flussi e raggiungono prestazioni elevate sui *benchmark* di riferimento [3].

Affidare la decisione a un modello di *machine learning* introduce però un'ulteriore superficie di attacco. L'*adversarial machine learning* studia come un soggetto malintenzionato possa indurre errori in un classificatore manipolando i dati che gli vengono sottoposti [4]. Nel rilevamento delle intrusioni di rete l'obiettivo tipico è l'*evasion*: alterare il traffico malevolo in modo che il classificatore lo etichetti come benigno, riducendo la quota di attacchi effettivamente intercettati.

## 1.2 Problema

La robustezza dei ML-NIDS di fronte a queste manipolazioni è oggetto di numerose valutazioni, che però adottano spesso assunzioni difficili da giustificare in un contesto operativo. Molti lavori misurano l'evasione in *feature space*, perturbando direttamente le variabili numeriche già estratte dai flussi, oppure attribuiscono all'attaccante capacità poco realistiche, come il controllo dell'*exporter* che produce i record di flusso o la possibilità di interrogare ripetutamente il classificatore [3].

In un'installazione reale, tuttavia, il sensore e la pipeline di estrazione delle feature risiedono su un'infrastruttura protetta, fuori dalla portata dell'attaccante. Un avversario credibile può intervenire soltanto sul traffico generato dagli host che controlla, non sulle feature di flusso né sul modello. Perturbazioni applicate direttamente alle feature, senza verificare che siano realizzabili sul traffico sottostante, rischiano quindi di sovrastimare l'efficacia dell'evasione rispetto a ciò che un attaccante può effettivamente ottenere.

## 1.3 Obiettivo e contributo

Questa tesi propone *adversarial attack* in *traffic space* contro un ML-NIDS. Le perturbazioni sono introdotte a livello di pacchetto su *packet capture* (`.pcap`), prima dell'aggregazione in flussi e dell'estrazione delle feature. A differenza degli attacchi in *feature space*, ogni modifica deve corrispondere a un pacchetto valido e conforme ai protocolli, così che l'effetto misurato sia quello realmente ottenibile da un attaccante che agisce sul proprio traffico.

Si propongono due strategie di perturbazione, valutate separatamente. La prima è il *padding* del payload applicativo, che aggiunge byte ai pacchetti del traffico malevolo aumentando il volume dei dati trasmessi senza alterarne la semantica. La seconda è il *ritardo* dell'invio dei pacchetti, che posticipa gli istanti di invio del traffico malevolo per simulare un ritardo nella trasmissione. Entrambe agiscono unicamente sul traffico malevolo e restano vincolate a ciò che un attaccante può realizzare sul singolo pacchetto.

Il *threat model* adottato descrive un attaccante *blind* che ha una conoscenza di tipo *black box* del sistema target: non conosce l'architettura del NIDS né l'insieme delle feature, non costruisce modelli surrogati e non interroga il classificatore. Le perturbazioni rispettano vincoli di protocollo verificati a monte sul pacchetto, come la presenza di un *handshake* completo, il limite imposto dall'MTU (*Maximum Transmission Unit*) e il ricalcolo dei *checksum*.

Il contributo di questa tesi è la proposta di *adversarial attack* realistici in *traffic space* contro un ML-NIDS e la loro valutazione. Le perturbazioni sono realizzate e verificate sul singolo pacchetto, in modo conforme ai protocolli, con le feature ricalcolate dopo aggregazione in flussi e preprocessing con gli stessi strumenti e le stesse regole usati per il traffico non perturbato, così da misurare l'evasione effettivamente realizzabile sul traffico e non una sua stima ottenuta sulle sole feature. La valutazione si basa su tracce di traffico reale dei dataset pubblici CTU-13 [9] e MCFP [15], che includono sia traffico benigno sia malevolo e coprono famiglie di *botnet* e *malware* del 2011 e del 2017. Sul traffico non perturbato i classificatori raggiungono una *detection rate* elevata, in linea con le prestazioni attese da un detector basato su *machine learning* e compatibili con un impiego operativo.

Tra le due strategie, il *padding* del payload è la più efficace, in particolare sul traffico UDP, mentre la perturbazione sulla durata incide poco o nulla sul rilevamento. Quest'ultima produce su

TCP una robustezza solo apparente: la *detection rate* resta stabile non perché il modello resista, ma perché la perturbazione, nella maggior parte dei casi, non sposta in misura apprezzabile le feature su cui il classificatore decide.

Il confronto con la replica in *feature space* costituisce il risultato principale. La valutazione condotta direttamente sulle feature, frequente in letteratura e spesso assunta come stima ottimistica dell'evasione, tende a sovrastimarla rispetto a quella realizzabile in *traffic space*. Misurare la robustezza di un ML-NIDS sulle sole feature può quindi discostare la stima dell'evasione da quella effettivamente conseguibile sul traffico.

## 1.4 Struttura della tesi

Il resto dell'elaborato è organizzato come segue. Il Capitolo 2 fornisce il *background* su NIDS, ML-NIDS e *adversarial machine learning*, e si chiude con il *problem statement* che motiva l'analisi in *traffic space*. Il Capitolo 3 formalizza il *threat model* (scenario di rete, target ML-NIDS, capacità dell'attaccante) e definisce le due famiglie di perturbazione con i relativi vincoli di protocollo. Il Capitolo 4 descrive l'implementazione: risorse di calcolo, dataset, preprocessing, classificatori e scenari sperimentali. Il Capitolo 5 presenta le metriche e i risultati negli scenari *clean* e perturbato, insieme al confronto con il *feature space*. Il Capitolo 6 riassume il contributo, ne discute i limiti e indica possibili sviluppi futuri.

# 2. Background e analisi dello stato dell'arte

Per valutare la robustezza di un classificatore basato su *machine learning* di fronte ad attacchi mirati, occorre innanzitutto chiarire come funzionano i sistemi di rilevamento delle intrusioni di rete, quali informazioni estraggono dal traffico e quali limiti ne condizionano l'efficacia.

Il capitolo procede come segue: la §2.1 introduce architetture e rappresentazioni del traffico; la §2.2 descrive pipeline, algoritmi e metriche; la §2.3 commenta prestazioni e criticità strutturali; la §2.4 tratta tassonomia degli attacchi, spazi di perturbazione e difese; la §2.5 riassume le criticità metodologiche delle valutazioni di robustezza e, in §2.5.2, confronta quattro lavori sugli *adversarial attacks* contro ML-NIDS posizionando il contributo di questa tesi.

La §2.6 chiude il capitolo evidenziando i limiti delle assunzioni adottate in letteratura e motivando l'analisi in *traffic space* su *packet capture*, sviluppata nel Capitolo 3.

## 2.1 Network Intrusion Detection Systems

### 2.1.1 Definizione e ruolo

Un *Network Intrusion Detection System* (NIDS) è un componente di sicurezza il cui compito è monitorare il traffico di rete al fine di individuare attività potenzialmente malevole o anomale [13]. Rispetto a un firewall, che applica policy di filtraggio per consentire o negare il transito delle comunicazioni, il NIDS è orientato al rilevamento: monitora il traffico di rete, ne analizza pacchetti e pattern, e segnala eventi sospetti senza bloccare automaticamente le connessioni. L'*Intrusion Prevention System* (IPS), invece, può interrompere attivamente sessioni ritenute malevole; NIDS e IPS condividono spesso moduli di analisi, ma restano distinti per il grado di intervento sul traffico in transito.

Operativamente, il NIDS si compone di sensori posti in punti strategici della rete che raccolgono il traffico osservato e lo trasmettono a un motore di analisi. Quest'ultimo applica regole o modelli di rilevamento e, in caso di anomalia, produce allarmi destinati agli operatori o a un *Security Information and Event Management* (SIEM), sistema per la raccolta e l'analisi centralizzata degli

eventi di sicurezza. In molte installazioni il sensore non è collocato sul percorso principale dei pacchetti: il traffico gli viene fornito in copia, ad esempio tramite *mirroring* di porta (*Switched Port Analyzer*, SPAN) configurato su uno switch di rete, che replica verso una porta dedicata il transito osservato su altre porte. La Figura 2.1 illustra un esempio di posizionamento passivo del NIDS: il traffico dall'Internet transita attraverso router e firewall fino a raggiungere gli host interni, mentre una copia del flusso sullo switch di accesso viene inviata al sensore tramite port mirroring (SPAN); a seguito di un allarme, il NIDS può notificare operatori o SIEM e, in alcune architetture, suggerire contromisure al firewall.

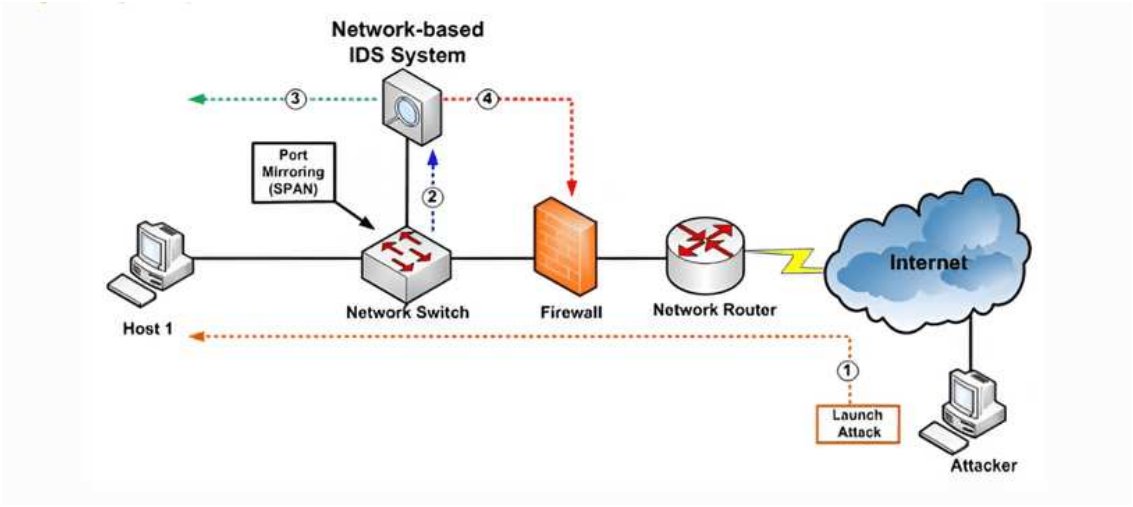



Figura 2.1: Esempio di collocazione passiva di un NIDS: il sensore riceve tramite SPAN una copia del traffico sullo switch di rete e segnala eventi sospetti senza interporre sul percorso originale dei pacchetti.

L'obiettivo resta individuare per tempo attacchi, traffico anomalo o comportamenti incompatibili con la normalità attesa, delegando alle fasi successive la valutazione e l'eventuale risposta.

### 2.1.2 Approcci di rilevamento

I NIDS tradizionali si classificano, in linea di massima, in due famiglie complementari [13].

I sistemi *signature-based* (o basati su regole) confrontano il traffico osservato con un insieme di pattern associati ad attacchi noti: stringhe maliziose, sequenze di eventi già catalogate. Offrono interpretabilità elevata e bassi tassi di falsi positivi quando la firma è accurata, ma richiedono

manutenzione continua delle regole e faticano a generalizzare ad attacchi mai visti (*zero-day*) o a varianti che eludono le firme esistenti.

I sistemi *anomaly-based* modellano il comportamento "normale" della rete, per host, per servizio o per aggregati di flusso, e segnalano scostamenti significativi rispetto a tale modello. Possono teoricamente intercettare minacce non ancora codificate in regole, ma sono sensibili alla definizione di normalità: cambiamenti legittimi del traffico (nuovi servizi, picchi stagionali, manutenzioni) possono generare falsi allarmi, mentre attacchi lenti o mimici del traffico ordinario possono sfuggire al rilevamento.

Nella pratica, molte installazioni adottano architetture ibride che combinano firme aggiornabili, euristiche e componenti statistici o basati su *machine learning*, così da bilanciare copertura, tasso di allarme e costo di gestione [13]. Per la tesi è rilevante osservare che, indipendentemente dal metodo di decisione, il NIDS resta un sistema di monitoraggio spesso collocato in segmenti di rete protetti: l'analisi avviene *dopo* la cattura del traffico e non presuppone che un attaccante remoto possa modificare liberamente i moduli interni del sensore.

### 2.1.3 Rappresentazione del traffico

Il traffico osservato da un NIDS può essere analizzato a livello di singolo pacchetto oppure in forma aggregata [13]. Nel primo caso, ogni *frame* o pacchetto conserva header e, quando disponibile, payload applicativo: massimo dettaglio, maggiore costo di storage e analisi, minore utilità del payload in presenza di cifratura end-to-end.

Per scale di volume tipiche delle reti aziendali e degli operatori, è comune raggruppare pacchetti correlati in *flussi* (*flows*), identificati ad esempio dalla quintupla sorgente/destinazione (indirizzi IP, porte, protocollo) e da intervalli temporali. Protocolli come NetFlow ed esportatori affini producono record con contatori (pacchetti e byte trasmessi), durata della connessione e statistiche derivate dagli header: informazioni sufficienti per molte analisi di sicurezza anche quando il contenuto applicativo non è accessibile [7, 3]. Nel contesto operativo del rilevamento in rete e nella letteratura sui ML-NIDS, l'analisi per flusso è una scelta ricorrente [3], anche nelle valutazioni che ne misurano la robustezza di fronte ad attacchi *adversarial* [2].

La scelta tra analisi per pacchetto e per flusso influenza architettura, performance e tipo di attacco rilevabile. Un IDS orientato ai pacchetti può ispezionare sequenze e payload non cifrati;

un IDS orientato ai flussi sfrutta regolarità statistiche e relazioni tra connessioni. Nella letteratura sull'*adversarial machine learning* applicata agli IDS, molte evasioni sono modellate come perturbazioni di variabili numeriche già estratte dal flusso, ad esempio durata, byte scambiati o contatori di pacchetti, piuttosto che come modifiche al traffico grezzo osservato in rete [1]. Una linea di lavoro alternativa interviene invece sui dati grezzi del problema (*problem space*), alterando il traffico a monte dell'estrazione delle feature [2]. La distinzione tra *feature space* e perturbazioni a monte del sensore è approfondita in §2.5.2.

## 2.2 ML-based Network Intrusion Detection

### 2.2.1 Motivazione

I NIDS basati esclusivamente su regole statiche presentano limiti noti: manutenzione delle firme, scarsa generalizzazione agli attacchi non catalogati e difficoltà a tenere il passo con volumi di traffico crescenti. Ciò ha motivato l'adozione di tecniche di *machine learning* nel rilevamento delle intrusioni di rete [13, 7]. Un modello addestrato su dati etichettati o su rappresentazioni del traffico normale può apprendere regolarità difficili da codificare manualmente e, in molti studi, superare detector puramente basati su firme su benchmark pubblici.

Si definisce *machine learning-based network intrusion detection system* (ML-NIDS) un NIDS che include, tra i propri componenti, almeno un modello di *machine learning* addestrato per produrre inferenze sul traffico osservato [3]. Nel paragrafo precedente si è distinto tra analisi per pacchetto e per flusso. Gli ML-NIDS adottano quasi sempre flussi aggregati o feature estratte da essi: il volume del traffico rende impraticabile un'analisi pacchetto per pacchetto su larga scala.

### 2.2.2 Architettura e pipeline

Un ML-NIDS estende l'architettura di un NIDS classico con una pipeline di elaborazione del traffico e una fase di inferenza basata su *machine learning* [3]. I dati in ingresso possono provenire da PCAP (`.pcap`), da exporter NetFlow o da altre sorgenti già integrate nel NIDS. L'elaborazione può avvenire in batch (analisi offline) o in tempo quasi reale, a seconda dell'architettura adottata. Le fasi tipiche di tale pipeline sono riassunte di seguito.

1. **Acquisizione** del traffico di rete (file '.pcap').
2. **Aggregazione** in flussi o finestre temporali coerenti con la chiave di sessione adottata.
3. **Estrazione di feature** numeriche da header e statistiche di flusso (durata, contatori di pacchetti e byte, flag di protocollo, ecc.).
4. **Preprocessing**: normalizzazione, codifica di attributi categorici, gestione di valori mancanti e, talvolta, *feature selection*.
5. **Addestramento e inferenza** mediante un algoritmo di *machine learning* (es. alberi di decisione, *Random Forest*, reti neurali).
6. **Output**: allarme, etichetta di classe o punteggio di rischio verso operatori o SIEM.

L'implementazione concreta sarà descritta nei capitoli sperimentali.

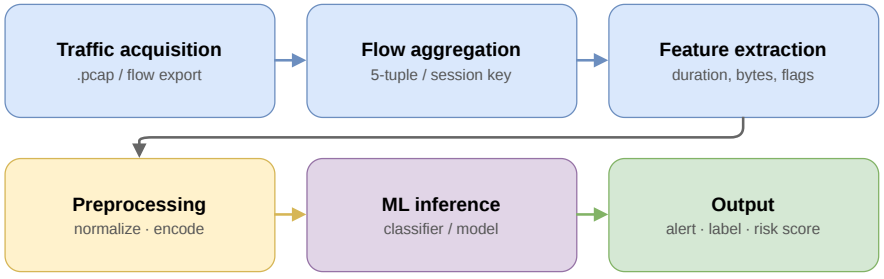



Figura 2.2: Schema logico di una pipeline ML-NIDS: dal traffico osservato in rete alla generazione di un allarme o di un'etichetta.

Il preprocessing trasforma i dati grezzi in un vettore di feature compatibile con l'algoritmo scelto. L'addestramento richiede un insieme di esempi etichettati (supervisionato) oppure, negli approcci non supervisionati, solo esemplari di traffico considerato normale; in entrambi i casi la qualità e la rappresentatività del dataset condizionano le prestazioni in produzione [7].

### 2.2.3 Algoritmi e modalità di apprendimento

Si distinguono, in linea di massima, approcci *supervisionati* e *non supervisionati* [3, 7]. Con l'apprendimento supervisionato il modello è addestrato su flussi etichettati come benigni o malevoli, ma ottenere etichette corrette e rappresentative del traffico operativo resta oneroso e soggetto a errori. Negli approcci non supervisionati o semi-supervisionati, il modello apprende un profilo di traffico considerato legittimo e segnala deviazioni rispetto ad esso. Poiché uno scostamento statistico non implica necessariamente un'intrusione, gli allarmi generati richiedono in genere una revisione da parte degli operatori di sicurezza [13].

Per quanto riguarda gli algoritmi, la letteratura sui ML-NIDS include metodi classici (*Decision Trees*, *Random Forest*, *Support Vector Machines*, *k-NN*) e architetture di deep learning (*Multilayer Perceptron*, CNN, LSTM), senza un consenso univoco sul paradigma migliore: le performance dipendono dal dataset, dalla pipeline di feature e dal contesto operativo [13, 3]. Molte valutazioni ML-NIDS, inclusa quella di questa tesi, si appoggiano a *Random Forest* o classificatori analoghi su feature estratte dai flussi, privilegiando efficacia e leggerezza computazionale rispetto ad architetture di deep learning.

### 2.2.4 Metriche di valutazione

Per valutare l'efficacia di un ML-NIDS si adottano le metriche usuali della classificazione, in genere in formulazione binaria (traffico benigno *vs* malevolo). Definiti veri positivi (TP), veri negativi (TN), falsi positivi (FP) e falsi negativi (FN), le metriche più utilizzate in letteratura sono:

- **Precision** (TP / (TP + FP)): quota di allarmi di minaccia effettivamente corretti rispetto al totale degli allarmi emessi;
- **Recall** (TP / (TP + FN)), talvolta indicata come *detection rate*: quota di attacchi reali individuati rispetto a quelli presenti nel traffico osservato;

- **F1-score**: media armonica di precision e recall, utile a sintetizzare il compromesso tra allarmi affidabili e copertura del rilevamento;
- **Tasso di falsi positivi** (FPR, FP / (FP + TN)): quota di traffico benigno classificato erroneamente come malevolo; incide sul carico operativo legato alla revisione degli allarmi segnalati.

La sola *accuracy* può risultare fuorviante in presenza di classi fortemente sbilanciate, condizione frequente nel traffico di rete. In un contesto di rilevamento, la *recall* esprime invece la quota di attacchi reali intercettati: una sua riduzione indica che parte del traffico malevolo sfugge al classificatore, effetto tipico delle evasioni *adversarial*. Anche un F1 elevato non garantisce per sé l'applicabilità operativa se il tasso di falsi allarmi resta insostenibile per gli operatori [6, 3]. Il confronto sistematico tra prestazioni dei ML-NIDS e dei NIDS classici, nonché i limiti legati ai benchmark di valutazione, sono approfonditi nella sezione successiva.

## 2.3 Sicurezza e limiti dei ML-NIDS

La sezione precedente ha introdotto pipeline, algoritmi e metriche di valutazione. Si passa ora a confrontare le prestazioni dei ML-NIDS con quelle dei NIDS classici e a discutere i limiti che non si evidenziano dai soli benchmark presenti in letteratura.

### 2.3.1 Prestazioni rispetto ai NIDS classici

Su dataset etichettati ampiamente utilizzati in letteratura, molti ML-NIDS raggiungono recall e F1 elevati rispetto a detector basati esclusivamente su firme [6, 3]. Un modello addestrato su un numero sufficiente di esempi può intercettare pattern di attacco non ancora codificati in regole statiche e adattarsi, entro certi limiti, a varianti non previste da un NIDS *signature-based*. Inoltre, l'aggregazione per flusso e l'estrazione automatica di feature possono ridurre la necessità di aggiornare manualmente regole di rilevamento quando il traffico o le minacce cambiano.

Una recall elevata da sola non determina quale approccio sia preferibile in rete. I detector a regole offrono spesso maggiore interpretabilità dell'allarme e, quando le firme sono accurate, tassi di falsi positivi ridotti. Un ML-NIDS può eccellere su benchmark di laboratorio e risultare

meno prevedibile in produzione, dove la distribuzione del traffico diverge da quella dei dataset di addestramento [3, 7]. Non esiste dunque un approccio superiore assoluto: la scelta tra NIDS a firme, architettura ibrida o ML-NIDS va confrontata con il traffico da monitorare, il numero di allarmi che gli operatori possono gestire e il bilanciamento desiderato tra attacchi intercettati e falsi allarmi.

### 2.3.2 Limiti strutturali

Apruzzese et al., nel SoK sulla valutazione del ML per il rilevamento di rete, segnalano un divario tra ricerca e impiego operativo: molti articoli si limitano a mostrare che un nuovo metodo supera lo stato dell'arte su benchmark condivisi, con *accuracy*, F1 o recall leggermente migliori rispetto al paper precedente, e considerano sufficiente quel risultato [3]. In un'installazione reale, invece, conta il bilanciamento tra prestazioni del rilevamento e costi sostenibili negli scenari di routine: revisione degli allarmi, risorse di calcolo, manutenzione del modello. Migliorare di poco una metrica su un dataset di laboratorio non equivale a dimostrare l'utilità di un ML-NIDS nell'uso quotidiano.

Gran parte della letteratura adotta ancora un protocollo sperimentale ereditato da altri domini del *machine learning*: partizione casuale train/test, metriche aggregate e dataset consolidati ma spesso datati o fortemente sbilanciati. Mancano, con frequenza, split che rispettino l'ordine temporale del traffico, attaccanti adattivi o condizioni vicine a quelle di rete; spesso si assume che l'attaccante possa agire sulle feature già estratte anziché sul traffico osservato. In queste condizioni, *accuracy* e F1 possono risultare elevati pur descrivendo in modo incompleto quanto il classificatore performerebbe una volta installato.

Da qui nasce l'esigenza di valutazioni meno ancorate ai benchmark di laboratorio e più attente alle condizioni in cui un ML-NIDS opera davvero, anche sotto attacco. Proprio in questa direzione si muove l'analisi sperimentale di questa tesi, che estende il filo del lavoro di Apruzzese verso uno scenario di minaccia e di perturbazione del traffico più vicino a quanto un attaccante può realisticamente fare in rete; i dettagli sono sviluppati nel *problem statement* e nei capitoli successivi.

Il traffico reale evolve nel tempo (*concept drift*): nuovi servizi, cambiamenti stagionali o aggiornamenti infrastrutturali modificano la distribuzione delle feature osservate e possono degradare le prestazioni di un modello addestrato su dati storici [7, 2]. Accanto al drift temporale, incidono la qualità del dataset di addestramento e l'affidabilità delle etichette: definiscono cosa il modello

apprende come traffico legittimo o minaccioso. Etichette errate o campioni poco rappresentativi del traffico osservato in rete si riflettono sulle inferenze successive, alterando la soglia effettiva tra allarmi corretti e falsi positivi.

Anche quando F1 o recall risultano elevati sui benchmark, un tasso di falsi positivi persistente impone agli operatori di revisionare molti allarmi per intercettare le minacce reali: il rilevamento diventa oneroso da gestire nel tempo. Molti modelli ad alta capacità predittiva restano inoltre difficili da interpretare: a differenza di un allarme prodotto da una regola esplicita, non è sempre immediato ricostruire quali caratteristiche del flusso abbiano determinato la classificazione, con ricadute sulla verifica dell'incidente e sulla risposta.

Oltre a drift, scarsa qualità dei dataset e costo operativo degli allarmi, i ML-NIDS sono esposti all'evasione: a *test time* un attaccante può eludere il classificatore con perturbazioni mirate del traffico, minaccia affrontata nella sezione successiva nel quadro dell'*adversarial machine learning*.

## 2.4 Adversarial ML attacks

L'*adversarial machine learning* studia come un soggetto malintenzionato possa indurre errori in un modello mediante manipolazioni dei dati di ingresso [4]. Nel dominio del rilevamento di rete, l'obiettivo tipico dell'attaccante è l'*evasion*: far classificare come benigno traffico malevolo già intercettabile in assenza di perturbazione, riducendo la recall del ML-NIDS. Si distingue questo scenario da attacchi di *poisoning*, operati in fase di addestramento per alterare il comportamento del modello a partire da campioni corrotti o rimossi dal dataset.

### 2.4.1 Conoscenza del sistema target

La letteratura classifica gli attacchi in base alla conoscenza che l'attaccante possiede del sistema difensivo [4]. In un setting *black box* il modello e la pipeline di feature restano ignoti; l'attaccante osserva solo le proprie azioni e, talvolta, un feedback limitato sugli esiti. In un setting *grey box* sono noti elementi parziali, ad esempio famiglia di algoritmo, insieme di feature o dataset usato in addestramento, come negli esperimenti di evasione su classificatori di botnet basati su flussi e *Random Forest* [1]. In un setting *white box* l'attaccante conosce l'intero NIDS, con architettura, feature e modello noti, e non solo il classificatore isolato; in produzione questa ipotesi resta rara,

perché il sistema difensivo è protetto e chi ne avesse il controllo completo tenderebbe a intervenire in modo più diretto che limitarsi a perturbare il traffico.

È opportuno considerare il bersaglio dell'attacco non come il solo classificatore, ma l'intera catena acquisizione–feature–inferenza: perturbazioni efficaci sulle feature possono risultare inutilizzabili se non realizzabili sul traffico grezzo che le genera [3].

### 2.4.2 Spazi di perturbazione

Molti studi sul ML-NIDS modellano l'evasione in *feature space*: le perturbazioni agiscono sulle variabili numeriche già estratte dal flusso, anziché sul traffico osservato a monte del sensore, e non sempre viene verificato che ogni valore alterato corrisponda ad una perturbazione realistica del traffico. Apruzzese et al. [1] mostrano un caso in cui è presente un detector botnet basato su flussi e *Random Forest* e l'attaccante introduce piccoli ritardi o byte aggiuntivi nel traffico controllato; tali modifiche si traducono in variazioni delle feature estratte dal flusso e possono far classificare come benigno traffico malevolo.

Nel *problem space* le perturbazioni intervengono sul traffico grezzo anziché sulle feature già estratte, come nel setting con perturbazioni *blind* e *concept drift* analizzato in [2]. Il modello di minaccia descrive un attaccante che altera il traffico originato dagli host sotto il proprio controllo, anziché intervenire sull'exporter NetFlow o sulle feature già materializzate dal sensore.

Poiché perturbazioni in *feature space* possono violare vincoli di protocollo, timing o volume e risultare irrealizzabili, stimare l'evasione modificando direttamente le feature senza verificarne la realizzabilità sul traffico può portare a risultati ottimistici. Per avvicinare il modello di minaccia a un attaccante reale, questa tesi opera in *traffic space* su PCAP (`.pcap`).

## 2.5 Stato dell'arte e lavori correlati

La sezione precedente ha introdotto tassonomia, spazi di perturbazione e contromisure nell'ambito dell'*adversarial machine learning*. Questa sezione riassume le criticità metodologiche ricorrenti nelle valutazioni di robustezza e confronta alcuni lavori presenti in letteratura con l'approccio adottato in questa tesi.

### 2.5.1 Evasione e valutazione empirica

Il *machine learning* è stato impiegato per migliorare il rilevamento di minacce su traffico di rete [6, 19]; allo stesso tempo, classificatori addestrati su tracce pubbliche o operative risultano esposti all'evasione quando il traffico malevolo è alterato in modo mirato [13]. Spesso la valutazione riguarda detector su feature di flusso e modelli di minaccia con conoscenza parziale del sistema difensivo [1]. Prestazioni elevate su dataset condivisi non implicano che il modello mantenga recall e precisione una volta applicate perturbazioni realistiche al traffico.

In molti casi la robustezza è misurata su feature già estratte o sotto ipotesi sulle capacità dell'attaccante che in produzione sono difficili da giustificare (ad esempio interrogazioni ripetute del classificatore). Per avvicinare le valutazioni di robustezza all'impiego operativo, Apruzzese et al. indicano criteri più stringenti per i ML-NIDS [3]. Tra questi rientrano minacce capaci di adattarsi al detector, il rispetto dell'ordine temporale del traffico nelle prove e perturbazioni limitate a comportamenti realizzabili sulla rete.

### 2.5.2 Related works

Si confrontano alcuni contributi rappresentativi sugli *adversarial attacks* contro ML-NIDS, concentrandosi su come modellano la minaccia, su dove agiscono le perturbazioni e su quali modifiche restano realizzabili in base a ciò che l'attaccante può controllare. La Tabella 2.1 li organizza distinguendo i lavori che agiscono sul traffico grezzo da quelli in *feature space*, e posiziona il contributo sperimentale di questa tesi. Il *threat model* operativo è sviluppato nel Capitolo 3 (§3.1) e non viene riportato qui.

Come anticipato in §2.4.2, molti studi operano in *feature space* alterando direttamente le variabili di flusso, mentre altri intervengono su catture di pacchetti o su tracce pre-collezionate. Verkerken et al. [18] osservano che la realizzabilità di una perturbazione dipende dalle capacità dell'attaccante: modificare un PCAP già acquisito dal router non equivale, dal punto di vista operativo, a controllare il traffico emesso da un host compromesso. Nel contesto dei ML-NIDS, Han et al. [10] denominano *traffic space* le perturbazioni che agiscono su pacchetti o PCAP (`.pcap`) prima dell'aggregazione in flussi, mentre altri lavori, incluso Apruzzese et al. [2], impiegano per attacchi concettualmente simili l'etichetta *problem space*. Le due denominazioni non segnano spazi distinti, ma terminologie

diverse per la stessa famiglia di perturbazioni a monte del sensore, distinte dalla *feature space* di §2.4.2.

Tabella 2.1: Confronto tra lavori sugli *adversarial attacks* contro ML-NIDS e approccio adottato in questa tesi.

| **Lavoro** | **Spazio** | **Minaccia** | **Trasporto** | **Perturbazioni** | **Valutazione evasione** |
|---|---|---|---|---|---|
| Apruzzese et al. [2] | Problem / PCAP | Black box (blind) | UDP + TCP (PSH) | Padding payload [1–100] B su traffico malevolo | Recall su feature di flusso sotto *concept drift* |
| Galli et al. [8] | Problem / PCAP | Tre scenari ICS | Solo Modbus TCP | Jitter (timestamp) e padding payload | Detection rate su CIC Modbus 2023 |
| Han et al. [10] | Traffic / PCAP | Grey / black box | Generico (header); TCP e UDP | Operatori di mutazione (timing, iniezione) guidati da GAN + PSO | Robustezza su NIDS multipli; extractor e classificatore surrogati |
| Yan et al. [22] | Traffic / PCAP | Black box (*model extraction*) | TCP/IPv4 | Padding, numero pacchetti e timing guidati da surrogato; transfer attack | Robustezza su NIDS multipli |
| Hashemi et al. [11] | Traffic / PCAP (pacchetto) | White box (copia locale del NIDS) | TCP | Operatori su pacchetti (split, delay, inject) | Tre detector *anomaly-based* (es. Kitsune) |
| Homoliak et al. [12] | Traffic (connessioni) | Conosce il NIDS | Solo TCP (NetEm + Metasploit) | Operatori di offuscamento *non-payload-based* su connessioni TCP | TPR su cinque classificatori |
| Yaich et al. [14] | Feature | White box | Generico (flussi) | FGSM, BIM, JSMA, C&W su vettori di feature | Metriche su tre dataset; criteri di validità del traffico |

*(continua nella pagina successiva)*

*(segue Tabella 2.1)*

| **Lavoro** | **Spazio** | **Minaccia** | **Trasporto** | **Perturbazioni** | **Valutazione evasione** |
|---|---|---|---|---|---|
| Vitorino et al. (A2PM) [20, 21] | Feature / tabellare | Grey box (feature set) | Generico (Enterprise / IoT) | Pattern adattivi per classe (A2PM) con vincoli di realismo | MLP/RF e RF/XGB/LGBM/IFOR; adversarial training |
| **Questa tesi** | Traffic / PCAP | Black box (blind) | TCP e UDP separati | Padding e durata *separati*; MTU, checksum, esclusione handshake | Evasione blind sulla pipeline del detector; vincoli protocollo a monte |

Un primo gruppo di contributi interviene sul traffico grezzo o sulle catture, anziché sulle sole feature già aggregate. Apruzzese et al. [2] costituiscono il riferimento di partenza: applicano *padding* casuale sul payload di segmenti UDP e TCP con flag PSH, limitando le modifiche al traffico malevolo nelle tracce CTU-13 e misurando l'evasione sulle feature di flusso in presenza di *concept drift*. Rispetto a questo lavoro, la tesi mantiene la minaccia *blind* e l'uso di tracce reali, ma esclude il drift dall'esperimento, separa *padding* e durata in scenari distinti e tratta TCP e UDP separatamente, con vincoli espliciti su *handshake*, MTU del datagramma IP e *checksum*. Galli et al. [8] manipolano catture a livello di pacchetto con *jitter* sui timestamp e *padding* sul payload, ma circoscrivono il dominio al solo Modbus TCP e a tre scenari ICS sul dataset CIC Modbus 2023: la presente analisi generalizza il trasporto oltre il Modbus TCP.

Tre contributi agiscono sul traffico ma presuppongono un modello di minaccia più forte del *blind*. Han et al. [10] formulano l'attacco in *traffic space* come ottimizzazione a due livelli: una GAN cerca *feature adversarial* su un classificatore sostitutivo (sfruttando la trasferibilità) e un secondo passo, basato su PSO, muta il traffico con operatori sugli header (timing dei pacchetti, iniezione di pacchetti craftati) per avvicinarsi a tali feature, sotto un vincolo di *overhead* e mutando solo il traffico dei dispositivi controllati. Pur trattando il modello bersaglio come *black box*, l'attacco richiede comunque di addestrare un classificatore sostitutivo e un estrattore di feature surrogato. Inoltre agisce solo su feature *non-payload*, cioè calcolate da header e statistiche di traffico (dimensioni, numero e tempi dei pacchetti) e non dal contenuto dei pacchetti, su NIDS pensati per attacchi volumetrici o iterativi. Inoltre tratta i protocolli in modo generico (tipo di pacchetto craftato

assegnato in modo casuale, con il solo vincolo di non rompere la comunicazione), mentre la tesi modella separatamente TCP e UDP con vincoli di ammissibilità specifici per protocollo (esclusione dell'handshake, MTU del datagramma IP, ricalcolo del checksum), verificati a monte sul pacchetto e senza alcun modello o estrattore surrogato né ottimizzazione. Yan et al. [22] operano in *black box* con un attacco automatico composto di tre fasi. Nella prima, la *model extraction*, interrogano ripetutamente il bersaglio con query sintetiche e ne addestrano una copia approssimata sotto forma di *ensemble* di modelli surrogati. Nella seconda, avendo pieno accesso ai surrogati, vi generano gli esempi avversari guidando le modifiche al traffico (padding del payload, numero di pacchetti, timing) con un'ottimizzazione che ne riduce il costo. Nella terza riusano gli stessi esempi contro il NIDS reale: poiché un esempio efficace su più modelli diversi tende a esserlo anche su un modello sconosciuto (*trasferibilità*), l'attacco va a segno senza accesso diretto al bersaglio. I vincoli di protocollo che impongono valgono però solo per TCP/IPv4 (ad esempio il limite di 1460 byte sul payload). La tesi, all'opposto, è *blind*: non interroga il bersaglio né costruisce surrogati, applica perturbazioni predeterminate senza ottimizzazione, e modella separatamente TCP e UDP con vincoli di ammissibilità per protocollo (handshake, MTU del datagramma IP, checksum). Hashemi et al. [11] si collocano in un setting *white box*: sfruttano una copia locale del NIDS per provare e selezionare, tramite ricerca, le trasformazioni che abbassano il punteggio di anomalia, attaccando detector *anomaly-based* non supervisionati con operatori di *split*, *delay* e *inject*. Per i detector per-flusso usano invece un attacco basato sul gradiente nello spazio delle feature e ottengono solo un limite inferiore di robustezza, senza garantire che le *feature adversarial* corrispondano a una reale sequenza di pacchetti. La tesi opera invece in *black box* e *blind*, su un classificatore *flow-based* supervisionato, con padding e durata predeterminati su TCP e UDP e realizzabilità verificata sul PCAP prima dell'estrazione delle feature.

Homoliak et al. [12] considerano un attaccante che conosce i dettagli del NIDS e, con gli strumenti NetEm e Metasploit, applicano operatori di offuscamento *non-payload-based* a connessioni TCP (ritardi, perdita, duplicazione, frammentazione), con un'impostazione orientata soprattutto alla difesa, cioè all'addestramento di classificatori consapevoli dell'offuscamento. La loro tecnica resta vincolata al solo TCP, mentre la tesi è *blind*, estende le perturbazioni anche a UDP e valuta l'evasione su PCAP reali con vincoli di protocollo verificati a livello di pacchetto.

Un secondo gruppo non parte dal traffico grezzo. Yaich et al. [14] valutano sette attacchi a

gradiente in *white box* (FGSM, BIM, DeepFool, JSMA, C&W) contro una rete neurale (MLP), applicando le perturbazioni direttamente ai vettori di feature di tre dataset tabellari (NSL-KDD, UNSW-NB15, CIDDS-001). Mostrano che molte feature perturbate violano i vincoli del traffico e introducono quattro criteri di validità, ma li verificano solo a posteriori, senza realizzare i pacchetti. La tesi opera invece in *traffic space* su PCAP in contesto *blind*, senza gradienti né accesso al modello, contro un classificatore *flow-based* supervisionato, e garantisce la validità a monte sul pacchetto (handshake, MTU, checksum) su TCP e UDP separati. Vitorino et al. propongono il metodo A2PM [20] e lo riusano nello scenario IoT [21]: il metodo genera esempi *adversarial* realistici assegnando a ciascuna classe una sequenza di pattern di perturbazione adattivi, che modificano sottoinsiemi di feature rispettando vincoli di dominio e vincoli specifici di classe. Opera però su dati tabellari, perturbando i record di flusso dei dataset CIC-IDS2017 e IoT-23 in setting *grey box*, con la sola conoscenza dell'insieme di feature. Gli esempi generati servono sia per l'*adversarial training* sia per attaccare direttamente i classificatori valutati (MLP, RF, XGB, LGBM e IFOR), con attacchi *targeted* e *untargeted*. A differenza di questo approccio, la tesi opera in *traffic space* su PCAP in contesto *blind*, con padding e durata predeterminati su TCP e UDP e vincoli di realismo verificati sui pacchetti (handshake, MTU, checksum) anziché sul vettore di feature. Inoltre la tesi non impiega *adversarial training*, limitandosi a valutare l'evasione.

Da questa lettura emerge che l'efficacia dell'evasione non dipende soltanto dall'algoritmo del classificatore, ma dallo spazio in cui si costruisce l'attacco e dai vincoli impliciti lungo la pipeline. Rispetto ai lavori in tabella, il contributo di questa tesi si distingue per la combinazione di perturbazioni in *traffic space* su PCAP del solo traffico malevolo controllato dall'attaccante, modello di minaccia *black box* e *blind* senza modelli surrogati né interrogazioni, trattamento separato di TCP e UDP, separazione tra *padding* e durata, e vincoli di ammissibilità espliciti (esclusione dell'*handshake*, MTU del datagramma IP, ricalcolo del *checksum*) verificati a monte dell'estrazione delle feature, sviluppati nel Capitolo 3.

## 2.6 Problem statement

Le sezioni precedenti hanno evidenziato che i ML-NIDS possono ottenere metriche convincenti su dataset di benchmark, ma la loro efficacia non si estende automaticamente a scenari in cui un

attaccante modifica il traffico per eludere il rilevamento. Restano inoltre criticità nelle modalità di valutazione e, in molti lavori, nel realismo del modello di minaccia rispetto a un NIDS operativo. La letteratura misura spesso la robustezza in *feature space*, con variabili di flusso già estratte, oppure assume capacità dell'attaccante difficili da sostenere su un NIDS reale, come l'accesso all'exporter NetFlow o interrogazioni ripetute del classificatore [3, 4]. Queste assunzioni rendono gli esperimenti più gestibili in laboratorio. In un NIDS operativo, invece, l'attaccante realistico non controlla il flow exporter né il classificatore: può limitarsi a modificare il traffico che genera.

Il NIDS opera in genere su un'infrastruttura non accessibile all'attaccante: non può realisticamente controllare la cattura del traffico, l'aggregazione in flussi né la rappresentazione numerica su cui inferisce il classificatore. Più credibile è che modifichi il traffico che genera e che l'effetto dell'attacco si propaghi attraverso la stessa pipeline descritta in §2.2 fino al classificatore. Perturbazioni applicate direttamente alle feature, senza verificarne la realizzabilità sul traffico sottostante, rischiano di sovrastimare l'evasione rispetto a un attacco realistico.

Come illustrato in §2.5.2, il lavoro di partenza [2] combina perturbazioni *blind* e *concept drift* su tracce reali. L'analisi di questa tesi mantiene la minaccia *blind* e le perturbazioni realizzabili, ma valuta l'evasione in *traffic space* su *packet capture* senza studiare il drift nel tempo.

L'obiettivo della tesi è valutare l'efficacia dell'evasione con perturbazioni in *traffic space* su PCAP (`.pcap`). Le modifiche intervengono prima dell'estrazione delle feature e dell'inferenza; i vincoli di protocollo e la coerenza dei pacchetti sono controllati a monte, non solo sulle statistiche di flusso risultanti. A differenza degli scenari *white box*, qui l'attaccante opera in un contesto *black-box*: non accede al ML-NIDS né al sensore.

Il Capitolo 3 presenta il *threat model* in §3.1 (scenario di rete, target ML-NIDS, obiettivo e capacità dell'attaccante), le perturbazioni in *traffic space* in §3.3 e la validazione concettuale in §3.4. Dataset, pipeline sperimentale e metriche sono sviluppati nel Capitolo 4.

# 3. Adversarial attack in *traffic space*

Il *problem statement* del Capitolo 2 (§2.6) [3] ha evidenziato che molte valutazioni di robustezza contro ML-NIDS assumono capacità dell'attaccante poco realistiche, oppure operano in *feature space* su variabili di flusso già estratte. In *traffic space* l'attaccante modifica i frame registrati in un PCAP, cioè il traffico grezzo prima dell'aggregazione in flussi. Un operatore di rete o un sistema di cattura non espone all'attaccante remoto un'interfaccia per alterare arbitrariamente features come `TotBytes` o `Dur` di un record di flusso: quelle quantità derivano da pacchetti effettivamente trasmessi e osservati. Ciascuna perturbazione deve corrispondere a un frame con header e payload validi, vincoli di protocollo rispettati e checksum allineati ai byte presenti; solo dopo questa fase il traffico perturbato può essere aggregato in flussi, trasformato nel vettore di feature su cui inferisce il ML-NIDS, come nella pipeline di §2.2. L'evasione non è quindi assunta per definizione: perturbazioni conformi al protocollo devono produrre, dopo l'estrazione, uno spostamento delle feature sufficiente a indurre una decisione errata del classificatore.

Si formalizza il *threat model* adottato (§3.1): scenario di rete, target dell'attaccante (ML-NIDS), obiettivo, conoscenza del sistema target, capacità operative e strategie di perturbazione valutate separatamente. Per interpretare dove e con quali regole tali modifiche sono ammissibili, la sezione sui livelli di rete e la cattura del traffico (§3.2) richiama la struttura dei frame in PCAP, il passaggio da frame a record di flusso e i vincoli di protocollo che delimitano le perturbazioni ammesse.

Le due famiglie di attacco sono poi sviluppate in §3.3 (padding del payload in §3.3.2, perturbazione sulla durata in §3.3.3). Infine, §3.4 riassume i criteri di accettazione (dettagliati in §3.3), illustra con esempi come l'effetto si propaga fino alle grandezze di flusso su cui inferisce il classificatore (§3.4.2) e delimita i limiti di un attacco operato offline sulla cattura. L'implementazione concreta delle perturbazioni, il dataset e il preprocessing del classificatore sono descritti nel capitolo successivo. La Figura 3.1 riassume il percorso adottato nel capitolo: attaccante *blind*, editing *offline* su PCAP, due famiglie di perturbazione valutate separatamente, vincoli di protocollo e propagazione fino all'inferenza del classificatore.

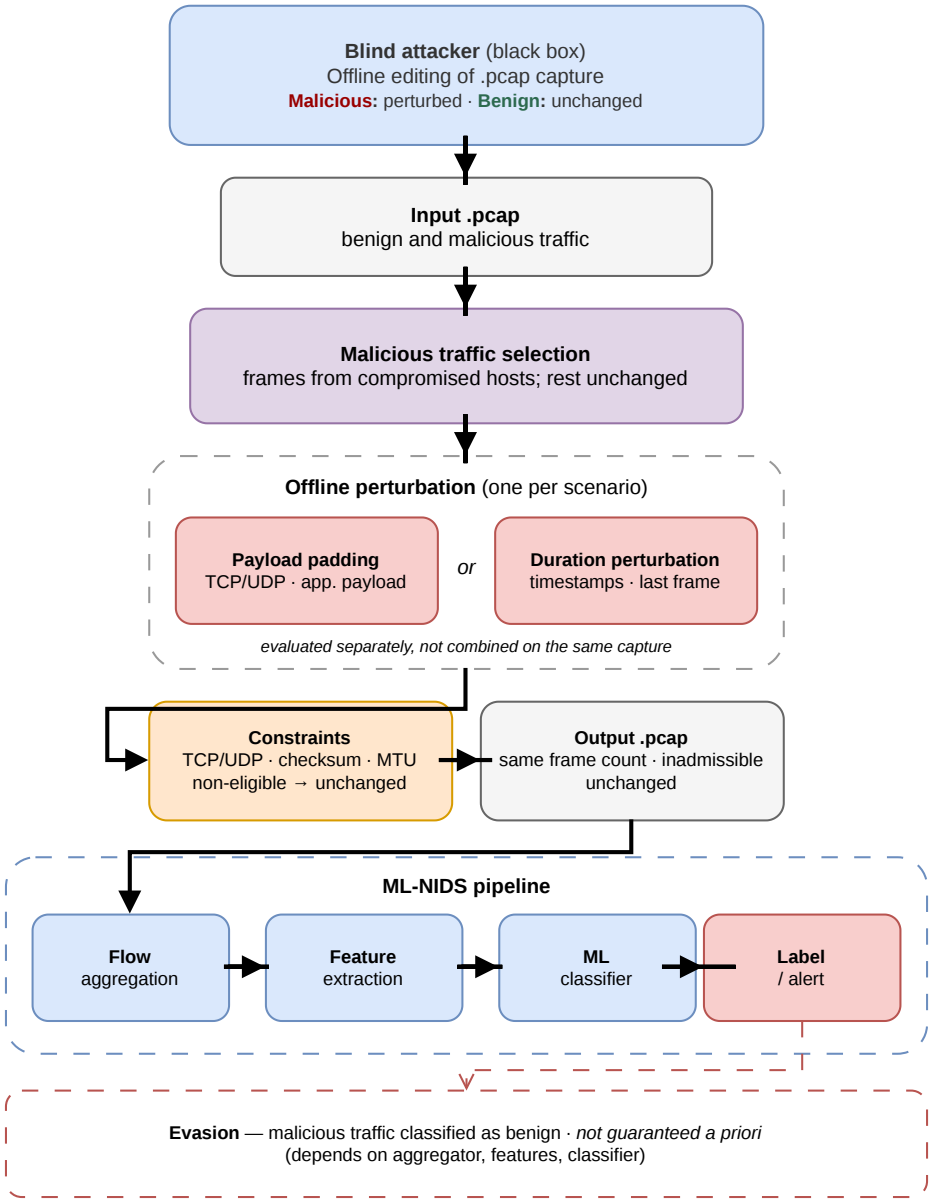



Figura 3.1: Panoramica del capitolo: attaccante *blind* che applica perturbazioni *offline* su un PCAP (traffico malevolo selezionato e modificato, benigno invariato); padding del payload *oppure* ritardo di invio dei pacchetti, valutati separatamente e soggetti a vincoli TCP/UDP, checksum e MTU; a valle, aggregazione in flussi, estrazione delle feature e inferenza del classificatore. L'evasione non è garantita a priori e dipende anche dall'aggregatore e dal set di feature adottati dal NIDS.

## 3.1 Threat model

Il *threat model* adottato in questa tesi formalizza chi è l'attaccante, quale obiettivo persegue e quali azioni può compiere sul traffico, nel rispetto del *problem statement*. Come nel lavoro di partenza [2], si considerano perturbazioni *blind* e realizzabili sul traffico malevolo. L'effetto del *concept drift* nel tempo non è oggetto di questa analisi. La Figura 3.2 introduce lo scenario operativo e il confine di fiducia tra ciò che l'attaccante controlla e ciò che resta fuori dalla sua portata.

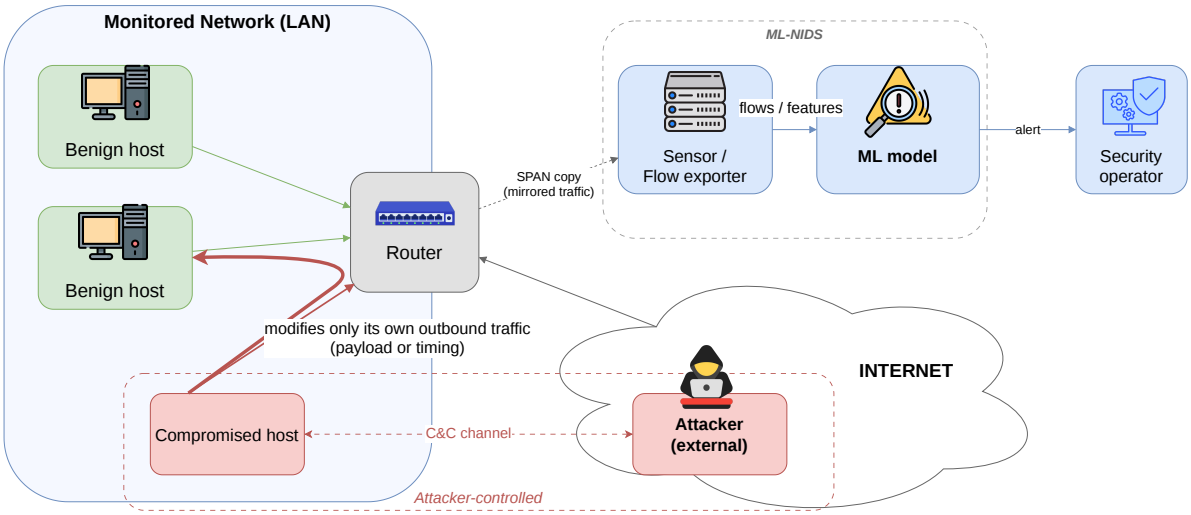



Figura 3.2: *Threat model*: L'attaccante è esterno alla rete monitorata e ne controlla un host interno tramite un canale di *Command and Control* (C&C). Solo l'host compromesso e il suo traffico in uscita sono sotto il controllo dell'attaccante (rosso), mentre l'infrastruttura di rilevamento (sensore ed *exporter* di flussi, ML-NIDS, operatore di sicurezza) e gli host benigni (verde) restano fuori dalla sua portata.

La sezione si articola come segue: lo scenario di rete (§3.1.1), il target ML-NIDS (§3.1.2), l'obiettivo dell'attaccante (§3.1.3), la conoscenza del sistema (§3.1.4), le capacità (§3.1.5) e le strategie di perturbazione valutate (§3.1.6).

### 3.1.1 Scenario di rete

Nel contesto operativo si considera una rete locale monitorata, tipicamente una LAN aziendale o un segmento interno di un'organizzazione. Gli host interni, workstation o server con sistemi operativi diffusi (ad esempio Windows o Linux), sono collegati a uno switch di accesso e raggiungono Internet e reti esterne tramite un router di confine, di solito preceduto o affiancato da un firewall. L'attaccante remoto non agisce direttamente sull'infrastruttura di rete: compromette almeno un host interno e lo controlla tramite un canale di *Command and Control* (C&C).

Più host generano traffico di routine. Almeno uno è compromesso e origina, oltre all'attività legittima, comunicazioni malevole verso l'esterno o verso altri nodi. Il sensore del NIDS non intercetta il percorso dei pacchetti: riceve una copia del transito osservato sullo switch mediante *port mirroring* (SPAN), come nella Figura 2.1 del Capitolo 2. La stessa finestra di osservazione può quindi includere traffico benigno di più host e traffico malevolo dell'host compromesso.

L'attaccante non controlla switch, router di confine, firewall, sensore né il resto dell'infrastruttura di rilevamento. Può intervenire solo sul traffico in uscita dall'host sotto il proprio controllo.

Nella valutazione sperimentale si utilizzano acquisizioni pubbliche del Malware Capture Facility Project (CTU-13 e tracce MCFP più recenti) [9, 15]: reti universitarie o di laboratorio in cui il malware gira su macchine virtuali Windows, affiancate a traffico benigno di altre workstation e servizi interni (nella stessa acquisizione per CTU-13, in acquisizioni dedicate per le tracce MCFP del 2017). Il dettaglio delle catture è nel Capitolo 4. L'editing *offline* dei PCAP non modifica questa cornice: si chiede se perturbazioni realizzabili sull'host compromesso avrebbero eluso il rilevamento su traffico dello stesso tipo che un sensore passivo registrerebbe in rete.

### 3.1.2 Target dell'attaccante

Il bersaglio è un ML-NIDS collocato in modo passivo sulla rete monitorata (§3.1.1). Si tratta di un NIDS che integra un modello di *machine learning* addestrato a inferire sul traffico osservato [3], secondo la pipeline introdotta in §2.2 e riassunta in Figura 2.2.

Il sensore registra i frame che transitano in rete. Un aggregatore di flussi raggruppa i pacchetti per chiave di sessione (quintupla IP, porte, protocollo di trasporto) e ne ricava statistiche di volume e durata. Un modulo di estrazione produce, per ciascun flusso, un vettore numerico di feature.

Un classificatore supervisionato, addestrato su flussi etichettati come benigni o malevoli, assegna un'etichetta o un punteggio di rischio. Se un flusso è classificato come malevolo, il sistema emette un allarme verso operatori di sicurezza o verso un SIEM.

L'obiettivo operativo del ML-NIDS è intercettare comunicazioni sospette nel traffico osservato, mentre, l'attaccante punta al contrario: far etichettare come benigno traffico malevolo che, senza perturbazione, sarebbe rilevato (§3.1.3).

Il target non è solo il modello, ma l'intera catena acquisizione, aggregazione in flussi, estrazione delle feature, inferenza e generazione dell'allarme. Perturbazioni realizzabili sui pacchetti dell'host compromesso non sono sufficienti per evadere se l'aggregatore o il set di feature adottati dal NIDS non traducono la modifica in uno spostamento della decisione del classificatore.

Nella valutazione sperimentale i PCAP sono aggregati in flussi con Argus, che assume il ruolo di aggregatore nella pipeline descritta sopra. Sui record così ottenuti si addestrano e si valutano tre classificatori supervisionati (Capitolo 4).

### 3.1.3 Obiettivo dell'attaccante

L'obiettivo dell'attaccante è l'*evasione* del ML-NIDS [4]: indurre il classificatore a etichettare come benigno traffico che, in assenza di perturbazione, risulterebbe sospetto o malevolo. Non si considerano attacchi di poisoning in fase di addestramento né la compromissione del sensore o del modello interno al dispositivo di rilevamento. Come descritto in §3.1.2, l'evasione riguarda l'intera pipeline fino all'allarme, non il solo classificatore.

### 3.1.4 Conoscenza del sistema target

Si assume un attaccante in contesto *black box* [4]: non ha accesso all'architettura del NIDS, al modello addestrato, all'insieme di feature né alle soglie di decisione e non può interrogare ripetutamente il classificatore per ottimizzare le perturbazioni.

L'attaccante conosce a livello generale il contesto operativo: il traffico sotto analisi è aggregato in flussi e sottoposto a un classificatore basato su *machine learning*, in linea con le architetture ML-NIDS considerate nel capitolo precedente. Non sono noti all'attaccante l'algoritmo di classificazione, l'elenco delle feature di flusso né i parametri del modello usati dal NIDS.

### 3.1.5 Capacità dell'attaccante

L'attaccante dispone del controllo di almeno un host compromesso nella rete monitorata: può alterare payload o ritmo di invio del proprio traffico in uscita prima che i pacchetti siano osservati da un ML-NIDS collocato in modo passivo sulla rete (§3.1.1). Non può alterare comunicazioni di altri host né il sensore, l'exporter di flussi, il preprocessore interno al NIDS né le feature di flusso già estratte dal sistema di rilevamento.

In coerenza con la distinzione discussa in §2.5.2 [18], le perturbazioni rappresentano alterazioni realizzabili dal malware sull'host malevolo, non manipolazioni irrealistiche del file di cattura sul percorso di rete.

Per valutare l'evasione, su un PCAP storico si applicano perturbazioni *offline* limitatamente al traffico malevolo, producendo un PCAP perturbato distinto da quello originale; il file risultante non viene ritrasmesso in rete. Si riesegue la catena di aggregazione in flussi, estrazione delle feature e inferenza descritta per gli ML-NIDS, chiedendosi se pacchetti così modificati avrebbero indotto una decisione errata del classificatore.

### 3.1.6 Strategie di attacco

Si valutano due strategie di perturbazione distinte, entrambe definite in *traffic space* sulla cattura del traffico malevolo, valutate separatamente e non combinate sullo stesso file. In entrambi i casi le modifiche devono rispettare i vincoli dei protocolli di trasporto (TCP e UDP); nel contesto operativo corrispondono ad azioni che un host compromesso può compiere sul proprio traffico, mentre nella valutazione sperimentale si traducono in editing offline della cattura, come descritto in §3.1.5. Il dettaglio implementativo è rinviato al capitolo successivo e alle sezioni dedicate in §3.3. La prima strategia agisce sul *payload applicativo* dei pacchetti ammissibili, incrementando i byte trasportati per frame. L'obiettivo è spostare verso l'alto le statistiche di volume del flusso (ad esempio `TotBytes` e grandezze correlate) dopo l'aggregazione. Operativamente, corrisponde a un host compromesso che invia segmenti con contenuto applicativo più lungo, entro i limiti del protocollo e dell'MTU del percorso.

La seconda strategia simula un ritardo nell'invio dei pacchetti di una conversazione malevola, posticipando la trasmissione di frame selezionati senza alterare payload, header o checksum.

L'obiettivo è allungare l'intervallo temporale del flusso dopo l'aggregazione (ad esempio `Dur` e statistiche normalizzate rispetto al tempo), a parità di byte trasportati. Poiché la durata è una grandezza derivata dalle regole dell'aggregatore, l'effetto di questa strategia dipende anche da come il sensore del NIDS delimita e chiude i flussi.

L'efficacia dell'evasione non è garantita a priori. Dipende da come le modifiche sui pacchetti si propagano, dopo l'aggregazione, verso le grandezze di flusso e il vettore di feature su cui inferisce il classificatore; dipende inoltre dall'aggregatore o dall'exporter adottato dal NIDS e dall'insieme di feature estratte, aspetti che l'attaccante non controlla. Lo stesso frame perturbato può quindi produrre effetti diversi a seconda del sensore e del preprocessore, e solo in ultima istanza la decisione del modello addestrato su quella rappresentazione determina se l'evasione riesce.

Le due strategie parallele, i vincoli di protocollo e la pipeline ML-NIDS fino all'esito di evasione sono sintetizzati in Figura 3.1.

## 3.2 Livelli di rete e cattura del traffico

Le perturbazioni del *threat model* intervengono su frame e timestamp in un PCAP alterato *offline*. Per chiarire dove e con quali regole tali modifiche sono ammissibili, occorre analizzare come il traffico è strutturato a livello di protocollo, come viene registrato in cattura e come, a valle, viene aggregato in flussi su cui opera il ML-NIDS.

### 3.2.1 Da frame a flusso

In questa tesi perturbazioni ed esperimenti riguardano solo flussi TCP o UDP su catture Ethernet con traffico IPv4. Ogni scenario adotta uno dei due protocolli di trasporto, non entrambi insieme. Nella stessa acquisizione possono comparire anche altri frame, ad esempio ICMP, che restano invariati e non sono inclusi nei dataset di addestramento e di test. I criteri ed il dettaglio della costruzione dei dataset sono nel capitolo successivo. Per il traffico TCP/UDP oggetto di studio, ogni frame contiene un header Ethernet, un datagramma IP, l'header di trasporto e il payload applicativo. La Figura 3.3 richiama il modello a strati (ISO/OSI) nei livelli rilevanti e mostra come, per incapsulamento, si forma il frame e quali porzioni compaiono in un PCAP; la Figura 3.4 ne illustra un esempio concreto.

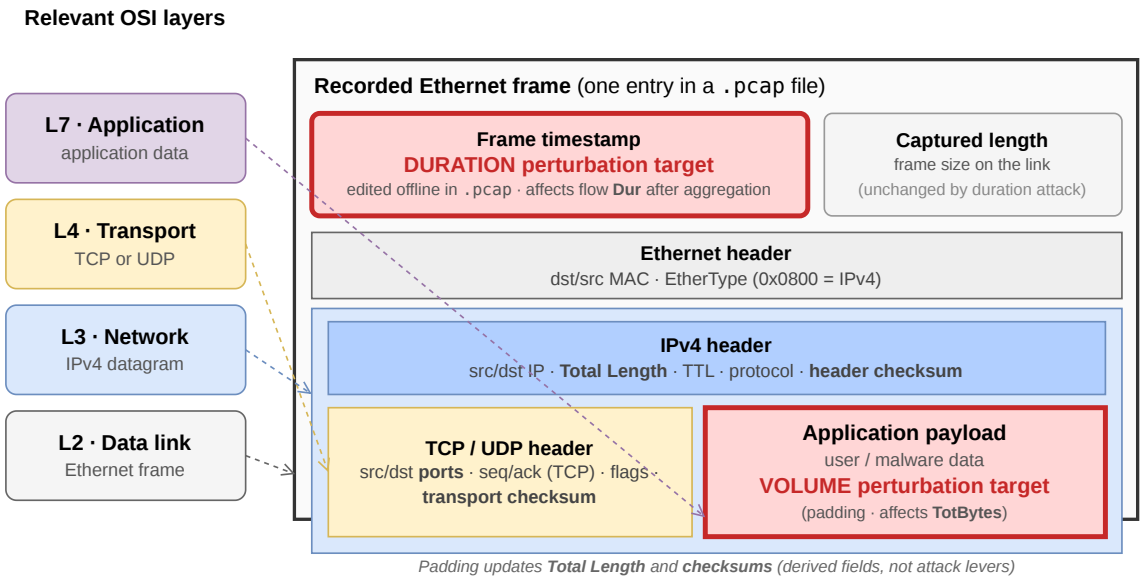



Figura 3.3: Modello a strati semplificato e composizione di un frame in cattura: incapsulamento da payload applicativo a frame Ethernet; evidenziati payload (bersaglio del padding) e timestamp del frame (bersaglio della perturbazione sulla durata).

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000 | 147.32.84.165 | 74.125.232.195 | TCP | 62 | 1027 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM |
| 2 | 0.000011 | 147.32.84.165 | 74.125.232.195 | TCP | 62 | [TCP Retransmission] 1027 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM |
| 3 | 0.008046 | 74.125.232.195 | 147.32.84.165 | TCP | 62 | 80 → 1027 [SYN, ACK] Seq=0 Ack=1 Win=5720 Len=0 MSS=1430 SACK_PERM |
| 4 | 0.008293 | 147.32.84.165 | 74.125.232.195 | TCP | 60 | 1027 → 80 [ACK] Seq=1 Ack=1 Win=64350 Len=0 |
| 5 | 0.008303 | 147.32.84.165 | 74.125.232.195 | TCP | 60 | [TCP Dup ACK 4#1] 1027 → 80 [ACK] Seq=1 Ack=1 Win=64350 Len=0 |
| 6 | 0.008594 | 147.32.84.165 | 74.125.232.195 | HTTP | 447 | GET /service/check2?appid=%7B430FD4D0-B729-4F61-AA34-91526481799D%7D&appversion=1.3 |
| 7 | 0.008604 | 147.32.84.165 | 74.125.232.195 | TCP | 447 | [TCP Retransmission] 1027 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64350 Len=393 |
| 8 | 0.016648 | 74.125.232.195 | 147.32.84.165 | TCP | 60 | 80 → 1027 [ACK] Seq=1 Ack=394 Win=6432 Len=0 |
| 9 | 0.042295 | 74.125.232.195 | 147.32.84.165 | HTTP | 131 | HTTP/1.1 204 No Content |
| 10 | 0.045126 | 147.32.84.165 | 74.125.232.195 | TCP | 60 | 1027 → 80 [RST, ACK] Seq=394 Ack=78 Win=0 Len=0 |
| 11 | 0.045136 | 147.32.84.165 | 74.125.232.195 | TCP | 60 | 1027 → 80 [RST, ACK] Seq=394 Ack=78 Win=0 Len=0 |

Figura 3.4: Esempio di sequenza di frame della stessa conversazione TCP in un PCAP. Le colonne *Time* e *Length* riportano rispettivamente il timestamp del frame e la dimensione registrata in cattura; *Info* riassume la quintupla di trasporto e il contenuto protocollo rilevante.

Un PCAP non è un record di flusso né un export NetFlow: è una sequenza ordinata di frame registrati dal sensore o da uno strumento di acquisizione, ciascuno con un timestamp. Più frame con la stessa quintupla sorgente/destinazione (indirizzi IP, porte, protocollo) e intervallo temporale coerente con la politica di aggregazione del sistema di analisi costituiscono, dopo l'aggregazione, un *flusso* le cui statistiche (contatori di byte e pacchetti, durata, flag di protocollo) alimentano il

classificatore. La conversazione TCP della Figura 3.4 ne offre un esempio concreto: ciascun frame ha un timestamp e una lunghezza registrati in cattura. Dopo l'aggregazione, il flusso riassume il traffico osservato con contatori di pacchetti e byte; la durata è una grandezza derivata dai timestamp secondo le regole del flow exporter adottato, che possono variare tra sistemi (timeout, chiusura di sessione, istante del primo e dell'ultimo pacchetto considerato). La Figura 3.5 riporta il record risultante per quella conversazione, con quintupla, `TotPkts`, `TotBytes` e `Dur`; il dettaglio del flow exporter è nel capitolo sperimentale.

| Dur | SrcAddr | Sport | DstAddr | Dport | TotPkts | TotBytes | Proto |
|---|---|---|---|---|---|---|---|
| **0.045136** | 147.32.84.165 | 1027 | 74.125.232.195 | 80 | 11 | 1511 | tcp |

Figura 3.5: Record di flusso ottenuto aggregando gli undici frame della conversazione TCP di Figura 3.4: quintupla, contatori di pacchetti e byte e durata sono grandezze sintetiche calcolate dall'aggregazione. Le colonne visibili costituiscono un sottoinsieme esemplificativo di quelle che un export di flusso può contenere.

Le grandezze su cui il classificatore opera a livello di flusso derivano quindi da quanto registrato frame per frame nella cattura, non da valori assegnati direttamente a un record di export. Le perturbazioni di questa tesi intervengono sui singoli frame della cattura *prima* di tale aggregazione; l'effetto sulle grandezze di flusso è quindi indiretto e dipende da come le modifiche si riflettono sui contatori e sugli intervalli temporali calcolati a valle.

### 3.2.2 Struttura del pacchetto e campi rilevanti

L'header IP include, tra gli altri campi, la lunghezza totale del datagramma e un checksum di integrità calcolato sull'header e sul payload di rete; sotto IPv4, TCP e UDP introducono header di trasporto (porte, numeri di sequenza e flag nel caso TCP, checksum di trasporto) e un payload applicativo.

L'unità massima di trasmissione (MTU, *Maximum Transmission Unit*) tipica del percorso, pari a 1500 byte nei casi più comuni, limita la dimensione massima del datagramma IPv4 non frammentato; va distinta dalla dimensione del frame in cattura, che include anche l'header di link layer oltre al

datagramma IP. Allungare il payload applicativo incrementa la lunghezza del datagramma IP e, di conseguenza, quella del frame sul link.

Per le perturbazioni di volume considerate in questa tesi, l'unico punto di modifica intenzionale è il *payload applicativo* del segmento TCP o del datagramma UDP: non si alterano arbitrariamente gli header Ethernet, IP o di trasporto per simulare l'attacco. Nel modello adottato, un host compromesso può allungare i dati che invia a livello applicativo; modificare porte, indirizzi, flag o numeri di sequenza equivarrebbe invece a forgiare un'altra conversazione o a produrre frame incoerenti con la sessione osservata, al di fuori delle capacità dell'attaccante formalizzate in §3.1.5. In conseguenza dell'aggiunta di byte variano anche alcuni campi di header, in particolare la lunghezza del datagramma IP e le checksum di rete e trasporto. Tali campi non sono leve dell'attacco ma aggiornamenti necessari alla coerenza del protocollo. In transito, uno stack TCP/IP che rileva checksum errati o lunghezze incoerenti scarta il datagramma: il frame non viene accettato dal destinatario e non contribuisce al traffico effettivamente consegnato a livello applicativo, pur potendo essere stato registrato da un sensore passivo che osserva una copia del link. In una cattura modificata *offline*, invece, frame inconsistenti possono restare memorizzati nel file perché nessun componente di rete valida il contenuto al momento della scrittura; nel modello adottato ogni padding ammesso richiede quindi la ricostruzione coerente del datagramma, altrimenti la perturbazione non rientra nel perimetro di realizzabilità precisato in §3.3.2.

Non ogni frame è candidato al padding; le regole di selezione sono dettagliate in §3.3.2. Analogamente, non sono oggetto delle perturbazioni di volume i frame privi del livello IP o del livello di trasporto previsto. Per la perturbazione sulla durata non si modificano byte nel payload, ma i timestamp con cui i frame risultano registrati in cattura; da lì, dopo l'aggregazione in flusso, cambiano le grandezze temporali del flusso, come la durata. Operativamente, un host compromesso può introdurre ritardi tra l'invio di segmenti consecutivi del proprio traffico, in modo che il sensore osservi un intervallo temporale più ampio tra il primo e l'ultimo frame della conversazione a parità di byte trasportati. Nella valutazione sperimentale questo comportamento è modellato alterando *offline* il timestamp dell'ultimo frame ammissibile di ciascun flusso malevolo, lasciando invariati payload, header e checksum; le regole di selezione sono dettagliate in §3.3.3. La Figura 3.6 schematizza la composizione di un frame con datagramma IP e segmento TCP, evidenziando il payload applicativo come unico bersaglio del padding.

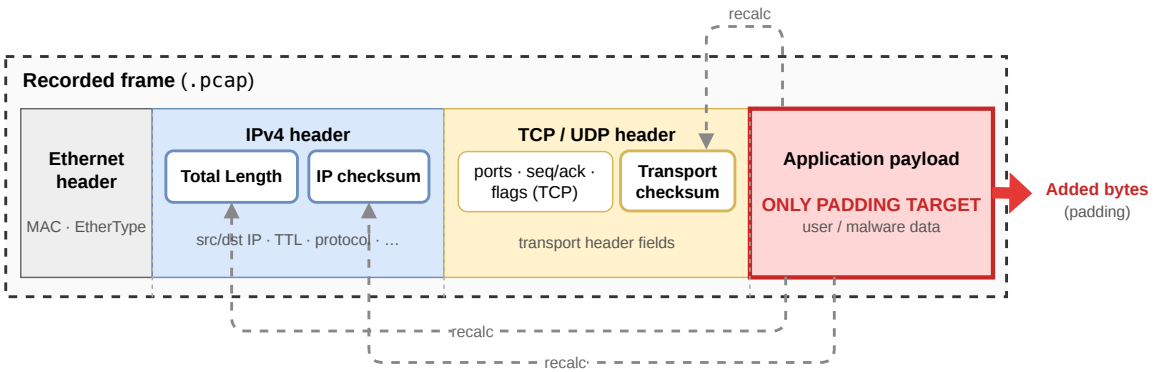



Figura 3.6: Anatomia di un frame con datagramma IP e segmento TCP o UDP: il payload applicativo è l'unico bersaglio del padding; un incremento del payload richiede l'aggiornamento di lunghezze e checksum.

### 3.2.3 Vincoli di protocollo sulle modifiche

Il padding è vincolato dall'MTU del percorso: la lunghezza totale del datagramma IP, dopo l'aggiunta, non può superare quel limite. L'incremento ammesso su ciascun frame è quindi al più pari allo spazio residuo tra la lunghezza attuale del datagramma e l'MTU. Se il margine non consente l'aggiunta prevista, il frame non può essere esteso in modo conforme al protocollo e, nel modello adottato, resta invariato piuttosto che essere scartato. Si tratta di una scelta cautelativa coerente con il *threat model*: l'attaccante applica l'incremento di volume solo dove resta realizzabile entro l'MTU, senza frammentare arbitrariamente il datagramma, superare il limite del percorso o rimuovere pacchetti dalla cattura. Omettere la perturbazione su un frame candidato ma non estendibile limita l'effetto sulle feature del flusso, ma evita di introdurre traffico potenzialmente invalido o di alterare la struttura della conversazione in modi difficili da giustificare per un host compromesso che punta all'evasione senza interrompere la propria comunicazione.

Le perturbazioni non rimuovono frame dalla cattura; i frame su cui il padding non si applica restano invariati. Operando *offline*, ogni padding ammesso ricostruisce il datagramma in modo che il frame risultante resti conforme al protocollo (Figura 3.6).

## 3.3 Perturbazioni in *traffic space*

Le due strategie anticipate nel *threat model* e i vincoli di protocollo corrispondono a due famiglie di perturbazione *offline* sui frame della cattura: l'arricchimento del payload applicativo (padding) e il ritardo dell'invio di frame selezionati, modellato aggiornando i timestamp di registrazione in cattura. In entrambi i casi la modifica interviene sui singoli pacchetti prima dell'aggregazione in flussi. Entrambe operano su file PCAP in ingresso e producono un PCAP perturbato con lo stesso numero di frame. Il dettaglio operativo (strumenti, configurazioni numeriche, dataset) è descritto nel Capitolo 4; qui si fissano principi, regole di applicazione ed effetto atteso sulle grandezze di flusso.

### 3.3.1 Principi comuni

Indipendentemente dal tipo di perturbazione, il modello adottato condivide le seguenti regole: la cattura in ingresso contiene traffico benigno e malevolo. Le modifiche riguardano solo i frame inviati dagli host malevoli, mentre, il resto dell'acquisizione resta invariato. Ogni scenario sperimentale considera un solo protocollo di trasporto alla volta (TCP oppure UDP). I frame che non soddisfano i vincoli del tipo di perturbazione scelto (payload assente, margine MTU insufficiente, flusso non ammissibile per la durata, ecc.) restano identici a quelli registrati in origine: non vengono eliminati dalla cattura in uscita. Le perturbazioni di volume e di durata sono valutate separatamente: non si combinano padding e ritardo temporale sullo stesso file di cattura negli scenari considerati.

### 3.3.2 Padding del payload

La perturbazione di volume aggiunge un numero fisso di byte al *payload applicativo* dei segmenti TCP o dei datagrammi UDP ammissibili. Un frame è candidato solo se appartiene allo scenario di trasporto considerato (TCP oppure UDP), ha sorgente tra gli host compromessi e trasporta già un payload applicativo non nullo: restano esclusi segmenti di controllo privi di dati, come quelli dell'handshake TCP o gli ACK senza payload, perché non offrono un payload da allungare e alterarli equivarrebbe a modificare la semantica di sessione al di fuori delle capacità dell'attaccante (§3.1.5). Analogamente, non si paddano datagrammi UDP vuoti né frame la cui sorgente non coincide con un

host sotto controllo dell'attaccante: quest'ultimo può allungare solo i dati che invia dal proprio host, non le risposte del destinatario né il traffico di altri nodi nella stessa acquisizione. La Figura 3.7 illustra, su un flusso TCP malevolo, quali segmenti sono candidati al padding (quelli con payload applicativo in uscita dall'host compromesso) e quali restano invariati (handshake, ACK senza dati e traffico del server).

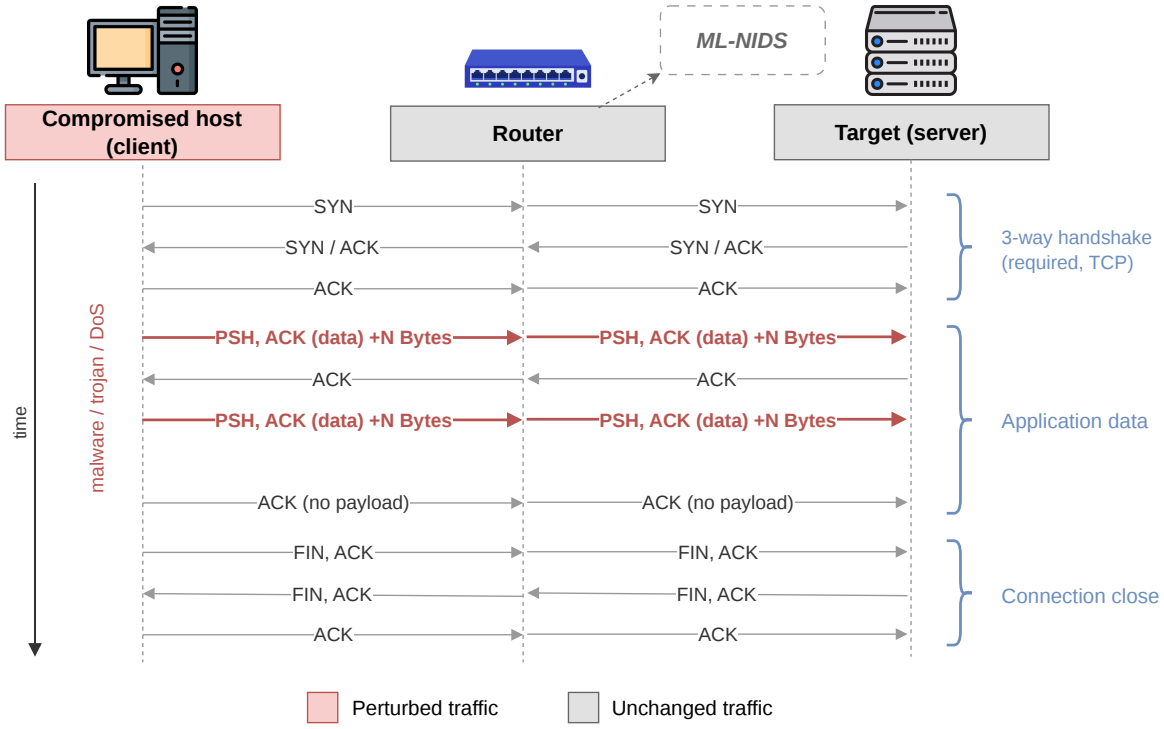



Figura 3.7: Strategia di padding su un flusso TCP malevolo: vengono allungati di un incremento fisso (+N byte) solo i segmenti con payload applicativo in uscita dall'host compromesso (client, in rosso). Handshake, ACK senza dati e le risposte del server restano invariati.

All'interno di una conversazione bidirezionale malevola, la perturbazione non riguarda quindi tutti i frame del flusso, ma solo quelli in uscita dall'host compromesso che soddisfano i criteri precedenti; su ciascuno di essi si tenta di applicare l'incremento configurato, mentre gli altri frame della cattura restano identici a quelli originali. L'intensità del padding è parametrizzata per scenario con un valore costante (configurazioni di intensità crescente, dalla perturbazione minima fino a incrementi dell'ordine di centinaia o migliaia di byte): su ogni frame candidato si tenta di

aggiungere esattamente l'incremento configurato, non un intervallo casuale. Prima di accettare la modifica si verifica che la lunghezza totale del datagramma IP, dopo l'aggiunta prevista, non superi l'MTU del percorso: se il margine residuo è insufficiente per l'intero incremento, il frame non viene alterato. Quando il padding è applicato, il datagramma viene ricostruito aggiornando lunghezze e checksum di livello IP e trasporto, in modo che il frame risultante resti conforme al protocollo pur essendo stato prodotto *offline* sulla cattura. A valle dell'aggregazione, l'effetto atteso è un aumento delle statistiche di volume del flusso, proporzionale al numero di frame effettivamente paddati e all'intensità configurata. Il passaggio dal PCAP perturbato al record di flusso, con aumento di `TotBytes` rispetto a Figura 3.5, è illustrato in Figura 3.9 (§ 3.4.2).

In uno scenario operativo, l'intensità dell'incremento è scelta dall'attaccante come parametro esplicito. Strumenti come Nmap consentono di aggiungere un numero fisso di byte ai pacchetti inviati, sia su UDP sia su TCP, mediante l'opzione `–data-length`. Ad esempio, cento byte aggiuntivi si ottengono con `nmap -sU –data-length 100 -p 53 host` su UDP o con `nmap -sS –data-length 100 -p 80 host` su TCP. Su TCP, l'incremento può ricadere sui segmenti inviati dall'host attaccante, inclusi i SYN di apertura sessione; nel modello adottato il padding offline riguarda invece i segmenti che trasportano già payload applicativo, escludendo i segmenti di controllo del handshake. Nmap accetta valori elevati di `–data-length`, ma non applica un controllo esplicito sul margine MTU del percorso: se la lunghezza totale del datagramma supera l'MTU tipico (1500 byte su Ethernet), il pacchetto può essere frammentato in transito oppure non essere inviato dallo stack di rete, a differenza del modello adottato in valutazione, dove un frame il cui margine residuo è insufficiente resta invariato (§3.2.3).

### 3.3.3 Perturbazione sulla durata del flusso

La seconda famiglia simula un ritardo nell'invio di frame selezionati, modellato aggiornando i timestamp di registrazione in cattura, senza alterare payload, header, lunghezze o checksum. L'obiettivo è allungare la durata osservata del flusso dopo l'aggregazione. I frame sono raggruppati per conversazione bidirezionale, cioè per l'insieme registrato in cattura con la stessa quintupla (indirizzi IP, porte e protocollo di trasporto) che, dopo l'aggregazione, costituisce un unico flusso. Una conversazione è ammissibile se appartiene allo scenario di trasporto considerato (TCP oppure UDP) e, nel caso TCP, presenta un inizio di sessione riconoscibile tramite handshake a tre vie

(3WHS). Per UDP non è richiesto un handshake analogo. Restano escluse le conversazioni prive di 3WHS completo (TCP) o il cui ultimo frame in cattura non proviene da un host compromesso: l'attaccante può ritardare solo il traffico in uscita dal proprio host, non i segmenti inviati dall'altro host della conversazione (§3.1.5). La Figura 3.8 illustra il criterio su una conversazione TCP malevola: a parità di payload, viene ritardato di Δ il solo ultimo frame, ammesso solo se la sua sorgente è l'host compromesso e se la conversazione presenta apertura (3WHS) e chiusura (4WHS o RST) riconoscibili.

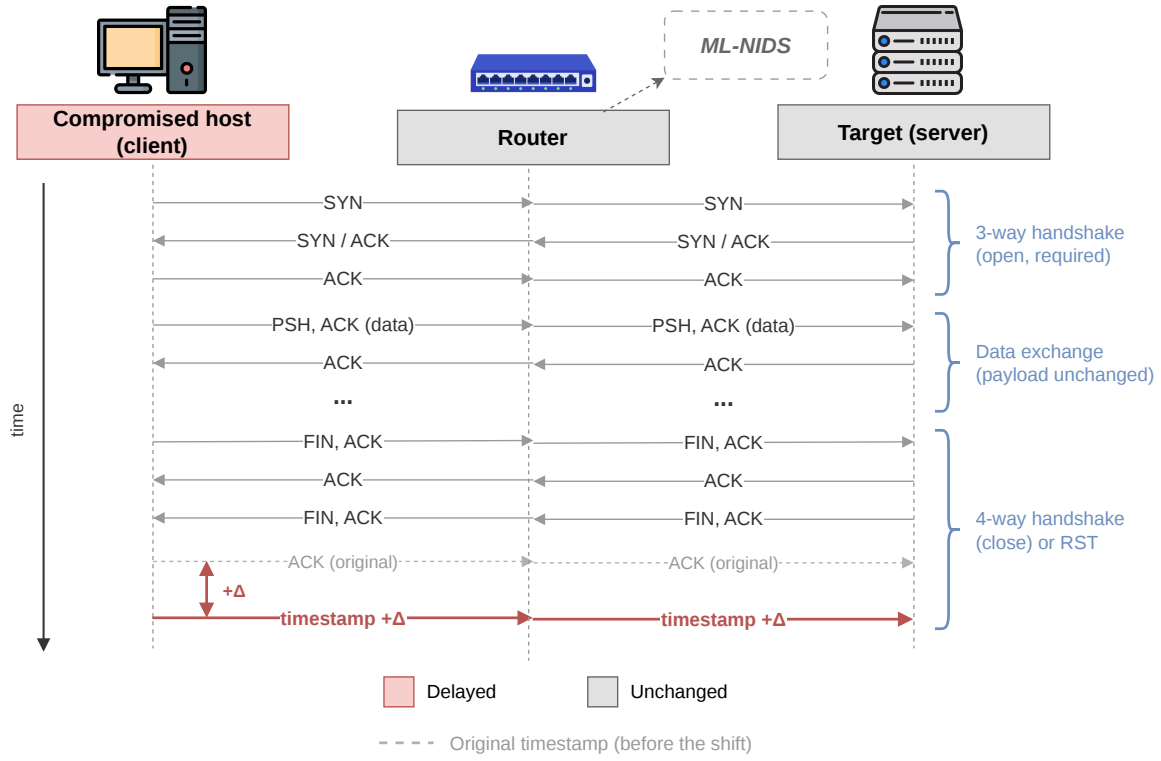



Figura 3.8: Strategia di perturbazione sulla durata su un flusso TCP malevolo: a parità di payload viene ritardato di Δ il solo ultimo frame del flusso (in rosso), allungando `Dur` dopo l'aggregazione. Il flusso è ammissibile solo con 3WHS in apertura e 4WHS (o RST) in chiusura riconoscibili e con l'ultimo frame inviato dall'host compromesso.

Per ciascuna conversazione ammissibile, la perturbazione colpisce un solo frame, l'ultimo per ordine temporale registrato in cattura e la cui sorgente è l'host malevolo. I timestamp degli altri frame della conversazione e dell'acquisizione restano invariati. Si assume che l'aggregatore calcoli

la durata del flusso come intervallo tra il timestamp del primo e quello dell'ultimo frame osservato nella conversazione (§3.2.1): spostare avanti di un ritardo $\Delta$ solo l'istante dell'ultimo frame allunga `Dur` di circa $\Delta$ a parità di byte e pacchetti trasportati. Ritardare invece frame intermedi rischierebbe di invertire l'ordine temporale registrato (ad esempio una richiesta che risulta posteriore alla relativa risposta) e di alterare la semantica della sessione in modi difficili da giustificare. Per questo il modello concentra la perturbazione sull'estremo temporale del flusso.

In uno scenario operativo, l'effetto corrisponde a ritardare la chiusura della connessione o l'ultimo invio in uscita dall'host compromesso: ad esempio, un client che attende prima di chiudere il socket TCP (equivalente a un `sleep` prima di `close()`, che fa slittare l'invio del segmento FIN di chiusura) o che posticipa l'ultimo datagramma UDP della conversazione. Su UDP, l'analogo è inviare l'ultimo datagramma della sessione dopo una pausa esplicita. In valutazione sperimentale, lo stesso effetto è modellato spostando avanti il timestamp di quell'unico frame finale ammissibile, cioè ritardando in cattura la fine osservata della conversazione senza modificare l'ordine relativo degli altri frame.

L'intensità del ritardo è parametrizzata per scenario con un valore fisso, da 1 millisecondo fino a 1 secondo. Dopo l'aggregazione, l'effetto atteso è un allungamento di `Dur` e, indirettamente, una variazione delle feature normalizzate rispetto al tempo (ad esempio byte o pacchetti al secondo). Un esempio concreto dell'allungamento di `Dur` a parità di byte trasportati, rispetto al record di Figura 3.5, è in Figura 3.10 (§ 3.4.2).

### 3.3.4 Confronto tra padding e perturbazione sulla durata

Le due famiglie condividono il perimetro del *threat model* (modifica *offline*, solo traffico malevolo, un protocollo di trasporto per scenario) ma differiscono per l'oggetto della modifica (payload applicativo o timestamp), per i controlli applicati e per l'effetto atteso sulle grandezze di flusso (statistiche di volume anziché durata). La Tabella 3.1 riassume il confronto tra le due perturbazioni.

### 3.3.5 Impatto sul vettore di feature

Le perturbazioni non agiscono direttamente sul vettore di feature del classificatore: alterano la cattura *prima* dell'aggregazione in flussi e dell'estrazione delle variabili numeriche. Il padding

Tabella 3.1: Confronto tra padding del payload e perturbazione sulla durata.

| | **Padding del payload** | **Perturbazione sulla durata** |
|---|---|---|
| Cosa si modifica | Byte nel payload applicativo (TCP/UDP) | Timestamp in cattura dell'ultimo frame ammissibile del flusso |
| Controlli principali | Payload applicativo non vuoto; margine MTU; ricalcolo lunghezze e checksum | Flusso ammissibile; 3WHS riconoscibile (TCP); ultimo frame con sorgente malevola; ritardo parametrizzato |
| Frame invariati | Handshake/ACK senza dati; payload applicativo assente; MTU insufficiente; sorgente non malevola | Flusso non ammissibile; 3WHS assente (TCP); ultimo frame non inviato dall'host compromesso |
| Effetto atteso sul flusso | Aumento delle statistiche di volume del flusso (`TotBytes`, derivati) | Allungamento di `Dur`; effetto indiretto su feature per secondo |

sposta verso l'alto le statistiche di volume del flusso; la perturbazione sulla durata sposta verso l'alto l'intervallo temporale del flusso e modifica di conseguenza le statistiche normalizzate rispetto al tempo. Se lo spostamento indotto su `TotBytes`, `Dur` o feature correlate è sufficiente a far varcare la soglia decisionale del modello, il traffico malevolo perturbato può essere etichettato come benigno: si tratta dell'effetto di evasione che gli esperimenti del capitolo successivo misurano sui classificatori addestrati. Il dettaglio della pipeline di aggregazione, preprocessing e inferenza, inclusi i nomi delle feature e le configurazioni numeriche degli scenari, è descritto nel Capitolo 4.

## 3.4 Validazione concettuale e limiti

Le perturbazioni introdotte operano sulla cattura prima dell'aggregazione in flussi. Questa sezione riassume i criteri di accettazione (dettagliati in §3.3), delimita i limiti di un attacco operato *offline* e illustra come l'effetto si propaga fino alle grandezze di flusso su cui inferisce il classificatore.

### 3.4.1 Criteri di accettazione

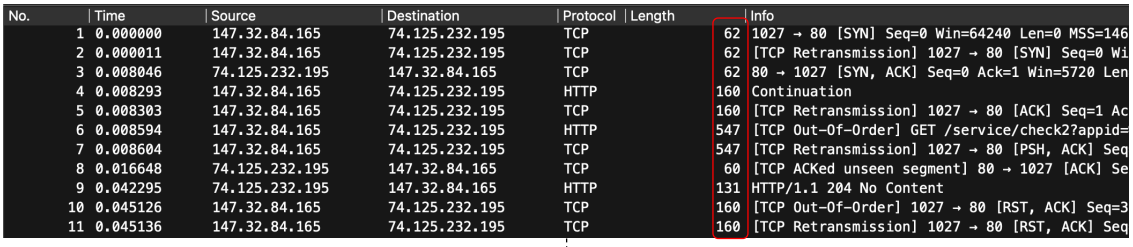
I criteri con cui un frame o un flusso perturbato è considerato ammissibile nel modello adottato sono sviluppati in §3.3.2 (padding) e in §3.3.3 (durata); qui se ne riassume lo schema. In entrambi i casi le perturbazioni riguardano solo il traffico malevolo, perché l'attaccante ha controllo esclusivo sul traffico in uscita dall'host compromesso e non può alterare comunicazioni di altri host né risposte del destinatario (§3.1.5); il resto dell'acquisizione resta invariato.

Per il *padding*, un frame è candidato se soddisfa i vincoli comuni di §3.3.1 e le regole di §3.3.2 (payload applicativo non vuoto, sorgente malevola, margine MTU sufficiente); altrimenti resta invariato. Se il padding è applicato, il datagramma deve essere ricostruito con lunghezze e checksum coerenti, come precisato in §3.3.2.

Per la *durata*, un flusso è ammissibile se soddisfa i vincoli di §3.3.3 (scenario TCP o UDP, 3WHS completo per TCP, ultimo frame del flusso inviato dall'host compromesso); su quel solo frame si incrementa il timestamp del ritardo configurato.

### 3.4.2 Esempi di propagazione verso il record di flusso

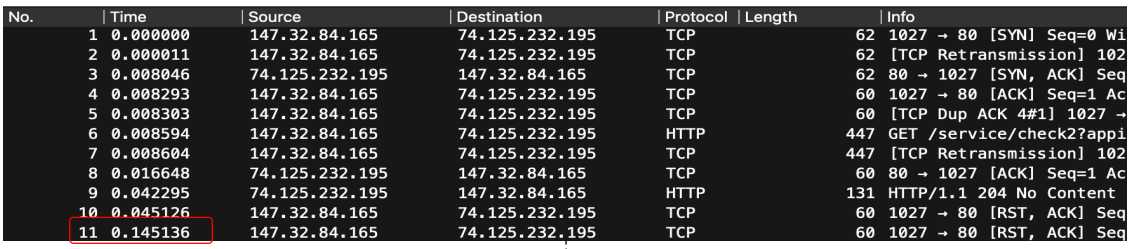
Oltre ai criteri sui singoli frame, il modello richiede che l'effetto delle perturbazioni sia tracciabile fino alle grandezze di flusso su cui il classificatore inferisce. La Figura 3.9 e la Figura 3.10 illustrano, per la stessa conversazione TCP di Figure 3.4 e 3.5, la cattura dopo padding e dopo perturbazione sulla durata, ciascuna con il record di flusso ottenuto per aggregazione. Rispetto al record non perturbato di Figura 3.5, il padding incrementa `TotBytes`; la perturbazione sulla durata allunga `Dur` a parità di byte trasportati.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000 | 147.32.84.165 | 74.125.232.195 | TCP | 62 | 1027 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=146 |
| 2 | 0.000011 | 147.32.84.165 | 74.125.232.195 | TCP | 62 | [TCP Retransmission] 1027 → 80 [SYN] Seq=0 Wi |
| 3 | 0.008046 | 74.125.232.195 | 147.32.84.165 | TCP | 62 | 80 → 1027 [SYN, ACK] Seq=0 Ack=1 Win=5720 Len |
| 4 | 0.008293 | 147.32.84.165 | 74.125.232.195 | HTTP | 160 | Continuation |
| 5 | 0.008303 | 147.32.84.165 | 74.125.232.195 | TCP | 160 | [TCP Retransmission] 1027 → 80 [ACK] Seq=1 Ac |
| 6 | 0.008594 | 147.32.84.165 | 74.125.232.195 | HTTP | 547 | [TCP Out-Of-Order] GET /service/check2?appid= |
| 7 | 0.008604 | 147.32.84.165 | 74.125.232.195 | TCP | 547 | [TCP Retransmission] 1027 → 80 [PSH, ACK] Seq |
| 8 | 0.016648 | 74.125.232.195 | 147.32.84.165 | TCP | 60 | [TCP ACKed unseen segment] 80 → 1027 [ACK] Se |
| 9 | 0.042295 | 74.125.232.195 | 147.32.84.165 | HTTP | 131 | HTTP/1.1 204 No Content |
| 10 | 0.045126 | 147.32.84.165 | 74.125.232.195 | TCP | 160 | [TCP Out-Of-Order] 1027 → 80 [RST, ACK] Seq=3 |
| 11 | 0.045136 | 147.32.84.165 | 74.125.232.195 | TCP | 160 | [TCP Retransmission] 1027 → 80 [RST, ACK] Seq |

| Dur | SrcAddr | Sport | DstAddr | Dport | TotPkts | TotBytes | Proto |
|---|---|---|---|---|---|---|---|
| **0.045136** | 147.32.84.165 | 1027 | 74.125.232.195 | 80 | 11 | 2111 | tcp |

Figura 3.9: Propagazione del padding (+100 bytes): undici frame della conversazione TCP di Figura 3.4 dopo perturbazione di volume (cattura in alto) e record di flusso risultante dopo aggregazione (in basso). Rispetto a Figura 3.5, `TotBytes` aumenta.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000 | 147.32.84.165 | 74.125.232.195 | TCP | 62 | 1027 → 80 [SYN] Seq=0 Wi |
| 2 | 0.000011 | 147.32.84.165 | 74.125.232.195 | TCP | 62 | [TCP Retransmission] 102 |
| 3 | 0.008046 | 74.125.232.195 | 147.32.84.165 | TCP | 62 | 80 → 1027 [SYN, ACK] Seq |
| 4 | 0.008293 | 147.32.84.165 | 74.125.232.195 | TCP | 60 | 1027 → 80 [ACK] Seq=1 Ac |
| 5 | 0.008303 | 147.32.84.165 | 74.125.232.195 | TCP | 60 | [TCP Dup ACK 4#1] 1027 → |
| 6 | 0.008594 | 147.32.84.165 | 74.125.232.195 | HTTP | 447 | GET /service/check2?appi |
| 7 | 0.008604 | 147.32.84.165 | 74.125.232.195 | TCP | 447 | [TCP Retransmission] 102 |
| 8 | 0.016648 | 74.125.232.195 | 147.32.84.165 | TCP | 60 | 80 → 1027 [ACK] Seq=1 Ac |
| 9 | 0.042295 | 74.125.232.195 | 147.32.84.165 | HTTP | 131 | HTTP/1.1 204 No Content |
| 10 | 0.045126 | 147.32.84.165 | 74.125.232.195 | TCP | 60 | 1027 → 80 [RST, ACK] Seq |
| 11 | 0.145136 | 147.32.84.165 | 74.125.232.195 | TCP | 60 | 1027 → 80 [RST, ACK] Seq |

| Dur | SrcAddr | Sport | DstAddr | Dport | TotPkts | TotBytes | Proto |
|---|---|---|---|---|---|---|---|
| **0.045136** | 147.32.84.165 | 1027 | 74.125.232.195 | 80 | 11 | 2111 | tcp |

Figura 3.10: Propagazione della perturbazione sulla durata (+100 ms): stessa conversazione di Figura 3.4 con timestamp modificati in cattura (in alto) e record di flusso risultante (in basso). Rispetto a Figura 3.5, `Dur` aumenta; `TotBytes` resta invariato.

# 4. Implementazione sperimentale

Il Capitolo 3 ha formalizzato il *threat model* e le perturbazioni in *traffic space* a livello concettuale: obiettivo di evasione, conoscenza *black box* del sistema target, modifica *offline* del traffico malevolo in una cattura PCAP e due famiglie di attacco valutate separatamente (padding del payload e ritardo nell'invio dei pacchetti). Questo capitolo descrive come tali scelte sono state tradotte in una pipeline sperimentale riproducibile, dalla cattura originale fino all'addestramento e alla valutazione dei classificatori.

A livello operativo, l'attaccante è modellato come un host compromesso che altera i propri pacchetti prima che vengano osservati dal NIDS. Nell'esperimento non si reinietta il traffico in rete: si parte da acquisizioni storiche del dataset CTU-13, si applicano le perturbazioni offline sui file PCAP, si riesegue l'estrazione dei flussi con gli stessi strumenti adottati per il traffico non perturbato e si valuta se i record così ottenuti inducono il classificatore a etichettare come benigni flussi originariamente malevoli. Il traffico benigno presente nella stessa acquisizione resta invariato, in linea con le capacità definite in §3.1.5.

Si parte dall'ambiente di setup (§4.1): hardware, dipendenze software e strumenti per la manipolazione del traffico e l'estrazione dei flussi. Si introduce poi il dataset CTU-13, le botnet considerate e i criteri con cui si seleziona il traffico malevolo da perturbare (§4.2). A valle delle catture PCAP, l'estrazione dei flussi mediante Argus produce i record su cui si applica il preprocessing (§4.3, §4.4). Ne derivano il vettore di feature, la suddivisione temporale dei dati, la distribuzione delle classi (§4.5) e infine l'addestramento dei classificatori di *machine learning* con i relativi iperparametri (§4.6). Le perturbazioni definite nel capitolo precedente trovano qui la loro traduzione operativa sui file PCAP (§4.7). Il capitolo si chiude con la definizione degli scenari sperimentali (§4.8). Metriche e risultati quantitativi sono oggetto del Capitolo 5.

## 4.1 Ambiente sperimentale e strumenti

### 4.1.1 Hardware e software

Le elaborazioni descritte in questo capitolo sono state eseguite in un ambiente di sviluppo locale con processore Apple M4 e 16 GB di memoria RAM e sistema operativo macOS. L'ambiente Python ospita i notebook che realizzano le fasi della pipeline sperimentale, dalla perturbazione del traffico alla valutazione dei classificatori. Le librerie installate includono *Scapy* ($\geq$ 2.4.3) per la manipolazione offline delle catture PCAP, *pandas* ($\geq$ 1.4.4) e *NumPy* ($\geq$ 1.21.5) per la gestione tabellare dei flussi e il preprocessing, *scikit-learn* ($\geq$ 1.0.2) per l'addestramento e la valutazione dei classificatori e *matplotlib* per la visualizzazione dei risultati.

### 4.1.2 Strumenti per la manipolazione del traffico e l'estrazione dei flussi

Oltre alle librerie Python, la pipeline si appoggia a componenti installati a livello di sistema per la conversione delle catture PCAP in record di flusso. `Argus` e il client `ra` leggono le acquisizioni e producono record di flusso in formato tabellare secondo parametri di configurazione dedicati all'estrazione dei flussi e alla formattazione dell'output. La manipolazione offline delle catture, necessaria per applicare le perturbazioni in *traffic space* formalizzate nel Capitolo 3, resta affidata a *Scapy*.

## 4.2 Dataset

### 4.2.1 Il dataset CTU-13

Il dataset CTU-13 [9] è una raccolta pubblica di catture di traffico reale, rilasciata dallo Stratosphere Laboratory della Czech Technical University, ottenuta monitorando una rete universitaria con macchine virtuali infette da famiglie di botnet affiancate a traffico benigno nella stessa acquisizione. È composto da tredici scenari, ciascuno corrispondente all'esecuzione di un diverso campione di malware in una finestra di osservazione, durante la quale host legittimi generano attività di routine. L'etichettatura è disponibile a livello di acquisizione: a ciascuna cattura è associata un'unica famiglia

di botnet. La regola con cui questa informazione viene tradotta in etichette binarie sui singoli flussi è definita in §4.2.4. Si è scelto CTU-13 perché ogni acquisizione contiene traffico benigno e malevolo nella stessa finestra temporale, in linea con il modello adottato in cui il NIDS osserva una rete condivisa, ed è già stato impiegato in valutazioni di ML-NIDS e in studi su perturbazioni *adversarial*, come il lavoro di partenza di questa tesi [2].

Per l'esperimento si utilizzano le catture PCAP fornite con il dataset, senza reiniezione in rete, e i flussi vengono riestratti con Argus (§4.3) anziché impiegare i record NetFlow già etichettati distribuiti con la raccolta, così da uniformare la pipeline di estrazione tra traffico originale e perturbato. Il traffico appartiene alla rete interna documentata come `147.32.0.0/16`, utilizzata anche in fase di preprocessing per distinguere indirizzi interni ed esterni. La Tabella 4.1 riassume le tracce malevole impiegate, riportando per ciascuna la data di acquisizione, il numero di pacchetti ottenuto con `capinfos` e il numero totale di flussi Argus estratti dall'intera cattura.

Tabella 4.1: Tracce malevole CTU-13 (file `botnet-capture`, 2011) usate nell'esperimento. Tra parentesi è riportato il corrispondente scenario 1-13 del dataset CTU-13 completo. I flussi totali sono i record Argus estratti dall'intera cattura.

| **Famiglia** | **Scenario** | **Data** | **Pacchetti** | **Flussi totali** |
|---|---|---|---|---|
| Neris | 42 (1) | 10 ago 2011 | 323 154 | 42 596 |
| Neris | 43 (2) | 11 ago 2011 | 176 064 | 23 140 |
| Rbot | 44 (3) | 12 ago 2011 | 495 056 | 39 063 |
| Rbot | 45 (4) | 15 ago 2011 | 256 712 | 884 |
| Virut | 46 (5) | 15 ago 2011 | 45 853 | 944 |
| Virut | 54 (13) | 15 ago 2011 | 440 625 | 41 424 |
| Menti | 47 (6) | 16 ago 2011 | 24 764 | 4 644 |
| Murlo | 49 (8) | 16 ago 2011 | 85 735 | 10 004 |
| Nsis | 53 (12) | 19 ago 2011 | 352 266 | 7 949 |

Da CTU-13 si considerano sei famiglie di botnet acquisite nel 2011: Neris, Rbot, Virut, Menti, Murlo e Nsis, con comportamenti di rete che spaziano da spam e click-fraud a scansione, attacchi DDoS e comunicazioni di *Command and Control* verso l'esterno. I flussi malevoli sono estratti dai

file `botnet-capture`, che contengono il solo traffico della macchina virtuale infetta, mentre i flussi benigni sono ricostruiti dalle catture complete troncate (`truncated`) dello stesso scenario.

Alcuni scenari del dataset CTU-13 completo non sono stati considerati per vincoli di dimensione delle catture e di tempo di elaborazione, pur restando disponibili nella raccolta originale. Le tracce incluse sono sufficienti a rappresentare varietà di malware e volumi di traffico comparabili a quelli usati in lavori correlati sugli stessi dati.

### 4.2.2 Il dataset MCFP

Per affiancare alle tracce del 2011 traffico malevolo più recente, si utilizzano quattro catture del 2017 distribuite dallo stesso laboratorio nell'ambito del Malware Capture Facility Project (MCFP) [15]. L'MCFP è il progetto continuativo dello Stratosphere Laboratory che acquisisce traffico malevolo, benigno e di background per la costruzione di dataset di sicurezza di rete, e di cui lo stesso CTU-13 costituisce un sottoinsieme curato. Le quattro famiglie introducono minacce più recenti e non tutte riconducibili a botnet in senso stretto: Trickbot e Dridex sono trojan bancari, Wannacry è un *ransomware* con propagazione di tipo *worm* via SMB e Artemis è un trojan con funzioni di *backdoor*.

I flussi malevoli provengono dalle rispettive catture dell'host infetto, documentato nella sottorete interna `192.168.1.0/24`. A differenza di CTU-13, in cui traffico benigno e malevolo convivono nella stessa acquisizione, per queste famiglie il traffico benigno è un pool condiviso ottenuto dalle acquisizioni `normal` 28-32, prive di host infetti. Queste acquisizioni benigne risalgono allo stesso periodo (2017) delle tracce malevole, così da preservare la coerenza temporale tra traffico benigno e malevolo, al costo di non disporre dei due tipi di traffico nella medesima cattura. Le Tabelle 4.2 e 4.3 riassumono rispettivamente il pool benigno condiviso e le tracce malevole recenti, con gli stessi campi descritti per la Tabella 4.1.

Tabella 4.2: Tracce benigne `normal` (2017) che compongono il pool benigno condiviso dalle famiglie MCFP recenti. Il numero di scenario rimanda alla cattura nel repository MCFP.

| **Scenario** | **Data** | **Pacchetti** | **Flussi totali** |
|---|---|---|---|
| 28 | 01 mag 2017 | 266 422 | 12 433 |
| 29 | 01 mag 2017 | 702 373 | 34 196 |
| 30 | 01 mag 2017 | 1 252 965 | 54 216 |
| 31 | 01 mag 2017 | 1 240 259 | 60 387 |
| 32 | 02 mag 2017 | 1 601 294 | 82 111 |

Tabella 4.3: Tracce malevole recenti del Malware Capture Facility Project (2017) per le quattro nuove famiglie. Il numero di scenario rimanda alla cattura nel repository MCFP.

| **Famiglia** | **Scenario** | **Data** | **Pacchetti** | **Flussi totali** |
|---|---|---|---|---|
| Trickbot | 244 | 12 apr 2017 | 429 302 | 55 129 |
| Trickbot | 247 | 17 apr 2017 | 895 172 | 104 135 |
| Trickbot | 261-1 | 15 mag 2017 | 705 397 | 81 719 |
| Trickbot | 261-3 | 08 mag 2017 | 1 233 817 | 127 856 |
| Dridex | 248 | 18 apr 2017 | 258 713 | 30 492 |
| Dridex | 249 | 18 apr 2017 | 270 832 | 61 354 |
| Dridex | 257 | 16 mag 2017 | 278 239 | 64 125 |
| Dridex | 259 | 15 mag 2017 | 100 969 | 42 774 |
| Dridex | 260 | 15 mag 2017 | 167 068 | 49 085 |
| Wannacry | 252 | 14 mag 2017 | 6 008 | 5 078 |
| Wannacry | 253 | 14 mag 2017 | 30 957 | 14 950 |
| Wannacry | 254 | 15 mag 2017 | 4 198 | 175 |
| Wannacry | 256 | 15 mag 2017 | 90 271 | 31 527 |
| Artemis | 275 | 24 giu 2017 | 163 924 | 23 077 |
| Artemis | 306 | 01 ago 2017 | 950 116 | 140 948 |

### 4.2.3 Selezione del traffico malevolo e IP infetti

Le perturbazioni in *traffic space* si applicano solo al traffico attribuibile agli host compromessi, in coerenza con §3.1.5. La documentazione di ogni cattura associa allo scenario gli indirizzi IP delle macchine infette, e l'implementazione adotta un insieme di IP infetti definito per scenario. Per gli scenari CTU-13 l'host compromesso è `147.32.84.165`, con l'aggiunta di `147.32.84.191` e `147.32.84.192` nella cattura di Nsis (scenario 53), mentre ciascuna cattura MCFP recente documenta un singolo host infetto nella sottorete interna `192.168.1.0/24`. Le acquisizioni `normal` 28-32 non contengono host infetti, quindi i loro flussi sono interamente benigni. Un frame è candidato alla perturbazione solo se la sorgente a livello IP appartiene all'insieme degli IP infetti dello scenario. I frame con sorgente diversa, registrati nelle stesse acquisizioni, restano invariati.

### 4.2.4 Etichettatura dei flussi

Per l'addestramento del classificatore si definisce un'etichetta binaria su ciascun record di flusso estratto con Argus (§4.3). Un flusso è considerato *malevolo* se almeno uno tra indirizzo sorgente e destinazione a livello IP appartiene all'insieme degli host infetti indicato in §4.2.3. Un flusso è considerato *benigno* se né la sorgente né la destinazione coinvolgono tali indirizzi. Il traffico benigno e malevolo coesiste nelle stesse acquisizioni e viene separato applicando questa regola ai record estratti, senza etichettatura manuale aggiuntiva. La perturbazione offline si limita ai frame con sorgente infetta, mentre l'etichetta malevola del flusso include anche i flussi in cui l'host compromesso compare come destinazione, in quanto riflettono comunicazioni che coinvolgono il botnet.

### 4.2.5 Organizzazione dei dati per botnet

Per ciascuna famiglia di botnet si aggregano i flussi malevoli e benigni ottenuti dalle acquisizioni PCAP delle Tabelle 4.1, 4.2 e 4.3, applicando la regola di §4.2.4. Si addestra un classificatore dedicato per famiglia, come descritto in §4.6. La selezione casuale dei flussi benigni, la suddivisione temporale dei dati e la distribuzione delle classi sono trattati in §4.5.

La stima dell'effetto delle perturbazioni richiede un numero sufficiente di flussi malevoli nel test set: con pochi campioni la detection rate diventa instabile e non interpretabile. Quindi si adotta

una soglia di significatività, escludendo dall'analisi le coppie (famiglia, protocollo) con meno di mille flussi malevoli di test. Su TCP sono escluse menti e nsis, mentre su UDP l'analisi si limita a neris, virut e nsis, poiché per le altre famiglie, comprese tutte le MCFP recenti, i flussi malevoli UDP sono troppo pochi.

La Tabella 4.4 (TCP) e la Tabella 4.5 (UDP) riportano, per le famiglie incluse, la numerosità del pool grezzo di flussi malevoli e benigni e la dimensione dei dataset di training e test con rapporto 10:1, come descritto in §4.5.2. La composizione mette in evidenza due situazioni opposte a seconda di quale classe sia il fattore limitante. Per le famiglie CTU-13 il fattore limitante è il traffico malevolo: i flussi benigni della cattura sono molto più abbondanti, quindi la selezione impiega tutti i flussi malevoli disponibili. Per le quattro famiglie MCFP recenti su TCP, che condividono lo stesso pool benigno `normal` 28-32, il fattore limitante diventa invece il traffico benigno: poiché il pool benigno condiviso è inferiore a dieci volte i flussi malevoli, la selezione riduce questi ultimi a un suo decimo. I quattro dataset TCP risultano di dimensione identica e gran parte del pool malevolo resta inutilizzata. Questa distinzione è ripresa nel Capitolo 5 per interpretare l'affidabilità delle metriche.

Tabella 4.4: Composizione dei dataset TCP per famiglia: pool grezzo di flussi malevoli e benigni (prima della selezione) e dimensione dei dataset di training e test con rapporto 10:1, espressi come benigni / malevoli.

| **Famiglia** | **Malevoli** | **Benigni** | **Train (ben / mal)** | **Test (ben / mal)** |
|---|---|---|---|---|
| Neris | 34 977 | 1 153 691 | 279 530 / 27 953 | 69 880 / 6 988 |
| Rbot | 34 076 | 2 056 028 | 271 520 / 27 152 | 67 878 / 6 787 |
| Virut | 34 014 | 569 468 | 271 112 / 27 111 | 67 776 / 6 777 |
| Murlo | 8 537 | 157 254 | 68 218 / 6 821 | 17 050 / 1 705 |
| Trickbot | 258 860 | 92 160 | 73 720 / 7 372 | 18 430 / 1 843 |
| Dridex | 60 782 | 92 160 | 73 720 / 7 372 | 18 430 / 1 843 |
| Wannacry | 50 415 | 92 160 | 73 720 / 7 372 | 18 430 / 1 843 |
| Artemis | 43 815 | 92 160 | 73 720 / 7 372 | 18 430 / 1 843 |

Tabella 4.5: Composizione dei dataset UDP per famiglia: pool grezzo di flussi malevoli e benigni (prima della selezione) e dimensione dei dataset di training e test con rapporto 10:1, espressi come benigni / malevoli.

| Famiglia | Malevoli | Benigni | Train (ben / mal) | Test (ben / mal) |
|---|---|---|---|---|
| Neris | 30 317 | 5 413 627 | 242 470 / 24 247 | 60 620 / 6 062 |
| Virut | 7 789 | 2 141 354 | 62 180 / 6 218 | 15 550 / 1 555 |
| Nsis | 7 348 | 302 344 | 58 770 / 5 877 | 14 690 / 1 469 |

La versione perturbata dei dati è ottenuta dagli stessi scenari di quella *clean*: le perturbazioni modificano i frame nelle catture PCAP del traffico malevolo e i flussi vengono poi riestratti con la stessa procedura. In questo modo il confronto tra i due scenari isola l'effetto delle perturbazioni, a parità di famiglia di malware e di flussi di partenza.

## 4.3 Estrazione dei flussi

Il classificatore non opera sui singoli pacchetti, ma su record di flusso che sintetizzano una comunicazione tra due host in un dato intervallo. Il passaggio dalle catture PCAP a tali record è affidato ad Argus, che aggrega i pacchetti in flussi, e al client `ra`, che esporta i flussi in formato tabellare. Questa rappresentazione orientata al flusso è coerente con la prospettiva del NIDS adottata nel Capitolo 3 ed è quella su cui si innesta il preprocessing descritto in §4.4.

### 4.3.1 Configurazione di Argus

Argus viene configurato per produrre flussi *bidirezionali* con chiave a cinque campi, ovvero indirizzi IP e porte di sorgente e destinazione insieme al protocollo di trasporto. La modalità bidirezionale riunisce nello stesso record le due direzioni della comunicazione, distinguendo le quantità relative all'host sorgente da quelle relative al destinatario. La configurazione abilita inoltre l'esportazione di metriche estese utili alla caratterizzazione del traffico: conteggi di pacchetti e byte per direzione, statistiche sulla dimensione dei pacchetti, indicatori temporali sull'andamento del flusso e metriche specifiche delle connessioni TCP. Per i flussi di lunga durata, lo stato viene riportato a intervalli

regolari, in modo che una comunicazione prolungata sia rappresentata da più record successivi anziché da un'unica osservazione aggregata. La medesima configurazione è applicata in maniera identica alle catture originali e a quelle perturbate, così che le due versioni differiscano solo per le perturbazioni e non per i parametri di estrazione.

### 4.3.2 Esportazione tabellare dei flussi

Per ciascun PCAP, originale o perturbato, la conversione segue la stessa sequenza a due passi:

Codice 4.1: Conversione da PCAP a CSV con Argus e `ra`

```
argus -F argus.conf -r input.pcap -w flows.argus
ra -r flows.argus -F ra.conf -n -c ',' > output.csv
```

Il primo passo aggrega i pacchetti in flussi secondo la configurazione di §4.3.1, mentre il secondo seleziona i campi esportati e produce il CSV di input al preprocessing. Ogni record riporta l'istante e la durata del flusso, gli indirizzi e le porte di sorgente e destinazione, il protocollo, i conteggi di pacchetti e byte per direzione e lo stato della connessione, oltre alle metriche estese abilitate in fase di configurazione. L'estrazione è indipendente dal protocollo di trasporto e ciascun record ne conserva l'indicazione: la separazione tra traffico TCP e UDP, trattata come due insiemi distinti, avviene in fase di preprocessing (§4.4). I file così ottenuti costituiscono l'input grezzo per la costruzione del vettore di feature e per le fasi successive della pipeline.

## 4.4 Preprocessing e costruzione del vettore di feature

I record prodotti da Argus non sono direttamente utilizzabili da un classificatore: contengono campi non numerici, indirizzi e porte grezzi oppure valori mancanti o non validi. Il preprocessing trasforma questi record in un vettore di feature numerico e uniforme, eliminando le informazioni che indurrebbero il modello a riconoscere gli host piuttosto che il comportamento del traffico. L'impostazione adottata si ispira allo schema di *feature engineering* per la rilevazione di botnet proposto con il dataset DReLAB [16], da cui riprende la categorizzazione di porte e indirizzi e l'arricchimento con feature derivate.

### 4.4.1 Trasformazioni applicate

Il preprocessing procede per passi successivi. Si seleziona innanzitutto il traffico del protocollo di trasporto considerato, trattando TCP e UDP come insiemi distinti. Le porte di sorgente e destinazione vengono sostituite da indicatori binari, uno per ciascuna classe (*well-known*, *registered* o *private*), anziché dal valore numerico puntuale. Le porte *well-known* (da 0 a 1023) sono associate ai servizi standard, quelle *registered* (da 1024 a 49151) agli applicativi registrati e quelle *private* (da 49152 in poi) sono assegnate dinamicamente alle connessioni effimere. Raggruppare le porte in queste classi conserva l'informazione sul tipo di servizio coinvolto senza legare il modello a numeri di porta specifici. Analogamente, ogni indirizzo IP è ridotto a un indicatore binario che segnala se sorgente e destinazione appartengono alla rete interna monitorata. Il prefisso interno non è unico sull'intero esperimento, ma dipende dalla cattura di provenienza (§4.2.1, §4.2.2): per le tracce CTU-13 si adotta `147.32.0.0/16`, per le catture malevole MCFP del 2017 `192.168.1.0/24`, per il pool benigno condiviso `normal` 28–32 le reti documentate con quelle acquisizioni (`10.0.2.0/24` e `192.168.33.0/24`). In ogni caso si conserva l'informazione interno o esterno senza esporre al modello gli indirizzi puntuali. Questa scelta, ripresa da DReLAB [16], evita che il classificatore distingua il traffico malevolo sulla base di specifici indirizzi o porte, fenomeno che comprometterebbe la capacità di generalizzazione. Le variabili categoriche rimaste, come la direzione e lo stato della connessione, vengono codificate in forma binaria. I valori mancanti o non validi, prodotti ad esempio dai rapporti con denominatore nullo, vengono ricondotti a valori ammissibili.

Combinando byte, pacchetti e durata del flusso si calcolano alcune feature derivate che sintetizzano l'intensità e la forma del traffico: il numero di byte per pacchetto, i byte e i pacchetti al secondo e il rapporto tra byte uscenti ed entranti. Come osservato in [16], l'aggiunta di queste grandezze derivate migliora la capacità discriminante dei classificatori. La Tabella 4.6 riassume per categoria le feature che compongono il vettore finale.

Tabella 4.6: Categorie di feature che compongono il vettore in ingresso al classificatore. Le feature di base provengono dall'estrazione con Argus, mentre quelle derivate e categoriche sono prodotte in fase di preprocessing.

| **Categoria** | **Features** | **Descrizione** |
|---|---|---|
| Temporali | `Dur`, `SrcDur`, `DstDur` | durata totale e per direzione |
| Volume | `TotBytes`, `SrcBytes`, `DstBytes` | byte totali e per direzione |
| Pacchetti | `TotPkts`, `SrcPkts`, `DstPkts` | pacchetti totali e per direzione |
| Header e QoS | `sTos`, `dTos`, `sTtl`, `dTtl`, `sHops` | tipo di servizio, TTL, hop |
| Stato e direzione | `State`, `Dir` | stato della connessione, direzione |
| Tipo di porta | `SrcPort*`, `DstPort*` | classe *well-known/registered/private* |
| Tipo di indirizzo | `IPSrcType`, `IPDstType` | interno o esterno alla rete monitorata |
| Derivate | `BytesPerPkt`, `BytesPerSec`, `PktsPerSec`, `RatioOutIn` | intensità e forma del traffico |

La codifica binaria delle variabili categoriche espande il numero di colonne effettive, quindi il vettore finale di features raggiunge un centinaio di componenti, in larga parte indicatori binari di stato e direzione. Da questo insieme sono escluse le informazioni puramente identificative, come l'istante iniziale del flusso, che non rappresentano comportamenti del traffico. La Figura 4.1 riassume l'intero flusso di trasformazione, dal record di flusso grezzo estratto con Argus al vettore di feature in ingresso al classificatore.

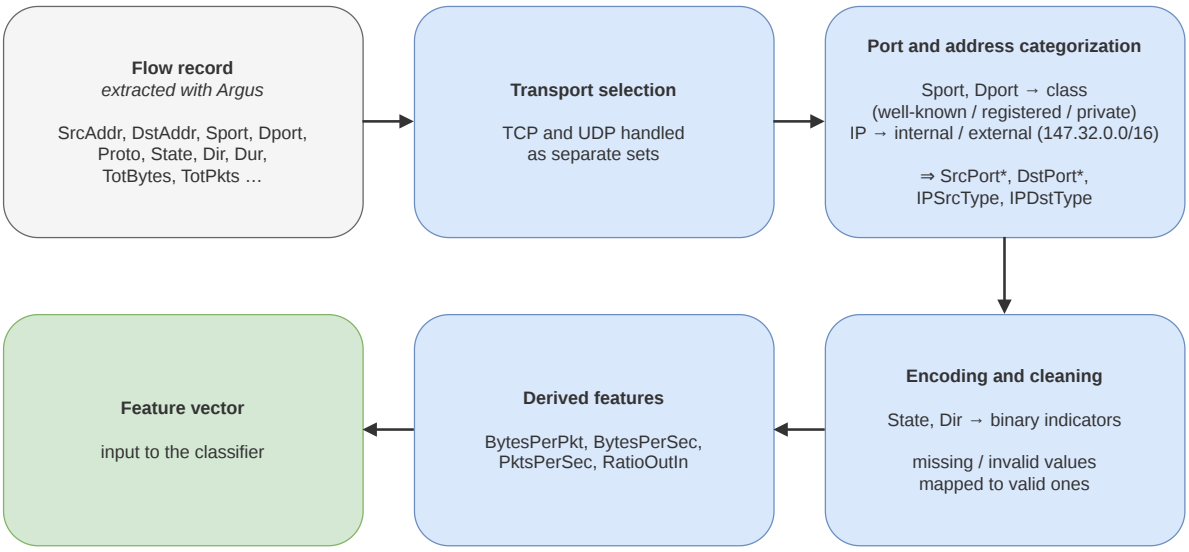



Figura 4.1: Dal record di flusso al vettore di feature: categorizzazione di porte e indirizzi, codifica delle variabili categoriche e derivazione delle feature di traffico.

### 4.4.2 Adattamenti rispetto a DReLAB

Il preprocessing adottato segue lo schema di DReLAB [16], con alcune differenze che si descrivono di seguito per chiarezza e riproducibilità. DReLAB considera il solo traffico TCP, mentre in questo lavoro TCP e UDP sono analizzati separatamente, così da valutare le perturbazioni anche sul traffico privo di handshake. La selezione delle famiglie di botnet è più ampia e include NSIS, che in DReLAB era stata esclusa per scarsità di campioni. Inoltre non si applica il filtraggio degli *outlier* basato su soglie percentili previsto in DReLAB: si è preferito preservare l'intera coda delle distribuzioni, poiché proprio i valori estremi di durata e volume sono quelli toccati dalle perturbazioni. Infine, i valori mancanti vengono ricondotti a valori validi anziché portare allo scarto del record, per non ridurre ulteriormente il già limitato numero di flussi malevoli di alcune famiglie. La codifica dello stato della connessione è inoltre più granulare, con indicatori distinti per sorgente e destinazione.

## 4.5 Suddivisione e composizione dei dati

I flussi pre-processati vengono suddivisi in un dataset di training e uno di test. Questa fase definisce come i dati sono divisi nel tempo, come viene fissato il rapporto tra le classi e come gli scenari *clean* e perturbato sono resi confrontabili. Per ciascuna famiglia di botnet si costruiscono tre tipi di dataset:

- **training**: 80% temporale dei flussi benigni e malevoli non perturbati, in rapporto di dieci flussi benigni per ogni flusso malevolo;
- **test *clean***: 20% temporale dei flussi benigni e malevoli non perturbati, con lo stesso rapporto;
- **test perturbato**: gli stessi flussi del test *clean*, con i soli flussi malevoli sostituiti dalla loro versione perturbata, uno per ogni tipo e livello di perturbazione.

Le sottosezioni seguenti dettagliano la suddivisione temporale, la distribuzione delle classi e la costruzione dei test perturbati. La Figura 4.2 illustra la suddivisione temporale e la derivazione dei tre dataset.

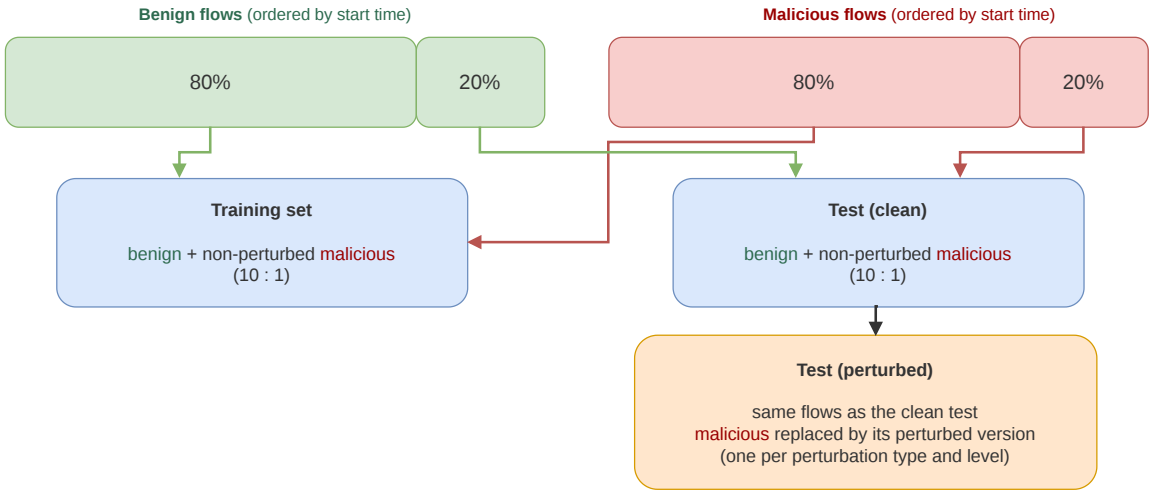



Figura 4.2: Suddivisione temporale 80/20 e costruzione dei dataset di test *clean* e perturbato a partire dagli stessi flussi.

### 4.5.1 Suddivisione temporale

La separazione tra addestramento e test segue l'ordine temporale dei flussi anziché una partizione casuale. Per ciascuna famiglia di botnet i flussi vengono ordinati in base all'istante iniziale e suddivisi in due parti: l'80% iniziale costituisce il dataset di training e il restante 20% il dataset di test. La suddivisione è applicata in modo indipendente ai flussi benigni e a quelli malevoli, così da preservare le proporzioni di entrambe le classi nei due insiemi. Una divisione temporale, anziché casuale, riproduce la condizione realistica in cui il classificatore è addestrato su traffico passato e valutato su traffico successivo, evitando che informazioni del futuro influenzino l'addestramento.

### 4.5.2 Distribuzione delle classi

Nelle acquisizioni il traffico benigno domina di gran lunga quello malevolo: nelle catture CTU-13 convivono milioni di flussi legittimi e poche migliaia di flussi attribuibili alla botnet, in linea con un NIDS operativo in cui gli eventi di attacco sono rari rispetto all'attività di routine. Per training e test non si utilizza quindi l'intero pool grezzo, ma si selezionano casualmente i flussi benigni fino al rapporto di 10:1 definito sopra. Tale scelta attenua lo squilibrio estremo dei dati pur lasciando prevalere il traffico legittimo, così che il classificatore non sia addestrato su un insieme artificialmente equilibrato e distante da uno scenario reale. Il rapporto dieci a uno è inoltre un compromesso pratico: per alcune famiglie i flussi malevoli disponibili sono già pochi, quindi un rapporto più sfavorevole ai benigni ridurrebbe ulteriormente la classe di interesse e la dimensione effettiva dei dataset (Tabelle 4.4 e 4.5).

### 4.5.3 Allineamento tra scenario clean e perturbato

Oltre al test *clean*, per ogni tipo di perturbazione e per ogni livello si costruisce un dataset di test perturbato. Le perturbazioni vengono applicate alle catture PCAP malevole della famiglia, dalle quali si riestraggono i flussi come descritto in §4.3. Da questi si selezionano le versioni perturbate degli stessi flussi malevoli presenti nel test *clean*, abbinandole tramite una chiave che identifica univocamente ciascun flusso a partire da indirizzi, porte e ordine temporale all'interno della comunicazione. I flussi benigni non vengono toccati e restano identici a quelli del test *clean*. Alcuni flussi malevoli possono coincidere con la loro versione originale, poiché le regole di

perturbazione definite nel Capitolo 3 non modificano tutti i frame. La differenza di prestazioni tra il test *clean* e un test perturbato è quindi attribuibile unicamente alle perturbazioni applicate, e non a variazioni nella composizione del dataset.

## 4.6 Classificatori

Per ciascuna famiglia di botnet si addestra un classificatore binario distinto, che separa i flussi malevoli da quelli benigni a partire dal vettore di feature descritto in §4.4. La scelta di un modello per famiglia riflette il fatto che ogni botnet presenta un comportamento di rete caratteristico, che un classificatore dedicato può cogliere meglio di un unico modello generico. Sono stati adottati tre modelli basati su alberi di decisione, una famiglia di algoritmi efficace sul traffico di rete in forma tabellare e poco sensibile alla scala delle singole feature, che non richiede quindi una normalizzazione preliminare [6].

### 4.6.1 Modelli adottati

La Tabella 4.7 riassume gli iperparametri principali dei tre modelli. Tutti ricevono `random_state` dal seme della singola run (§4.6.2) e non applicano ponderazione interna delle classi, dato che il rapporto tra le classi è fissato a monte tramite selezione casuale dei flussi in §4.5. Gli iperparametri non riportati mantengono i valori predefiniti della libreria.

Tabella 4.7: Iperparametri principali dei tre classificatori. I valori non riportati corrispondono ai predefiniti della libreria.

| **Modello** | **Iperparametri principali** |
|---|---|
| Decision Tree | `criterion=gini`, `splitter=best`, `max_depth=None`, `max_features=None` |
| Random Forest | `n_estimators=200`, `criterion=gini`, `max_features=sqrt`, `bootstrap=True` |
| HGB | `loss=log_loss`, `learning_rate=0.1`, `max_iter=100`, `max_leaf_nodes=31`, `min_samples_leaf=20`, `early_stopping=auto` |

I tre modelli rappresentano un compromesso crescente tra semplicità e capacità di rappresentazione. Il *Decision Tree* è un singolo albero di decisione, il riferimento più semplice e immediatamente interpretabile, ma soggetto a sovradattamento. La *Random Forest* è un insieme di alberi, ciascuno addestrato su un campione *bootstrap* del training e su sottoinsiemi casuali di feature, le cui predizioni vengono combinate per voto di maggioranza in modo da ridurre la varianza del singolo albero (*bagging*). L'*HistGradientBoosting* (HGB) costruisce invece gli alberi in sequenza, ognuno dei quali corregge gli errori dei precedenti (*boosting*). La sua variante a istogrammi discretizza i valori delle feature in un numero ridotto di intervalli, riducendo i tempi di addestramento su dataset con molte feature.

### 4.6.2 Addestramento e riproducibilità

L'addestramento e la valutazione sono replicati in modo indipendente per ogni famiglia di botnet: a partire dai flussi di quella famiglia si costruiscono i tre insiemi descritti in §4.5 (training, test *clean* e test perturbato). Sul training, con flussi benigni e malevoli non perturbati in rapporto di 10:1, si addestrano i tre classificatori; sul test *clean*, costruito con la stessa proporzione sul 20% temporale successivo, e sui test perturbati, in cui i flussi benigni restano invariati e i soli flussi malevoli sono sostituiti dalle rispettive versioni perturbate (§4.5.3), si misurano le prestazioni. Il rapporto tra le classi è imposto a monte tramite selezione casuale dei flussi, quindi i classificatori non applicano ponderazione interna delle classi. Gli esperimenti in *traffic space* sono ripetuti dieci volte, variando il seme di casualità da 0 a 9 per ogni combinazione di protocollo di trasporto e tipo di perturbazione. In ciascuna ripetizione il seme determina tutte le sorgenti stocastiche della pipeline: la selezione casuale dei flussi benigni nella composizione dei dataset e la costruzione degli alberi (scelta casuale delle feature nella Random Forest, inizializzazione nell'HGB). A parità di seme, l'intera procedura è quindi riproducibile. La scelta di ripetere l'addestramento e la valutazione risponde alla natura probabilistica di alcuni modelli e al rischio che una singola partizione casuale o un singolo addestramento produca risultati non rappresentativi. Apruzzese et al. [3], nel quadro della *pragmatic assessment*, raccomandano prove multiple per ottenere stime più affidabili: nel loro studio le ripetizioni arrivano a centinaia di trial, mentre qui se ne adottano dieci per limitare il costo computazionale, riportando nel Capitolo 5 la media sulle ripetizioni e, nelle figure dello scenario perturbato, l'intervallo di confidenza al 95%. Al termine di ogni addestramento si esportano, per

famiglia e modello, stime dell'importanza delle feature: riduzione di impurità (MDI) per il *Decision Tree* e la *Random Forest*, importanza per permutazione per l'HGB, calcolata su un sotto-campione stratificato del training.

## 4.7 Perturbazioni adversarial

Questa sezione descrive l'implementazione concreta delle due strategie di perturbazione definite a livello concettuale nel Capitolo 3. Entrambe agiscono *offline* sulle catture PCAP, limitatamente al traffico malevolo, ovvero il traffico sotto il controllo dell'attaccante e producono una cattura perturbata distinta dall'originale, dalla quale i flussi vengono nuovamente estratti come descritto in §4.3. Le due strategie sono valutate separatamente e non vengono mai combinate sulla stessa cattura. In entrambi i casi l'obiettivo è produrre perturbazioni conformi al protocollo e realizzabili da un host compromesso, così che l'eventuale evasione derivi da modifiche plausibili del traffico e non da manipolazioni arbitrarie della cattura.

### 4.7.1 Strumenti e flusso di lavoro

Le perturbazioni sono realizzate con la libreria Scapy, che consente di leggere una cattura PCAP frame per frame, modificarne i campi e riscriverla preservando l'ordine e i timestamp. La modifica interessa unicamente i frame la cui sorgente appartiene all'elenco di indirizzi infetti documentato per CTU-13, in coerenza con il *threat model* di un attaccante che controlla un solo host compromesso. Per ciascuna famiglia e per ciascun livello di perturbazione si genera una cattura perturbata indipendente, così da poter valutare separatamente l'effetto di ogni configurazione.

### 4.7.2 Padding del payload

Il padding incrementa il *payload applicativo* dei pacchetti, con la stessa logica per TCP e UDP. Sono candidati al padding solo i frame che trasportano già un payload applicativo non vuoto, escludendo quindi i pacchetti di puro controllo come quelli dell'handshake TCP. Si interviene sul solo payload applicativo perché un host compromesso può modificare i dati che genera, ma non gli header del traffico altrui. I pacchetti di puro controllo, privi di dati applicativi, non offrono un

punto di intervento realistico. L'incremento è vincolato dall'MTU: la lunghezza del datagramma IP dopo l'aggiunta non può superare i 1500 byte, quindi un frame il cui margine residuo non basta a contenere l'incremento richiesto resta invariato. Restare entro l'MTU garantisce che i frame siano consegnabili in rete senza frammentazione, come richiesto dai vincoli di realizzabilità discussi in §3.2.3. Poiché ogni configurazione applica un incremento fisso, un frame che non può riceverlo per intero entro l'MTU viene lasciato invariato anziché perturbato con meno byte, così da mantenere incrementi confrontabili tra frame e configurazioni. Quando il padding è ammesso, al payload vengono accodati byte casuali e il pacchetto viene ricostruito, ricalcolando lunghezze e checksum di rete e di trasporto in modo che il frame resti conforme al protocollo. Il contenuto dei byte aggiunti non incide sulle feature di flusso, che dipendono dai conteggi di byte e non dal loro valore, quindi l'effetto sul vettore di feature è univocamente determinato dalla configurazione. L'Algoritmo 1 riassume il procedimento.

**Algoritmo 1** Padding del payload applicativo (trasporto $T \in \{\text{TCP}, \text{UDP}\}$).

**Input:** cattura $C$, incremento fisso *pad*, IP infetti $M$, *MTU*

**Output:** cattura perturbata $C'$

$C' \leftarrow \langle\,\rangle$
**for** ogni frame $f \in C$ **do**
  **if** $f$ è privo del livello $T$ o di Ethernet **then**
    aggiungi $f$ a $C'$ **e passa al frame successivo**
  **else if** src$(f) \notin M$ **oppure** il payload applicativo di $f$ è vuoto **then**
    aggiungi $f$ a $C'$ **e passa al frame successivo**
  **else**
    *ip_tot* $\leftarrow$ lunghezza del datagramma IP di $f$
    **if** *ip_tot* + *pad* > *MTU* **then**
      aggiungi $f$ a $C'$ **e passa al frame successivo** ▷ margine MTU insufficiente
    **else**
      payload$(f) \leftarrow$ payload$(f)$ || byteCasuali(*pad*)
      invalida lunghezze e checksum di IP e $T$
      $f \leftarrow$ ricostruisci$(f)$ ▷ Scapy ricalcola lunghezze e checksum
      preserva il timestamp di $f$
      aggiungi $f$ a $C'$
    **end if**
  **end if**
**end for**
**return** $C'$

Ogni configurazione fissa l'incremento *pad* a un numero costante di byte per frame, pari a 1, 2, 5, 10, 50, 200, 500 e 1000, in modo da osservare l'effetto di incrementi di volume via via più marcati. L'incremento minimo è di un byte perché il padding agisce sul payload applicativo, le cui lunghezze sono espresse in byte a livello IP e trasporto: un singolo bit non produrrebbe una modifica coerente del datagramma né un effetto misurabile sulle statistiche di volume del flusso. In implementazione, all'incremento configurato corrisponde l'accodamento di un numero intero

di byte casuali. I livelli elencati non costituiscono una scelta ottimale a priori, ma una griglia di intensità crescente allineata ai valori di riferimento usati nella letteratura su evasione con padding sul traffico [1, 2], estesa verso incrementi più marcati per osservare l'effetto sulle feature di volume.

### 4.7.3 Perturbazione sulla durata

La perturbazione sulla durata non modifica byte, header o checksum, ma posticipa di un ritardo $\delta$ i timestamp di registrazione di frame selezionati nel PCAP, allungando di conseguenza la durata calcolata in fase di aggregazione. L'Algoritmo 2 descrive il procedimento.

**Algoritmo 2** Perturbazione sulla durata con ritardo $\delta$.

**Input:** cattura $C$, IP infetti $M$, trasporto $T$, ritardo $\delta$

raggruppa i frame di $C$ per quintupla simmetrica (chiave canonica)
**if** $T$ = TCP **then**
    **for** ogni connessione (3WHS completo e chiusura RST o FIN a quattro vie) di ogni gruppo per quintupla **do**
        $c \leftarrow$ ultimo frame della chiusura
        **if** src$(c) \in M$ **then** ritarda $c$ di $\delta$
        **end if**
    **end for**
**else if** $T$ = UDP **then**
    **for** ogni gruppo di frame con la stessa quintupla **do**
        $w \leftarrow$ ultimo frame del gruppo nel PCAP
        **if** src$(w) \in M$ **then** ritarda $w$ di $\delta$
        **end if**
    **end for**
**end if**
**return** $C$ con i timestamp aggiornati

La perturbazione agisce sul PCAP: per ciascun frame selezionato si aggiorna solo il timestamp di registrazione, senza alterare payload, header o checksum. Per individuare i frame da ritardare, l'implementazione raggruppa preventivamente i frame del PCAP per quintupla simmetrica: proto-

collo di trasporto e coppia ordinata dei due estremi (indirizzo IP e porta), indipendente dal verso del traffico. Questa operazione serve solo alla selezione dei frame e non coincide ancora con l'aggregazione in flussi eseguita da Argus a valle.

La scelta del frame dipende dal protocollo di trasporto. Su UDP, per ciascun gruppo di frame con la stessa quintupla, si ritarda l'ultimo frame presente nel PCAP. Su TCP, lo stesso raggruppamento per quintupla può includere più connessioni distinte. Si isolano quelle con handshake a tre vie completo e chiusura visibile (RST o sequenza FIN a quattro vie) e si ritarda l'ultimo frame della chiusura di ciascuna connessione. In entrambi i casi il ritardo è applicato solo se il frame selezionato proviene da un host infetto, coerentemente con un attaccante che modifica soltanto il proprio traffico.

Posticipare un frame in cattura allunga, dopo l'aggregazione, l'intervallo temporale del flusso corrispondente. Per Argus, `Dur` è la differenza tra il timestamp dell'ultimo e quello del primo pacchetto del record, perciò spostare in avanti l'istante finale incrementa la durata quando quel frame coincide con l'estremo temporale del flusso, come illustrato in Figura 3.8 del Capitolo 3. Quando più connessioni TCP con la stessa quintupla confluiscono in un unico record di flusso, gli estremi sono il primo pacchetto della prima connessione e l'ultimo pacchetto dell'ultima. Ritardare la chiusura di ogni connessione copre sia il caso in cui Argus emette un record per connessione sia quello in cui le fonde in un unico flusso, perché in quest'ultimo caso è comunque la chiusura dell'ultima connessione a fissare l'istante finale.

Dal punto di vista dell'attaccante la scelta non richiede conoscenza della pipeline del NIDS. Allungare una conversazione sul proprio traffico si riflette, dopo l'aggregazione, sulla durata del flusso osservato, indipendentemente dall'aggregatore specifico, in linea con l'ipotesi *black box*.

I ritardi sono espressi in millisecondi perché modellano pause brevi e plausibili tra invii consecutivi dall'host compromesso: ritardi dell'ordine dei secondi, se applicati in rete, aumenterebbero il rischio di timeout di sessione o di chiusura anticipata da parte della controparte e degli intermediari, oltre a allontanarsi da ritardi tipicamente introdotti da un client prima della chiusura di una connessione. I livelli 1, 5, 10, 50, 100, 250 e 500 ms, con un estremo a 1 s, coprono spostamenti temporali via via più ampi pur restando prevalentemente nella fascia sub-secondo, in linea con i piccoli ritardi considerati in studi precedenti su evasione tramite alterazioni temporali del traffico [1].

### 4.7.4 Verifica di correttezza

Poiché le perturbazioni sono applicate *offline*, una cattura potrebbe contenere frame incoerenti senza che alcun componente di rete li rifiuti. Per questo le catture perturbate vengono verificate prima di essere usate, controllando il rispetto dei vincoli di realizzabilità discussi in §3.2.3. La verifica si basa su Wireshark e sul suo strumento da riga di comando, abilitando il controllo dei checksum di IP, TCP e UDP: sui frame perturbati nessun checksum risulta errato, a conferma che la ricostruzione del pacchetto ha prodotto datagrammi conformi al protocollo. Si controlla inoltre che il numero di frame della cattura perturbata coincida con quello dell'originale, poiché nessuna delle due strategie rimuove o aggiunge pacchetti. Per la perturbazione sulla durata si verifica infine che payload, header e checksum restino identici all'originale e che cambino soltanto i timestamp dei frame selezionati.

### 4.7.5 Perturbazioni in feature space

Accanto alle perturbazioni in *traffic space* descritte sopra, si introduce una variante in *feature space* come riferimento di confronto. La distinzione tra i due spazi di attacco è discussa nel Capitolo 2 (§2.4.2): le perturbazioni in *feature space* agiscono direttamente sulle variabili numeriche già estratte dal flusso, senza passare dalla cattura PCAP. Questa variante non rappresenta un attacco realizzabile, ma riproduce sui dati di questo lavoro l'impostazione adottata da gran parte della letteratura, così da quantificare nel Capitolo 5 quanto la valutazione in *feature space* si discosti da quella in *traffic space*.

Le due perturbazioni replicano il padding e il ritardo sulla durata con gli stessi livelli di intensità (§4.8.2), ma applicati riga per riga ai soli flussi malevoli, senza intervenire sui pacchetti. Un flusso è perturbabile se la sua sorgente appartiene agli IP infetti dello scenario e il protocollo di trasporto è quello considerato, coerentemente con il *threat model* di §3.1.5. Il padding incrementa i byte lato sorgente e totali di una quantità pari all'incremento per pacchetto moltiplicato per il numero di pacchetti sorgente del flusso, mentre la perturbazione sulla durata aggiunge il ritardo a `Dur` e alla durata lato sorgente; in entrambi i casi le feature derivate (§4.4.1) vengono ricalcolate per coerenza. A differenza del *traffic space*, questa variante non modella alcun vincolo fisico: in particolare il padding non è soggetto al limite imposto dall'MTU e può quindi gonfiare il volume senza il tetto

di §4.7.2, mentre restano invariati il conteggio dei pacchetti e le statistiche sulla dimensione dei singoli pacchetti, che nel *traffic space* possono invece cambiare per effetto della riestrazione.

Il test perturbato in *feature space* riusa esattamente gli stessi flussi del test *clean* (§4.5.3), sostituendo i soli flussi malevoli con la loro versione perturbata.

## 4.8 Scenari di valutazione

Gli scenari sperimentali combinano i classificatori addestrati, le due strategie di perturbazione e i due protocolli di trasporto, e definiscono come gli esiti *clean* e perturbato vengono confrontati. Le metriche e i risultati numerici sono riportati nel Capitolo 5.

### 4.8.1 Protocollo di valutazione

Ogni classificatore viene addestrato esclusivamente sul dataset di training *clean* descritto in §4.5 e non osserva mai traffico perturbato durante l'addestramento. La valutazione avviene poi in due fasi: prima sul test *clean*, che misura le prestazioni di riferimento del classificatore, e poi su ciascun test perturbato. Poiché i due test condividono gli stessi flussi e differiscono solo per le perturbazioni applicate al traffico malevolo, come descritto in §4.5.3, il confronto isola l'effetto dell'attacco sulle prestazioni. In questo modo si riproduce la condizione realistica di un NIDS già in produzione, che incontra il traffico manipolato per la prima volta al momento dell'attacco. Contromisure di *hardening* arricchiscono l'addestramento con campioni avversari, anche sintetici [17]. La valutazione sperimentale adotta invece un protocollo senza *adversarial training*: includere esempi perturbati avrebbe permesso al modello di apprendere la perturbazione, attenuando in modo artificiale il calo di prestazioni sul test perturbato e falsando così la misura dell'efficacia dell'attacco, coerentemente con l'ipotesi di attaccante *blind* del Capitolo 3.

### 4.8.2 Matrice degli scenari

Uno scenario è individuato da una famiglia di botnet, un protocollo di trasporto, un tipo di perturbazione e un livello di intensità. I protocolli TCP e UDP sono trattati separatamente e, per ciascuno,

si valutano sia il padding sia la perturbazione sulla durata. La Tabella 4.8 riassume le combinazioni considerate.

Tabella 4.8: Scenari di valutazione: combinazioni di tipo di perturbazione e protocollo di trasporto, con le feature principalmente alterate e i livelli di intensità.

| **Perturbazione** | **Trasporto** | **Feature alterate** | **Livelli** |
|---|---|---|---|
| Padding | TCP, UDP | `TotBytes` e derivate (`BytesPerPkt`, `BytesPerSec`, `RatioOutIn`) | 1, 2, 5, 10, 50, 200, 500, 1000 byte |
| Durata | TCP, UDP | `Dur` e derivate (`BytesPerSec`, `PktsPerSec`) | 1, 5, 10, 50, 100, 250, 500 ms e 1 s |

Le due strategie agiscono su gruppi di feature diversi: il padding incrementa i byte trasportati e quindi `TotBytes` e le grandezze da esso derivate, mentre la perturbazione sulla durata allunga `Dur` e le grandezze temporali correlate. L'effetto di ciascuna configurazione sulle feature di flusso e sulle prestazioni di rilevazione è analizzato nel Capitolo 5.

# 5. Valutazione sperimentale e risultati

Questo capitolo presenta la valutazione sperimentale degli scenari definiti nel Capitolo 4. Si riportano le prestazioni di rilevazione negli scenari *clean* e perturbato e, in chiusura, una discussione che sintetizza l'effetto sulle feature di flusso e il confronto con la valutazione in *feature space*.

## 5.1 Metriche di valutazione

La rilevazione è formulata come un problema di classificazione binaria in cui la classe positiva corrisponde al traffico malevolo e quella negativa al traffico benigno. Dal confronto tra le predizioni del classificatore e le etichette di riferimento (§4.2.4) si contano i veri positivi (TP), i veri negativi (TN), i falsi positivi (FP) e i falsi negativi (FN). Da queste quantità derivano le tre metriche adottate sulla classe malevola: la *precision* $TP/(TP + FP)$, la *recall* $TP/(TP + FN)$ e il punteggio $F_1$, media armonica di precision e recall. Si riportano le metriche sulla classe malevola perché è quella di interesse per la rilevazione e perché il dataset è sbilanciato verso il traffico benigno (§4.5), condizione in cui l'accuratezza complessiva risulterebbe poco informativa.

Nello scenario *clean* si valutano precision, recall e $F_1$ per caratterizzare le prestazioni di riferimento del detector sul traffico non perturbato. Nello scenario perturbato l'indicatore principale è la recall sulla classe malevola, qui indicata come *detection rate*, ovvero la frazione di flussi malevoli ancora riconosciuti come tali dopo la perturbazione. La scelta è motivata dal modello di minaccia: l'attaccante perturba soltanto il traffico malevolo (§3.1.5), quindi un'evasione efficace si manifesta come flussi malevoli classificati benigni, cioè come un calo della detection rate, mentre precision e recall sul traffico benigno restano sostanzialmente invariate. Tutte le metriche sono mediate sulle dieci ripetizioni con semi diversi (§4.6.2). Nelle figure dello scenario perturbato la curva riporta la media e la banda indica l'intervallo di confidenza al 95% sulle ripetizioni.

## 5.2 Risultati nello scenario clean

Le Tabelle 5.1 e 5.2 riportano precision, recall (detection rate, §5.1) e $F_1$ sulla classe malevola nello scenario *clean*, per ciascuna famiglia di botnet inclusa nell'analisi (§4.2.5) e per ciascun classificatore (media su dieci ripetizioni). La recall costituisce il riferimento da cui partono le curve dello scenario perturbato, dove corrisponde al livello di perturbazione nullo. In quasi tutte le combinazioni la detection rate e la precision sono elevate (in genere superiore a 0,95 per la precision) e il punteggio $F_1$ resta prossimo all'unità: sul traffico non perturbato i detector uniscono pochi falsi positivi a pochi falsi negativi, con prestazioni in linea con quelle attese da un ML-NIDS e compatibili con un impiego operativo.

Tabella 5.1: Precision, recall (detection rate) e $F_1$ sulla classe malevola nello scenario *clean* su traffico TCP, per famiglia e classificatore (media su dieci ripetizioni).

| | **RF** | | | **DT** | | | **HGB** | | |
|---|---|---|---|---|---|---|---|---|---|
| **Famiglia** | P | R | $F_1$ | P | R | $F_1$ | P | R | $F_1$ |
| neris | 0.983 | 0.864 | 0.920 | 0.954 | 0.882 | 0.917 | 0.983 | 0.849 | 0.911 |
| rbot | 1.000 | 0.988 | 0.994 | 0.989 | 0.993 | 0.991 | 0.998 | 0.992 | 0.995 |
| virut | 0.989 | 0.985 | 0.987 | 0.932 | 0.987 | 0.959 | 0.958 | 0.986 | 0.972 |
| murlo | 1.000 | 0.999 | 0.999 | 0.989 | 0.998 | 0.993 | 0.998 | 0.999 | 0.998 |
| trickbot | 0.999 | 0.992 | 0.995 | 0.993 | 0.991 | 0.992 | 0.998 | 0.992 | 0.995 |
| dridex | 0.999 | 1.000 | 1.000 | 0.996 | 1.000 | 0.998 | 0.998 | 1.000 | 0.999 |
| wannacry | 1.000 | 0.995 | 0.997 | 0.997 | 0.993 | 0.995 | 0.999 | 0.993 | 0.996 |
| artemis | 1.000 | 0.971 | 0.985 | 0.996 | 0.968 | 0.981 | 0.998 | 0.976 | 0.987 |

Tabella 5.2: Precision, recall (detection rate) e $F_1$ sulla classe malevola nello scenario *clean* su traffico UDP, per famiglia e classificatore (media su dieci ripetizioni).

| | **RF** | | | **DT** | | | **HGB** | | |
|---|---|---|---|---|---|---|---|---|---|
| **Famiglia** | P | R | $F_1$ | P | R | $F_1$ | P | R | $F_1$ |
| neris | 1.000 | 0.862 | 0.925 | 0.998 | 0.868 | 0.929 | 0.999 | 0.864 | 0.926 |
| virut | 0.998 | 0.994 | 0.996 | 0.995 | 0.995 | 0.995 | 0.999 | 0.995 | 0.997 |
| nsis | 0.987 | 0.970 | 0.979 | 0.968 | 0.966 | 0.967 | 0.980 | 0.977 | 0.978 |

## 5.3 Risultati nello scenario perturbato

Per ogni combinazione di protocollo e tipo di perturbazione si riporta la detection rate sulla classe malevola al crescere del livello di perturbazione, con una curva per ciascun classificatore e una banda che indica l'intervallo di confidenza al 95% sulle dieci ripetizioni. Le figure includono solo le famiglie con almeno mille flussi malevoli di test per protocollo (§4.2.5): su TCP *neris*, *rbot*, *virut*, *murlo*, *trickbot*, *dridex*, *wannacry* e *artemis*, mentre su UDP *neris*, *virut* e *nsis*. Il valore al livello zero coincide con lo scenario *clean* e funge da riferimento, così che il confronto isoli l'effetto della perturbazione (§4.8.1). I valori numerici completi a ogni livello sono riportati, distinti per tipo di perturbazione, nelle Tabelle A.1 e A.2 dell'Appendice A.

### 5.3.1 Padding del payload

Su TCP (Figura 5.1) il padding riduce la detection rate in modo progressivo con la quantità di byte aggiunti. L'effetto è trascurabile fino a poche decine di byte e diventa apprezzabile per incrementi maggiori, coerentemente con l'incremento percentuale di `TotBytes` riportato in §5.4.1. La sensibilità dipende dalla famiglia: *rbot* e *virut* restano sopra 0,9 anche al livello massimo e *wannacry* sopra 0,92, *neris* e *murlo* calano in misura moderata e *dridex* solo lievemente, mentre *artemis* e *trickbot* sono le più sensibili e scendono al livello massimo fino a circa 0,40 e 0,53. Su

TCP i tre modelli si comportano in modo abbastanza simile tra loro, con scostamenti più marcati solo per le famiglie più sensibili.

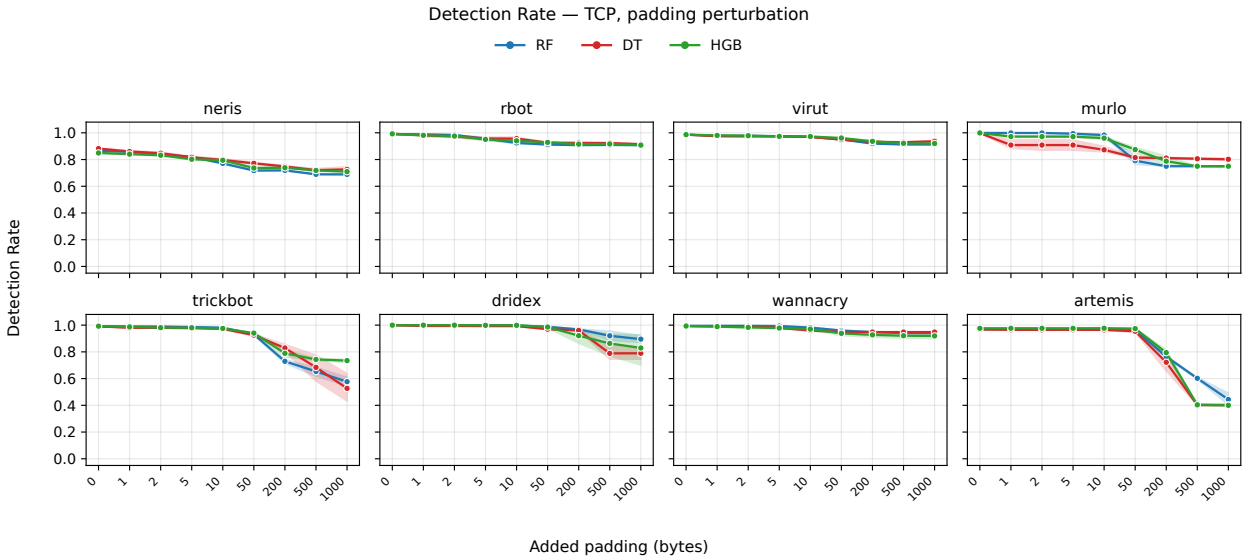



Figura 5.1: Detection rate sulla classe malevola al crescere del padding del payload, traffico TCP. Ogni subplot corrisponde a una famiglia; una curva per classificatore (media su dieci ripetizioni, banda: intervallo di confidenza al 95%).

Su UDP (Figura 5.2) il padding è nettamente più efficace e l'esito dipende fortemente dal classificatore. Per *neris* la detection rate di Random Forest e HGB crolla quasi a zero al livello massimo, mentre il Decision Tree si mantiene su valori più alti. Per *virut* si osserva il fenomeno opposto, con Random Forest che precipita a circa 0,25 e gli altri due modelli che restano intorno a 0,86. Su *nsis*, infine, la degradazione resta più contenuta, coerentemente con l'incremento percentuale di `TotBytes` riportato in §5.4.1. La divergenza tra modelli riflette il fatto che ciascun classificatore costruisce la decisione su sottoinsiemi di feature parzialmente diversi: poiché il padding altera `TotBytes` e le feature di volume da esso derivate, a cedere è il modello che vi attribuisce più peso. Gli intervalli di confidenza al 95% tengono conto della variabilità tra le dieci ripetizioni con semi diversi (§4.6.2): Random Forest e HGB non sono deterministici e, a parità di dataset, piccole differenze nell'addestramento si traducono in scostamenti visibili sulla recall malevola.

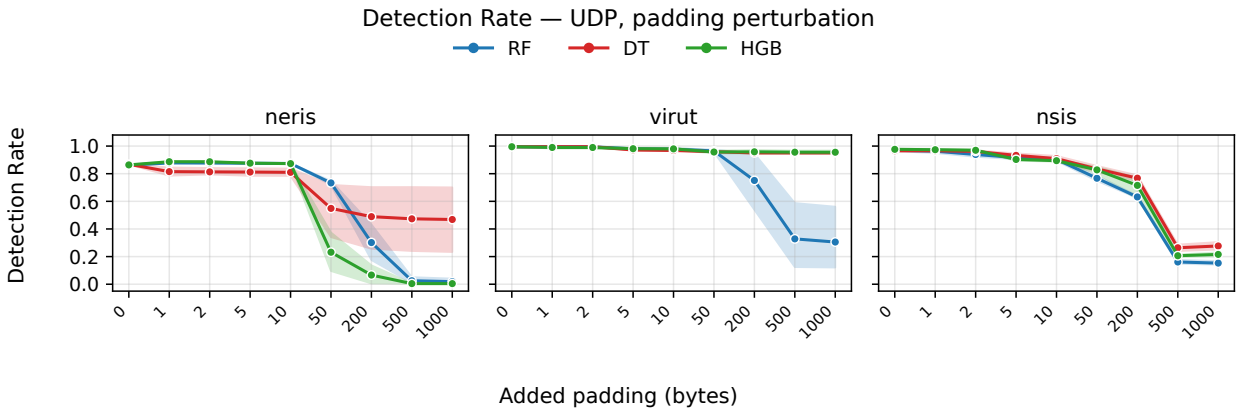



Figura 5.2: Detection rate sulla classe malevola al crescere del padding del payload, traffico UDP. Ogni subplot corrisponde a una famiglia; una curva per classificatore (media su dieci ripetizioni, banda: intervallo di confidenza al 95%).

### 5.3.2 Perturbazione sulla durata

Su TCP (Figura 5.3) la detection rate resta sostanzialmente costante a tutti i livelli per quasi tutte le famiglie. L'unica eccezione è *artemis*, la cui detection rate cala ai ritardi più alti (500 ms–1 s) fino a circa 0,73–0,86 a seconda del classificatore. Per le famiglie a detection rate piatta questo risultato non indica una reale robustezza del detector, ma una robustezza solo apparente. La perturbazione ritarda l'ultimo pacchetto inviato dall'attaccante per chiudere la connessione; la feature `Dur`, però, viene ricalcolata a valle dall'aggregatore come intervallo fra il primo e l'ultimo pacchetto del flusso. Il ritardo allunga quindi `Dur` soltanto quando il pacchetto spostato è davvero l'ultimo del flusso così come ricostruito in fase di aggregazione: se il flusso si chiude con un pacchetto successivo, ad esempio una risposta della controparte, l'intervallo misurato resta invariato. La detection rate piatta su TCP non riflette quindi una reale capacità del modello di resistere all'attacco, ma il fatto che la perturbazione sulla durata non modifica le feature. Il caso di *artemis* mostra il rovescio della medaglia: pur con uno spostamento di `Dur` modesto rispetto ad altre famiglie MCFP che restano comunque piatti (si veda §5.4.1), è l'unica famiglia su TCP in cui il ritardo si traduce in un calo della detection rate, segno che la vulnerabilità dipende da quanto ciascun modello pesa la feature `Dur` per quella famiglia, più che dall'entità assoluta dello spostamento.

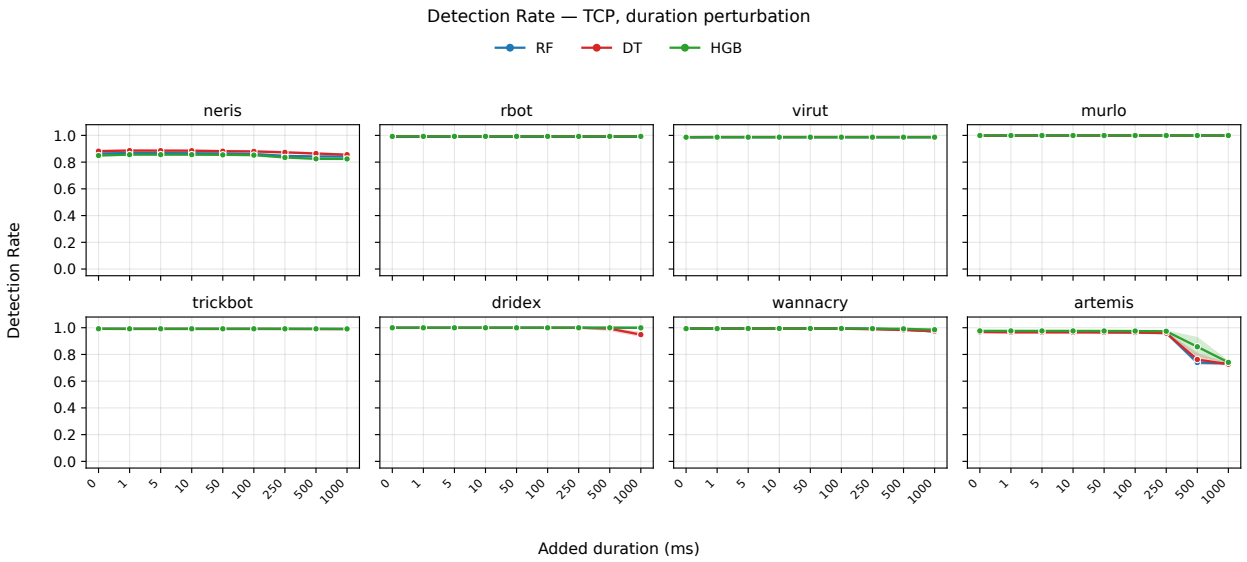



Figura 5.3: Detection rate sulla classe malevola al crescere del ritardo applicato, traffico TCP. Ogni subplot corrisponde a una famiglia; una curva per classificatore (media su dieci ripetizioni, banda: intervallo di confidenza al 95%).

Anche su UDP (Figura 5.4) la detection rate rimane stabile al crescere del ritardo e le poche oscillazioni visibili non seguono un andamento riconoscibile. Vale lo stesso meccanismo descritto per il TCP: ritardare l'ultimo pacchetto allunga `Dur` solo di poco e in pochi flussi, una variazione troppo piccola per spostare in modo apprezzabile la decisione del classificatore. L'asimmetria fra protocolli e dataset dipende soprattutto da quanto spesso il ritardo colpisce davvero l'ultimo pacchetto del flusso. Su TCP le famiglie CTU-13 incluse nelle figure (*neris*, *rbot*, *virut* e *murlo*) restano pressoché immutate al crescere del ritardo. Le famiglie MCFP mostrano spostamenti più visibili di `Dur`, ma la detection rate resta comunque piatta per quasi tutte. Su UDP *neris* e *virut* si comportano allo stesso modo. *nsis* è invece l'eccezione più rilevante, con un allungamento di `Dur` nettamente superiore (si veda §5.4.1). Nella quasi totalità dei casi, però, lo spostamento è troppo piccolo rispetto alla dispersione naturale di `Dur` per modificare la detection rate, che quindi rimane piatta non per robustezza del modello ma per debolezza dell'attacco a livello di feature. Fa eccezione *artemis* su TCP, dove il calo della detection rate ai ritardi maggiori indica che per quella famiglia la modifica di `Dur` è sufficiente a spostare la decisione del classificatore.

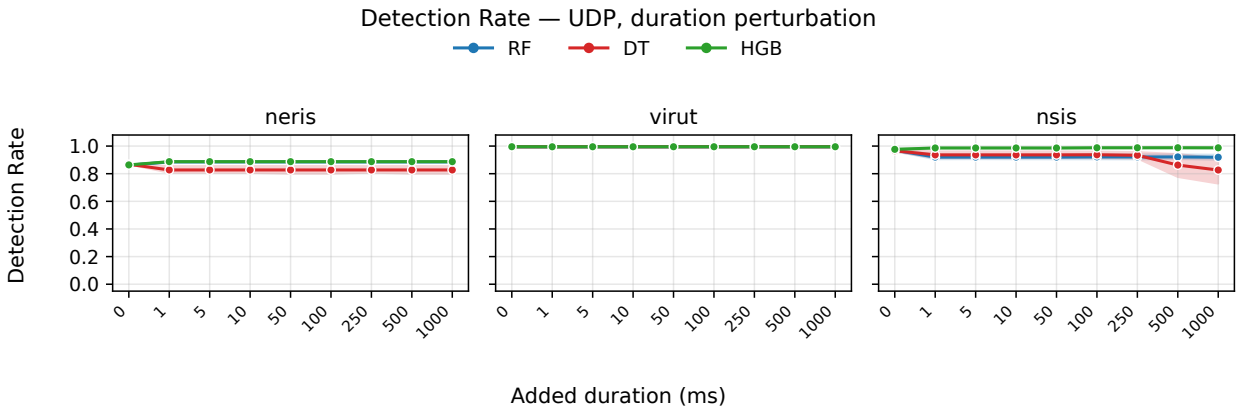



Figura 5.4: Detection rate sulla classe malevola al crescere del ritardo applicato, traffico UDP. Ogni subplot corrisponde a una famiglia; una curva per classificatore (media su dieci ripetizioni, banda: intervallo di confidenza al 95%).

## 5.4 Discussione

I risultati delineano un quadro coerente. Il padding del payload è la strategia di perturbazione più efficace, in particolare sul traffico UDP, mentre la perturbazione sulla durata incide poco o nulla sulla rilevazione. La detection rate piatta osservata su TCP con la perturbazione sulla durata è quindi una robustezza apparente, attribuibile all'attacco che non altera l'ingresso del classificatore, e non a una resilienza intrinseca del modello. L'unica eccezione rilevante è *artemis* su TCP, dove il ritardo riesce a spostare `Dur` abbastanza da ridurre la detection rate ai livelli più alti (§5.3.2).

### 5.4.1 Effetto delle perturbazioni sulle feature

Per verificare che ogni perturbazione modifichi le grandezze attese sui flussi malevoli, si calcola la media della feature bersaglio nello scenario *clean* e, a ogni livello di intensità, l'incremento percentuale rispetto a tale baseline. Le Tabelle 5.3–5.6 riportano i risultati per le famiglie incluse in ciascun protocollo (§4.2.5), calcolati sui soli flussi malevoli. Per il padding la feature osservata è `TotBytes`, per la durata è `Dur` (in secondi nella colonna *Clean*).

Il padding incrementa `TotBytes` in modo monotono e, al livello massimo, produce incrementi percentuali dell'ordine delle decine o delle centinaia di punti percentuali su TCP e UDP. La

perturbazione sulla durata, invece, sposta `Dur` in misura trascurabile su TCP per quasi tutte le famiglie (spesso inferiore all'1% anche al ritardo di 1 s). Su UDP *neris* e *virut* seguono lo stesso andamento, mentre solo *nsis* registra uno spostamento più marcato (fino a circa +6% a 1 s). Questa asimmetria anticipa i risultati di rilevamento di §5.3.1 e §5.3.2.

Tabella 5.3: Media di `TotBytes` sullo scenario *clean* e incremento percentuale al crescere del padding del payload (traffico TCP, soli flussi malevoli).

| **Famiglia** | **Clean** | +1 | +2 | +5 | +10 | +50 | +200 | +500 | +1000 |
|---|---|---|---|---|---|---|---|---|---|
| neris | 2,196 | +0,2% | +0,4% | +1,0% | +2,1% | +10,4% | +41,6% | +103,8% | +205,6% |
| rbot | 3,496 | +0,1% | +0,3% | +0,7% | +1,4% | +6,8% | +27,3% | +65,7% | +127,0% |
| virut | 3,941 | +0,1% | +0,2% | +0,4% | +0,8% | +4,2% | +16,6% | +41,4% | +80,0% |
| murlo | 1,835 | +0,2% | +0,4% | +1,0% | +2,3% | +10,2% | +40,8% | +101,0% | +196,5% |
| trickbot | 2,468 | +0,1% | +0,2% | +0,4% | +0,9% | +4,5% | +17,8% | +44,6% | +89,2% |
| dridex | 2,897 | +0,1% | +0,1% | +0,4% | +0,7% | +3,6% | +14,4% | +36,0% | +55,2% |
| wannacry | 541 | +0,1% | +0,2% | +0,4% | +0,8% | +2,9% | +9,7% | +23,7% | +46,3% |
| artemis | 7,629 | +0,1% | +0,1% | +0,3% | +0,6% | +2,8% | +11,3% | +28,2% | +56,4% |

Tabella 5.4: Media di `TotBytes` sullo scenario *clean* e incremento percentuale al crescere del padding del payload (traffico UDP, soli flussi malevoli).

| **Famiglia** | **Clean** | +1 | +2 | +5 | +10 | +50 | +200 | +500 | +1000 |
|---|---|---|---|---|---|---|---|---|---|
| neris | 316 | +0,6% | +1,3% | +3,2% | +6,4% | +31,9% | +127,5% | +318,8% | +637,6% |
| virut | 445 | +0,5% | +0,9% | +2,3% | +4,6% | +22,8% | +91,3% | +228,3% | +456,7% |
| nsis | 33,280 | +0,1% | +0,2% | +0,5% | +1,0% | +4,9% | +19,5% | +40,8% | +49,3% |

Tabella 5.5: Media di `Dur` (s) sullo scenario *clean* e incremento percentuale al crescere del ritardo applicato (traffico TCP, soli flussi malevoli). Le colonne indicano il ritardo in millisecondi; gli incrementi inferiori a 0,1% sono indicati come <0,1%.

| **Famiglia** | **Clean** | +1 | +5 | +10 | +50 | +100 | +250 | +500 | +1 s |
|---|---|---|---|---|---|---|---|---|---|
| neris | 10,8 | <0,1% | <0,1% | <0,1% | <0,1% | +0,1% | +0,1% | +0,2% | +0,5% |
| rbot | 7,97 | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% |
| virut | 5,20 | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% |
| murlo | 8,09 | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% |
| trickbot | 22,6 | <0,1% | <0,1% | <0,1% | +0,1% | +0,2% | +0,4% | +0,9% | +1,7% |
| dridex | 1,39 | <0,1% | +0,1% | +0,2% | +0,8% | +1,6% | +3,9% | +7,9% | +15,8% |
| wannacry | 0,28 | <0,1% | +0,1% | +0,3% | +1,3% | +2,2% | +6,4% | +12,0% | +24,9% |
| artemis | 0,66 | <0,1% | +0,1% | +0,1% | +0,6% | +1,2% | +3,0% | +6,1% | +11,8% |

Tabella 5.6: Media di `Dur` (s) sullo scenario *clean* e incremento percentuale al crescere del ritardo applicato (traffico UDP, soli flussi malevoli). Le colonne indicano il ritardo in millisecondi; gli incrementi inferiori a 0,1% sono indicati come <0,1%.

| **Famiglia** | **Clean** | +1 | +5 | +10 | +50 | +100 | +250 | +500 | +1 s |
|---|---|---|---|---|---|---|---|---|---|
| neris | 1,72 | <0,1% | <0,1% | <0,1% | <0,1% | <0,1% | +0,1% | +0,1% | +0,3% |
| virut | 0,23 | <0,1% | <0,1% | <0,1% | <0,1% | +0,1% | +0,2% | +0,3% | +0,7% |
| nsis | 9,65 | <0,1% | <0,1% | +0,1% | +0,3% | +0,4% | +1,3% | +2,8% | +5,8% |

### 5.4.2 Confronto con le perturbazioni in feature space

I risultati di rilevamento presentati in §5.3 misurano l'evasione in *traffic space*, con perturbazioni introdotte a livello di pacchetto e feature riestratte a valle. Per quantificare quanto questo approccio si differenzi da quelli adottati da gran parte della letteratura, le stesse perturbazioni sono state replicate in *feature space*, applicandole direttamente ai valori delle feature dei flussi malevoli

secondo il riferimento di confronto definito in §4.7.5. La domanda è se la valutazione in *feature space*, spesso assunta in letteratura come stima ottimistica dell'evasione, rappresenti davvero lo scenario peggiore per il detector, ossia se l'evasione ottenibile in *traffic space* resti sempre al di sotto di quella misurata sulle feature.

Il confronto si concentra sulla detection rate per il classificatore Random Forest, rappresentando in ciascuna figura le due curve (*traffic space* e *feature space*) al crescere dell'intensità di perturbazione. La detection rate *clean* è praticamente identica nei due spazi perché modello e dataset di partenza coincidono. A parità di baseline, la differenza fra i due spazi è quindi riconducibile soltanto a come la perturbazione raggiunge le feature. I valori numerici completi in *feature space* sono riportati nelle Tabelle B.1 e B.2 dell'Appendice B, quelli in *traffic space* nelle Tabelle A.1 e A.2 dell'Appendice A.

Per TCP la valutazione sulle feature fa apparire sistematicamente l'evasione più forte di quella reale nel padding del payload: il calo di detection rate al livello massimo è molto più marcato che in *traffic space* (ad esempio *virut* scende a 0,013 contro 0,912, *rbot* a 0,077 contro 0,907). Il motivo è strutturale: in *feature space* il padding non è soggetto al limite dell'MTU (§4.7.2) e può gonfiare `TotBytes` senza alcun tetto, spostando i flussi in una regione lontanissima da quella appresa dal classificatore. In *traffic space*, invece, lo stesso incremento è vincolato dalla capienza dei pacchetti e resta molto più contenuto. La Figura 5.5 riassume il confronto su TCP: la curva in *feature space* crolla rapidamente, mentre quella in *traffic space* cala in modo più graduale.

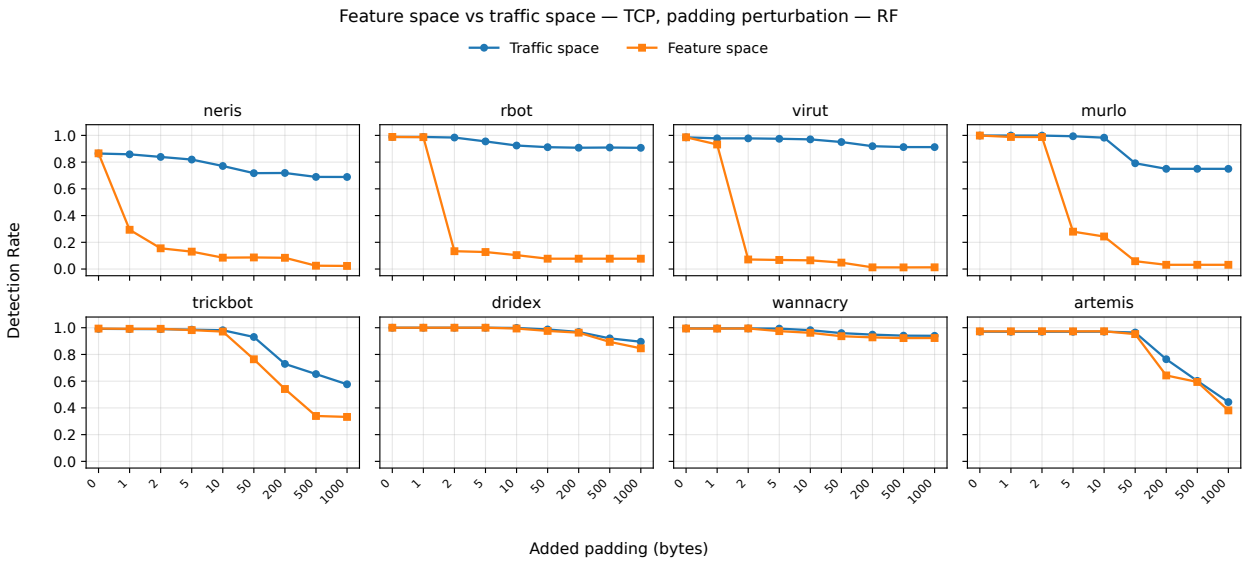



Figura 5.5: Detection rate (Random Forest) al crescere del padding del payload: confronto fra *traffic space* e *feature space*, traffico TCP.

Per UDP il comportamento è diverso. A differenza del TCP, i pacchetti malevoli su UDP sono piccoli e lasciano ampio margine rispetto all'MTU: il padding realizzabile in *traffic space* riesce quindi ad aggiungere quasi tutti i byte previsti, così che il volume iniettato diventa comparabile a quello del *feature space* illimitato (§4.7.2). Di conseguenza, su UDP la valutazione sulle feature non fa più apparire l'evasione più forte di quella reale, come accade invece su TCP: per alcune famiglie l'attacco realizzabile in *traffic space* riduce la detection rate quanto quello in *feature space*, o persino di più. Su UDP, però, il confronto vale solo come indicazione, perché i risultati in *feature space* si basano su una singola ripetizione. La Figura 5.6 riporta lo stesso confronto su UDP.

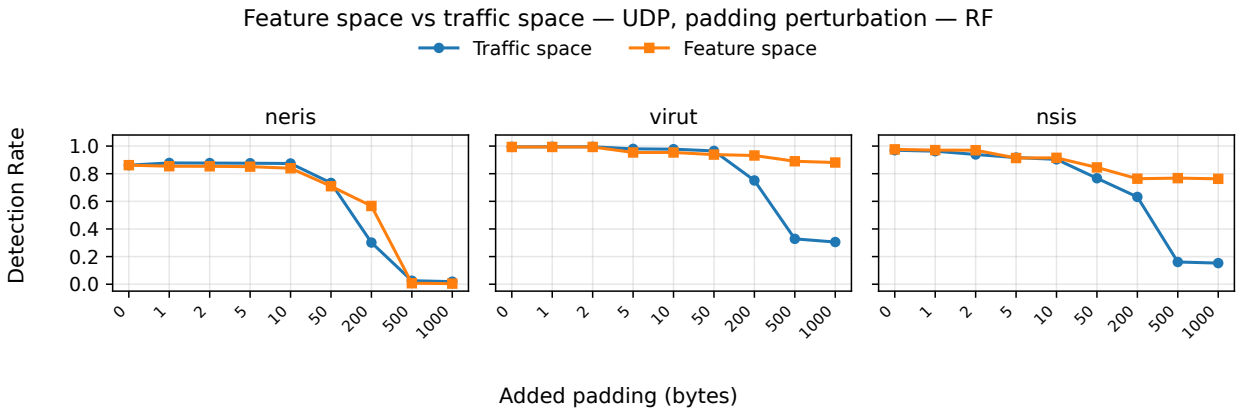



Figura 5.6: Detection rate (Random Forest) al crescere del padding del payload: confronto fra *traffic space* e *feature space*, traffico UDP.

Lo stesso schema vale, con sfumature diverse, per la perturbazione sulla durata in TCP. In *feature space* il ritardo viene sommato direttamente a `Dur`, ma l'effetto non è uniforme su tutte le famiglie. Su *murlo* la detection rate di Random Forest scende a 0,26 a 1 s mentre in *traffic space* resta sostanzialmente invariata, perché lì `Dur` quasi non si sposta (§5.3.2). Su *artemis* il calo è marcato in entrambi gli spazi (0,10 in *feature space* contro 0,73 in *traffic space*), ma resta più forte quando la feature viene alterata direttamente. Su *rbot*, *virut* e *trickbot*, invece, la detection rate resta alta in entrambi gli spazi: anche perturbando `Dur` in *feature space*, il classificatore sembra basarsi su altre feature per quelle famiglie. Quando i due scenari divergono, l'esito dipende dall'interazione fra realizzabilità della perturbazione sul traffico e rilevanza di `Dur` nella decisione del modello, più che da una generica resilienza del detector. La Figura 5.7 illustra il divario fra le due curve su TCP. Su UDP (Figura 5.8) l'effetto resta contenuto.

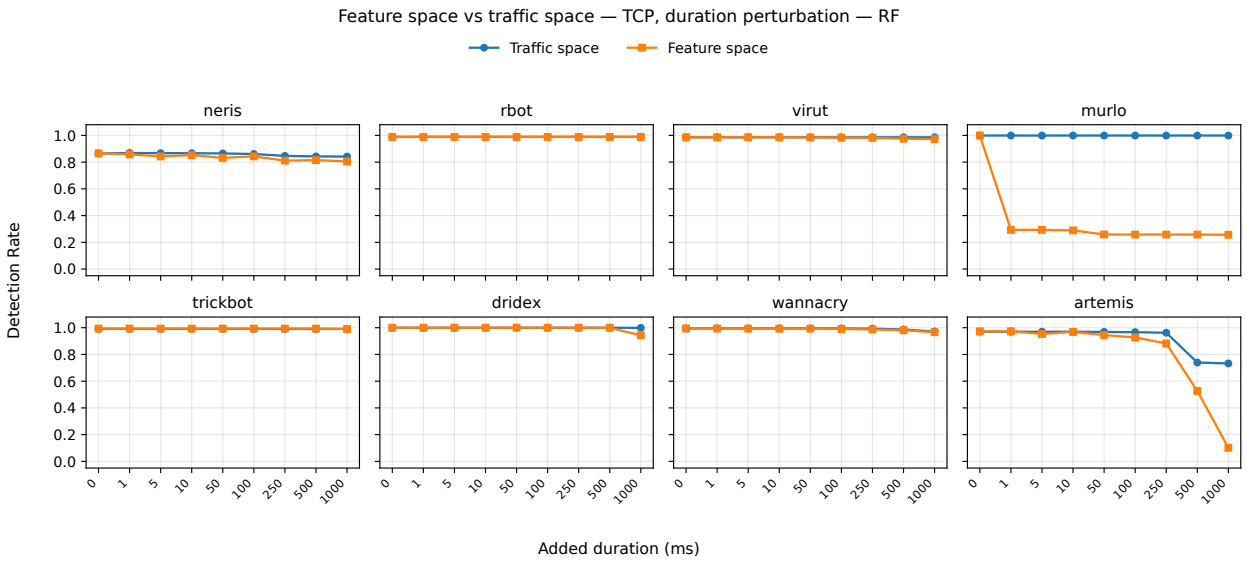



Figura 5.7: Detection rate (Random Forest) al crescere del ritardo applicato: confronto fra *traffic space* e *feature space*, traffico TCP. Ogni subplot corrisponde a una famiglia; in ascissa il ritardo in millisecondi (ultimo livello: 1 s).

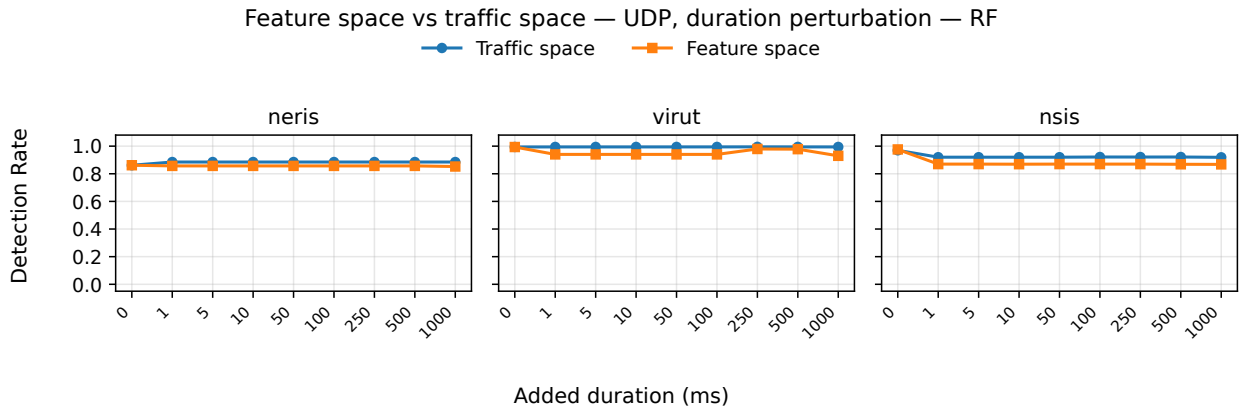



Figura 5.8: Detection rate (Random Forest) al crescere del ritardo applicato: confronto fra *traffic space* e *feature space*, traffico UDP. Ogni subplot corrisponde a una famiglia; in ascissa il ritardo in millisecondi (ultimo livello: 1 s).

Questo confronto va letto tenendo presente un limite della procedura: i risultati in *feature space* provengono da una singola ripetizione, mentre quelli in *traffic space* sono mediati su dieci ripetizioni (§4.6.2). Le stime in *feature space* sono quindi prive di banda di variabilità e il confronto è indicativo, in particolare sulle famiglie a basso supporto, dove già nello scenario *clean* la detection

rate può non coincidere fra i due spazi. Per questo l'analisi si limita alle famiglie con almeno mille flussi malevoli di test per protocollo (§4.2.5).

L'efficacia delle perturbazioni dipende inoltre dalla famiglia di botnet e dal classificatore. La variabilità tra le dieci ripetizioni si riflette anche nelle bande degli intervalli di confidenza al 95%: a ogni ripetizione cambiano la selezione casuale dei flussi benigni e la costruzione degli alberi, per cui la detection rate varia da una ripetizione all'altra. L'aggregazione di più alberi tipica degli *ensemble* (Random Forest e HGB) non si traduce in una maggiore robustezza: il modello che degrada di più cambia con la famiglia e il protocollo, segno che la vulnerabilità dipende dall'interazione tra le feature usate dal modello e quelle alterate dall'attacco.

Questi risultati vanno interpretati tenendo conto di come sono stati ottenuti, perché è proprio la procedura adottata a determinarne la validità. A differenza dei lavori che perturbano il traffico malevolo ma quantificano l'evasione direttamente sulle feature estratte dai flussi aggregati [2], qui le perturbazioni sono introdotte a livello di pacchetto, conformi al protocollo e verificate realizzabili, e le feature vengono riestratte a valle con la stessa procedura sperimentale (Argus, preprocessing, classificatore) adottata per il traffico non perturbato. Proprio questa prospettiva rende visibile un fenomeno che una valutazione in *feature space* non coglierebbe: alcune perturbazioni plausibili sul traffico, come il ritardo applicato su TCP, nella maggior parte dei casi non si traducono in uno spostamento utile delle feature e quindi non producono evasione, indipendentemente dalla robustezza del classificatore. In ogni caso, misurare la robustezza di un ML-NIDS sulle sole feature può discostare la stima dell'evasione da quella realizzabile: l'unica misura affidabile passa per perturbazioni realizzabili sul traffico e per la riestrazione delle feature dopo aggregazione in flussi e preprocessing, prima dell'inferenza del classificatore.

# 6. Conclusioni

## 6.1 Riepilogo

Questa tesi ha valutato la robustezza di un ML-NIDS all'evasione adottando un modello di minaccia più vicino a un attaccante reale. Le perturbazioni sono state introdotte in *traffic space*, a livello di pacchetto, e le feature sono state ricalcolate dopo aggregazione in flussi e preprocessing, con gli stessi strumenti e le stesse regole adottati per il traffico non perturbato. Il *threat model* descrive un attaccante *blind* in contesto *black box*: conosce in linea generale che il traffico è aggregato in flussi e classificato da un ML-NIDS, ma non ha accesso al modello, all'insieme delle feature né alle soglie di decisione, non costruisce modelli surrogati e non interroga il classificatore per ottimizzare le perturbazioni (§3.1.4). L'attaccante modifica soltanto il proprio traffico malevolo, nel rispetto di vincoli di protocollo verificati sul pacchetto. Si valutano separatamente due strategie: il *padding* del payload e il *ritardo* nell'invio dei pacchetti. L'analisi ha riguardato famiglie di botnet con almeno mille flussi malevoli di test per protocollo (§4.2.5), su traffico TCP e UDP, con tre classificatori e dieci ripetizioni con semi diversi.

## 6.2 Discussione del contributo

Il contributo principale non è dimostrare che un ML-NIDS possa essere evaso, ma definire *come* misurare l'evasione in modo coerente con un attaccante che agisce sul traffico dell'host compromesso. Gran parte della letteratura valuta la robustezza alterando direttamente le feature di flusso (§2.4.2) o assumendo capacità non disponibili a un attaccante (§2.6). Questa tesi inverte la prospettiva: le perturbazioni partono dal pacchetto, rispettano vincoli realizzabili (MTU, handshake, checksum) e solo a valle producono le feature osservate dal classificatore. In questo modo la stima dell'evasione resta legata a modifiche del traffico plausibili, anziché a valori numerici delle feature scollegati dal percorso di estrazione.

Il confronto controllato con la *feature space* (§5.4.2) quantifica quanto una valutazione più comune in letteratura possa discostarsi da quella realizzabile sul traffico. Su TCP il padding in

*feature space* fa apparire un crollo della detection rate molto più rapido che in *traffic space*, perché lì `TotBytes` può crescere senza il limite imposto dall'MTU. Su UDP, dove i pacchetti malevoli lasciano margine al padding realizzabile, i due spazi convergono: un risultato che emerge solo valutando sul traffico e che non sarebbe visibile limitandosi alle feature. Sul ritardo in TCP la situazione si inverte: in *traffic space* la detection rate resta piatta per quasi tutte le famiglie e suggerisce robustezza del detector, mentre in *feature space* alcune famiglie risultano vulnerabili quando `Dur` viene alterata direttamente.

Un secondo vantaggio metodologico riguarda la separazione fra realizzabilità della perturbazione e rilevanza delle feature per il modello. Le tabelle sull'effetto delle perturbazioni sulle feature (§5.4.1) documentano quanto ciascun attacco sposti `TotBytes` o `Dur` sui flussi malevoli inclusi nell'analisi. Quando la feature bersaglio resta quasi invariata, come accade per `Dur` su TCP a fronte del ritardo sul pacchetto, la detection rate piatta indica che l'attacco non raggiunge il classificatore in forma utile, indipendentemente dalla resilienza intrinseca del modello. Quando la feature si sposta ma il detector resta stabile, come per *rbot*, *virut* e *trickbot* anche in *feature space*, l'evasione fallisce perché quella grandezza pesa poco nella decisione per quella famiglia. Questa distinzione orienta l'interpretazione dei risultati e riduce il rischio di attribuire al modello robustezza che dipende in realtà dall'inefficacia dell'attacco sulle feature effettivamente usate.

Infine, la pipeline sperimentale rende il lavoro riproducibile e confrontabile. Perturbazioni, estrazione dei flussi, addestramento per famiglia e valutazione con metriche sulla classe malevola seguono un unico percorso, con valori numerici completi in appendice per il *traffic space* e per il riferimento in *feature space*. Per chi valuta o progetta un ML-NIDS, il messaggio operativo è che benchmark basati sulla sola manipolazione delle feature possono orientare in modo fuorviante le conclusioni su robustezza ed evasione. Una stima più affidabile passa per perturbazioni verificabili sul traffico e per la riestrazione delle feature lungo la stessa catena del sensore.

## 6.3 Limitazioni e sviluppi futuri

### 6.3.1 Limitazioni

Restano limiti legati alla natura offline dell'esperimento. Le perturbazioni agiscono su catture già registrate e non simulano interazioni con uno stack TCP attivo, ritrasmissioni o politiche di rete lungo il percorso. Padding e ritardo nell'invio dei pacchetti non sono stati applicati alla stessa cattura, per isolare l'effetto di ciascuna strategia sulle grandezze di flusso (§3.3.1). Il confronto con la *feature space* si basa su una singola ripetizione, a differenza delle dieci run del *traffic space*.

### 6.3.2 Sviluppi futuri

Estensioni naturali riguardano l'ampiezza dello spazio di attacco e della valutazione difensiva. Combinare padding e ritardo nell'invio dei pacchetti sulla stessa cattura, introdurre ulteriori operatori sul traffico e ampliare l'insieme dei classificatori permetterebbe di misurare la generalità delle conclusioni ottenute in *traffic space*. Sul fronte difensivo resta da verificare se tecniche di *adversarial training* e rinforzo del detector [4, 5] mantengano efficacia quando l'hardening usa evasioni generate a livello di pacchetto e riestratte con le stesse regole di aggregazione e preprocessing del sensore, anziché manipolazioni dirette delle feature di flusso.

# A. Detection rate in traffic space

Questa appendice raccoglie i valori numerici completi della detection rate (recall sulla classe malevola) nello scenario perturbato in *traffic space* (perturbazioni su PCAP e feature riestratte a valle, §5.3), sintetizzati graficamente nel Capitolo 5. Per ciascuna famiglia di botnet e classificatore si riporta la detection rate nello scenario *clean* e a ogni livello di perturbazione, distinta per protocollo di trasporto; una tabella raccoglie il padding del payload, l'altra la perturbazione sulla durata. Le famiglie incluse variano per protocollo secondo il criterio di §4.2.5: su TCP figurano le famiglie CTU-13 *neris*, *rbot*, *virut* e *murlo* e le quattro famiglie MCFP recenti *trickbot*, *dridex*, *wannacry* e *artemis*; su UDP solo *neris*, *virut* e *nsis*. La colonna *Clean* corrisponde al livello di perturbazione nullo e coincide con la colonna R (recall) delle Tabelle 5.1 e 5.2. Ogni valore è la media su dieci ripetizioni con semi diversi (§4.6.2); la variabilità tra ripetizioni è rappresentata negli intervalli di confidenza al 95% delle figure del Capitolo 5.

Tabella A.1: Detection rate (recall sulla classe malevola) al crescere del **padding** del payload (byte aggiunti) in *traffic space*, per protocollo di trasporto, famiglia e classificatore (media su dieci ripetizioni).

| **Famiglia** | **Modello** | **Clean** | **Padding (byte)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 5 | 10 | 50 | 200 | 500 | 1000 |
| **TCP** | | | | | | | | | | |
| | RF | 0.864 | 0.858 | 0.839 | 0.819 | 0.771 | 0.717 | 0.719 | 0.689 | 0.689 |
| *neris* | DT | 0.882 | 0.860 | 0.847 | 0.817 | 0.797 | 0.772 | 0.748 | 0.721 | 0.726 |
| | HGB | 0.849 | 0.840 | 0.832 | 0.802 | 0.795 | 0.737 | 0.739 | 0.718 | 0.709 |
| | RF | 0.988 | 0.988 | 0.984 | 0.955 | 0.924 | 0.912 | 0.908 | 0.909 | 0.907 |
| *rbot* | DT | 0.993 | 0.982 | 0.973 | 0.957 | 0.957 | 0.926 | 0.924 | 0.923 | 0.915 |
| | HGB | 0.992 | 0.981 | 0.975 | 0.951 | 0.940 | 0.928 | 0.915 | 0.917 | 0.908 |
| | RF | 0.985 | 0.978 | 0.978 | 0.975 | 0.970 | 0.950 | 0.919 | 0.913 | 0.912 |
| *virut* | | | | | | | | | | |

Tabella A.1 – continua dalla pagina precedente

| **Famiglia** | **Modello** | **Clean** | **Padding (byte)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 5 | 10 | 50 | 200 | 500 | 1000 |
| | DT | 0.987 | 0.974 | 0.973 | 0.973 | 0.969 | 0.949 | 0.933 | 0.929 | 0.937 |
| | HGB | 0.986 | 0.981 | 0.979 | 0.973 | 0.972 | 0.961 | 0.936 | 0.921 | 0.920 |
| | RF | 0.999 | 0.999 | 0.998 | 0.994 | 0.983 | 0.791 | 0.750 | 0.750 | 0.750 |
| *murlo* | DT | 0.998 | 0.908 | 0.908 | 0.908 | 0.873 | 0.815 | 0.811 | 0.806 | 0.802 |
| | HGB | 0.999 | 0.972 | 0.972 | 0.971 | 0.960 | 0.874 | 0.788 | 0.750 | 0.750 |
| | RF | 0.992 | 0.990 | 0.989 | 0.985 | 0.981 | 0.931 | 0.729 | 0.653 | 0.577 |
| *trickbot* | DT | 0.991 | 0.980 | 0.980 | 0.978 | 0.972 | 0.925 | 0.829 | 0.684 | 0.527 |
| | HGB | 0.992 | 0.987 | 0.981 | 0.981 | 0.975 | 0.940 | 0.789 | 0.743 | 0.735 |
| | RF | 1.000 | 1.000 | 1.000 | 1.000 | 0.998 | 0.987 | 0.968 | 0.920 | 0.895 |
| *dridex* | DT | 1.000 | 0.994 | 0.994 | 0.993 | 0.993 | 0.969 | 0.960 | 0.789 | 0.789 |
| | HGB | 1.000 | 1.000 | 1.000 | 0.998 | 0.999 | 0.986 | 0.921 | 0.863 | 0.829 |
| | RF | 0.995 | 0.995 | 0.995 | 0.993 | 0.982 | 0.960 | 0.948 | 0.941 | 0.940 |
| *wannacry* | DT | 0.993 | 0.991 | 0.991 | 0.979 | 0.960 | 0.949 | 0.948 | 0.948 | 0.948 |
| | HGB | 0.993 | 0.989 | 0.983 | 0.978 | 0.970 | 0.939 | 0.927 | 0.921 | 0.919 |
| | RF | 0.971 | 0.970 | 0.970 | 0.970 | 0.971 | 0.964 | 0.764 | 0.602 | 0.444 |
| *artemis* | DT | 0.968 | 0.965 | 0.965 | 0.965 | 0.964 | 0.954 | 0.722 | 0.402 | 0.402 |
| | HGB | 0.976 | 0.977 | 0.976 | 0.976 | 0.977 | 0.974 | 0.794 | 0.404 | 0.400 |
| **UDP** | | | | | | | | | | |
| | RF | 0.862 | 0.878 | 0.877 | 0.876 | 0.874 | 0.733 | 0.302 | 0.025 | 0.019 |
| *neris* | DT | 0.868 | 0.815 | 0.813 | 0.812 | 0.810 | 0.549 | 0.489 | 0.473 | 0.469 |
| | HGB | 0.864 | 0.887 | 0.886 | 0.876 | 0.873 | 0.232 | 0.066 | 0.004 | 0.004 |
| | RF | 0.994 | 0.995 | 0.995 | 0.980 | 0.977 | 0.965 | 0.751 | 0.328 | 0.305 |
| *virut* | DT | 0.995 | 0.995 | 0.995 | 0.971 | 0.968 | 0.959 | 0.950 | 0.951 | 0.951 |
| | HGB | 0.995 | 0.990 | 0.990 | 0.981 | 0.980 | 0.957 | 0.958 | 0.956 | 0.955 |

Tabella A.1 – continua dalla pagina precedente

| **Famiglia** | **Modello** | **Clean** | **Padding (byte)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 5 | 10 | 50 | 200 | 500 | 1000 |
| | RF | 0.970 | 0.963 | 0.939 | 0.917 | 0.904 | 0.767 | 0.632 | 0.161 | 0.153 |
| *nsis* | DT | 0.966 | 0.966 | 0.961 | 0.933 | 0.909 | 0.835 | 0.768 | 0.264 | 0.277 |
| | HGB | 0.977 | 0.974 | 0.970 | 0.903 | 0.894 | 0.828 | 0.715 | 0.206 | 0.216 |

Tabella A.2: Detection rate (recall sulla classe malevola) al crescere del ritardo applicato nella perturbazione sulla **durata** (millisecondi) in *traffic space*, per protocollo di trasporto, famiglia e classificatore (media su dieci ripetizioni).

| **Famiglia** | **Modello** | **Clean** | **Durata (ms)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 5 | 10 | 50 | 100 | 250 | 500 | 1000 |
| **TCP** | | | | | | | | | | |
| | RF | 0.864 | 0.868 | 0.867 | 0.867 | 0.865 | 0.861 | 0.847 | 0.843 | 0.842 |
| *neris* | DT | 0.882 | 0.886 | 0.886 | 0.885 | 0.882 | 0.880 | 0.873 | 0.864 | 0.856 |
| | HGB | 0.849 | 0.856 | 0.856 | 0.856 | 0.855 | 0.852 | 0.835 | 0.825 | 0.824 |
| | RF | 0.988 | 0.989 | 0.989 | 0.989 | 0.989 | 0.989 | 0.989 | 0.989 | 0.989 |
| *rbot* | DT | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 |
| | HGB | 0.992 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 |
| | RF | 0.985 | 0.986 | 0.986 | 0.986 | 0.986 | 0.986 | 0.986 | 0.986 | 0.986 |
| *virut* | DT | 0.987 | 0.988 | 0.988 | 0.988 | 0.988 | 0.988 | 0.988 | 0.988 | 0.988 |
| | HGB | 0.986 | 0.987 | 0.987 | 0.987 | 0.987 | 0.987 | 0.987 | 0.987 | 0.987 |
| | RF | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 |
| *murlo* | DT | 0.998 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 |
| | HGB | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 |
| | RF | 0.992 | 0.991 | 0.991 | 0.991 | 0.991 | 0.991 | 0.991 | 0.991 | 0.990 |
| *trickbot* | DT | 0.991 | 0.991 | 0.991 | 0.991 | 0.991 | 0.991 | 0.991 | 0.990 | 0.990 |

Tabella A.2 – continua dalla pagina precedente

| **Famiglia** | **Modello** | **Clean** | **Durata (ms)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 5 | 10 | 50 | 100 | 250 | 500 | 1000 |
| | HGB | 0.992 | 0.992 | 0.992 | 0.992 | 0.992 | 0.992 | 0.992 | 0.992 | 0.991 |
| | RF | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.999 |
| *dridex* | DT | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.999 | 0.993 | 0.948 |
| | HGB | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.999 |
| | RF | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.993 | 0.986 | 0.972 |
| *wannacry* | DT | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.992 | 0.989 | 0.983 | 0.974 |
| | HGB | 0.993 | 0.993 | 0.994 | 0.994 | 0.994 | 0.994 | 0.993 | 0.992 | 0.985 |
| | RF | 0.971 | 0.970 | 0.970 | 0.970 | 0.968 | 0.966 | 0.963 | 0.740 | 0.733 |
| *artemis* | DT | 0.968 | 0.965 | 0.965 | 0.965 | 0.964 | 0.964 | 0.960 | 0.762 | 0.727 |
| | HGB | 0.976 | 0.976 | 0.976 | 0.976 | 0.976 | 0.975 | 0.974 | 0.858 | 0.741 |
| **UDP** | | | | | | | | | | |
| | RF | 0.862 | 0.885 | 0.885 | 0.885 | 0.885 | 0.885 | 0.885 | 0.885 | 0.885 |
| *neris* | DT | 0.868 | 0.827 | 0.827 | 0.827 | 0.827 | 0.827 | 0.827 | 0.827 | 0.827 |
| | HGB | 0.864 | 0.887 | 0.887 | 0.887 | 0.887 | 0.887 | 0.887 | 0.887 | 0.887 |
| | RF | 0.994 | 0.994 | 0.994 | 0.994 | 0.994 | 0.994 | 0.995 | 0.995 | 0.994 |
| *virut* | DT | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 |
| | HGB | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 |
| | RF | 0.970 | 0.920 | 0.920 | 0.920 | 0.920 | 0.921 | 0.921 | 0.921 | 0.919 |
| *nsis* | DT | 0.966 | 0.936 | 0.936 | 0.936 | 0.936 | 0.937 | 0.933 | 0.862 | 0.826 |
| | HGB | 0.977 | 0.986 | 0.986 | 0.986 | 0.986 | 0.988 | 0.988 | 0.988 | 0.987 |

# B. Detection rate in feature space

Questa appendice raccoglie i valori numerici completi della detection rate (recall sulla classe malevola) nello scenario perturbato in *feature space* (§4.7.5), sintetizzati graficamente nel confronto con il *traffic space* di §5.4.2. Per ciascuna famiglia di botnet e classificatore si riporta la detection rate nello scenario *clean* e a ogni livello di perturbazione, distinta per protocollo di trasporto; una tabella raccoglie il padding del payload, l'altra la perturbazione sulla durata. Le famiglie incluse variano per protocollo secondo il criterio di §4.2.5. A differenza dell'Appendice A, i valori provengono da una singola ripetizione (seed 0).

Tabella B.1: Detection rate (recall sulla classe malevola) al crescere del **padding** del payload (byte aggiunti) in *feature space*, per protocollo di trasporto, famiglia e classificatore (singola ripetizione, seed 0).

| **Famiglia** | **Modello** | **Clean** | **Padding (byte)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 5 | 10 | 50 | 200 | 500 | 1000 |
| **TCP** | | | | | | | | | | |
| | RF | 0.866 | 0.294 | 0.155 | 0.130 | 0.085 | 0.087 | 0.084 | 0.025 | 0.023 |
| *neris* | DT | 0.883 | 0.529 | 0.210 | 0.194 | 0.142 | 0.121 | 0.093 | 0.091 | 0.088 |
| | HGB | 0.854 | 0.279 | 0.214 | 0.125 | 0.125 | 0.103 | 0.202 | 0.028 | 0.040 |
| | RF | 0.989 | 0.987 | 0.133 | 0.127 | 0.104 | 0.077 | 0.077 | 0.077 | 0.077 |
| *rbot* | DT | 0.993 | 0.992 | 0.142 | 0.084 | 0.083 | 0.083 | 0.083 | 0.083 | 0.083 |
| | HGB | 0.992 | 0.140 | 0.132 | 0.123 | 0.080 | 0.082 | 0.080 | 0.080 | 0.080 |
| | RF | 0.986 | 0.932 | 0.072 | 0.068 | 0.065 | 0.048 | 0.013 | 0.012 | 0.013 |
| *virut* | DT | 0.987 | 0.932 | 0.929 | 0.931 | 0.925 | 0.906 | 0.028 | 0.035 | 0.902 |
| | HGB | 0.988 | 0.935 | 0.076 | 0.072 | 0.070 | 0.067 | 0.028 | 0.015 | 0.013 |
| | RF | 0.999 | 0.988 | 0.988 | 0.280 | 0.243 | 0.059 | 0.032 | 0.032 | 0.032 |
| *murlo* | DT | 0.998 | 0.851 | 0.850 | 0.144 | 0.135 | 0.104 | 0.089 | 0.060 | 0.060 |

Tabella B.1 – continua dalla pagina precedente

| **Famiglia** | **Modello** | **Clean** | **Padding (byte)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 5 | 10 | 50 | 200 | 500 | 1000 |
| | HGB | 0.999 | 0.984 | 0.984 | 0.983 | 0.935 | 0.874 | 0.032 | 0.032 | 0.032 |
| | RF | 0.993 | 0.992 | 0.992 | 0.982 | 0.971 | 0.764 | 0.542 | 0.340 | 0.333 |
| *trickbot* | DT | 0.993 | 0.985 | 0.983 | 0.974 | 0.967 | 0.799 | 0.306 | 0.305 | 0.657 |
| | HGB | 0.992 | 0.828 | 0.828 | 0.816 | 0.808 | 0.758 | 0.595 | 0.595 | 0.602 |
| | RF | 1.000 | 1.000 | 1.000 | 1.000 | 0.994 | 0.976 | 0.963 | 0.894 | 0.846 |
| *dridex* | DT | 1.000 | 0.993 | 0.993 | 0.993 | 0.967 | 0.965 | 0.743 | 0.602 | 0.607 |
| | HGB | 1.000 | 1.000 | 0.995 | 0.995 | 0.982 | 0.972 | 0.979 | 0.979 | 0.981 |
| | RF | 0.994 | 0.993 | 0.995 | 0.974 | 0.962 | 0.937 | 0.928 | 0.922 | 0.923 |
| *wannacry* | DT | 0.993 | 0.984 | 0.984 | 0.970 | 0.960 | 0.940 | 0.950 | 0.950 | 0.950 |
| | HGB | 0.993 | 0.984 | 0.982 | 0.959 | 0.969 | 0.943 | 0.891 | 0.887 | 0.891 |
| | RF | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.952 | 0.643 | 0.594 | 0.381 |
| *artemis* | DT | 0.970 | 0.972 | 0.972 | 0.971 | 0.969 | 0.940 | 0.450 | 0.410 | 0.035 |
| | HGB | 0.972 | 0.960 | 0.961 | 0.967 | 0.967 | 0.939 | 0.417 | 0.061 | 0.000 |
| **UDP** | | | | | | | | | | |
| | RF | 0.862 | 0.854 | 0.854 | 0.851 | 0.840 | 0.709 | 0.566 | 0.007 | 0.004 |
| *neris* | DT | 0.869 | 0.763 | 0.763 | 0.763 | 0.761 | 0.042 | 0.001 | 0.001 | 0.001 |
| | HGB | 0.862 | 0.862 | 0.861 | 0.848 | 0.838 | 0.129 | 0.005 | 0.004 | 0.004 |
| | RF | 0.994 | 0.994 | 0.994 | 0.954 | 0.954 | 0.938 | 0.932 | 0.890 | 0.881 |
| *virut* | DT | 0.994 | 0.994 | 0.994 | 0.955 | 0.956 | 0.956 | 0.942 | 0.943 | 0.943 |
| | HGB | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.979 | 0.994 | 0.989 | 0.988 |
| | RF | 0.976 | 0.971 | 0.970 | 0.914 | 0.915 | 0.845 | 0.764 | 0.768 | 0.763 |
| *nsis* | DT | 0.977 | 0.948 | 0.945 | 0.939 | 0.927 | 0.833 | 0.887 | 0.828 | 0.827 |
| | HGB | 0.980 | 0.970 | 0.971 | 0.941 | 0.921 | 0.889 | 0.822 | 0.772 | 0.768 |

Tabella B.2: Detection rate (recall sulla classe malevola) al crescere del ritardo applicato nella perturbazione sulla **durata** (millisecondi) in *feature space*, per protocollo di trasporto, famiglia e classificatore (singola ripetizione, seed 0).

| **Famiglia** | **Modello** | **Clean** | **Durata (ms)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 5 | 10 | 50 | 100 | 250 | 500 | 1000 |
| **TCP** | | | | | | | | | | |
| | RF | 0.866 | 0.858 | 0.842 | 0.853 | 0.832 | 0.843 | 0.811 | 0.815 | 0.805 |
| *neris* | DT | 0.883 | 0.876 | 0.864 | 0.875 | 0.843 | 0.856 | 0.848 | 0.848 | 0.792 |
| | HGB | 0.854 | 0.851 | 0.853 | 0.853 | 0.845 | 0.840 | 0.748 | 0.729 | 0.728 |
| | RF | 0.989 | 0.989 | 0.989 | 0.989 | 0.989 | 0.989 | 0.988 | 0.988 | 0.988 |
| *rbot* | DT | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.992 |
| | HGB | 0.992 | 0.992 | 0.992 | 0.992 | 0.992 | 0.992 | 0.992 | 0.992 | 0.992 |
| | RF | 0.986 | 0.986 | 0.986 | 0.986 | 0.984 | 0.983 | 0.982 | 0.975 | 0.972 |
| *virut* | DT | 0.987 | 0.987 | 0.987 | 0.987 | 0.985 | 0.985 | 0.984 | 0.981 | 0.974 |
| | HGB | 0.988 | 0.988 | 0.988 | 0.987 | 0.987 | 0.986 | 0.986 | 0.985 | 0.975 |
| | RF | 0.999 | 0.293 | 0.293 | 0.289 | 0.259 | 0.258 | 0.258 | 0.258 | 0.256 |
| *murlo* | DT | 0.998 | 0.995 | 0.287 | 0.258 | 0.258 | 0.258 | 0.258 | 0.258 | 0.292 |
| | HGB | 0.999 | 0.293 | 0.293 | 0.293 | 0.293 | 0.293 | 0.293 | 0.293 | 0.293 |
| | RF | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.991 |
| *trickbot* | DT | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.993 | 0.992 | 0.991 | 0.990 |
| | HGB | 0.992 | 0.992 | 0.992 | 0.992 | 0.992 | 0.992 | 0.837 | 0.837 | 0.836 |
| | RF | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.999 | 0.944 |
| *dridex* | DT | 1.000 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 |
| | HGB | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.999 |
| | RF | 0.994 | 0.993 | 0.993 | 0.993 | 0.993 | 0.991 | 0.987 | 0.982 | 0.966 |
| *wannacry* | DT | 0.993 | 0.993 | 0.993 | 0.993 | 0.992 | 0.991 | 0.988 | 0.986 | 0.948 |
| | HGB | 0.993 | 0.993 | 0.993 | 0.993 | 0.992 | 0.992 | 0.993 | 0.993 | 0.990 |

Tabella B.2 – continua dalla pagina precedente

| **Famiglia** | **Modello** | **Clean** | **Durata (ms)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 5 | 10 | 50 | 100 | 250 | 500 | 1000 |
| | RF | 0.973 | 0.973 | 0.953 | 0.969 | 0.945 | 0.927 | 0.882 | 0.527 | 0.101 |
| *artemis* | DT | 0.970 | 0.971 | 0.951 | 0.956 | 0.865 | 0.547 | 0.534 | 0.103 | 0.034 |
| | HGB | 0.972 | 0.972 | 0.971 | 0.970 | 0.939 | 0.568 | 0.539 | 0.266 | 0.197 |
| **UDP** | | | | | | | | | | |
| | RF | 0.862 | 0.857 | 0.857 | 0.857 | 0.857 | 0.857 | 0.857 | 0.857 | 0.853 |
| *neris* | DT | 0.869 | 0.777 | 0.777 | 0.777 | 0.777 | 0.777 | 0.777 | 0.777 | 0.777 |
| | HGB | 0.862 | 0.858 | 0.859 | 0.859 | 0.859 | 0.859 | 0.859 | 0.858 | 0.858 |
| | RF | 0.994 | 0.940 | 0.940 | 0.940 | 0.940 | 0.940 | 0.979 | 0.978 | 0.930 |
| *virut* | DT | 0.994 | 0.925 | 0.925 | 0.925 | 0.925 | 0.925 | 0.925 | 0.925 | 0.920 |
| | HGB | 0.995 | 0.927 | 0.927 | 0.927 | 0.927 | 0.927 | 0.927 | 0.926 | 0.923 |
| | RF | 0.976 | 0.870 | 0.870 | 0.869 | 0.870 | 0.870 | 0.870 | 0.869 | 0.868 |
| *nsis* | DT | 0.977 | 0.873 | 0.873 | 0.873 | 0.873 | 0.873 | 0.871 | 0.873 | 0.871 |
| | HGB | 0.980 | 0.880 | 0.880 | 0.880 | 0.880 | 0.879 | 0.879 | 0.880 | 0.880 |

# Bibliografia


[1] Giovanni Apruzzese e Michele Colajanni. «Evading Botnet Detectors Based on Flows and Random Forest with Adversarial Samples». In: *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. 2018, pp. 1–8. DOI: 10.1109/NCA.2018.8548327.

[2] Giovanni Apruzzese, Aurore Fass e Fabio Pierazzi. «When Adversarial Perturbations meet Concept Drift: An Exploratory Analysis on ML-NIDS». In: *Proceedings of the 2024 Workshop on Artificial Intelligence and Security*. AISec '24. Salt Lake City, UT, USA: Association for Computing Machinery, 2024, 149–160. ISBN: 9798400712289. DOI: 10.1145/3689932.3694757. URL: https://doi.org/10.1145/3689932.3694757.

[3] Giovanni Apruzzese, Pavel Laskov e Johannes Schneider. «SoK: Pragmatic Assessment of Machine Learning for Network Intrusion Detection». In: *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. 2023, pp. 592–614. DOI: 10.1109/EuroSP57164.2023.00042.

[4] Giovanni Apruzzese et al. «Addressing Adversarial Attacks Against Security Systems Based on Machine Learning». In: *2019 11th International Conference on Cyber Conflict (CyCon)*. Vol. 900. 2019, pp. 1–18. DOI: 10.23919/CYCON.2019.8756865.

[5] Giovanni Apruzzese et al. «Deep Reinforcement Adversarial Learning Against Botnet Evasion Attacks». In: *IEEE Transactions on Network and Service Management* 17.4 (2020), pp. 1975–1987. DOI: 10.1109/TNSM.2020.3031843.

[6] Giovanni Apruzzese et al. «On the effectiveness of machine and deep learning for cyber security». In: *2018 10th International Conference on Cyber Conflict (CyCon)*. 2018, pp. 371–390. DOI: 10.23919/CYCON.2018.8405026.

[7] Giovanni Apruzzese et al. «The Role of Machine Learning in Cybersecurity». In: *Digital Threats* 4.1 (mar. 2023). DOI: 10.1145/3545574. URL: https://doi.org/10.1145/3545574.

[8] Dimitri Galli et al. «Evading ML Network Intrusion Detection Systems for Modbus TCP with Problem-Space Perturbations». In: *Proceedings of the Joint National Conference on Cybersecurity (ITASEC & SERICS 2026)*. Vol. 4198. CEUR Workshop Proceedings. CEUR-WS.org, 2026, p. 25. URL: https://ceur-ws.org/Vol-4198/paper25.pdf.

[9] S. García et al. «An empirical comparison of botnet detection methods». In: *Computers & Security* 45 (2014), pp. 100–123. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2014.05.011. URL: https://www.sciencedirect.com/science/article/pii/S0167404814000923.

[10] Dongqi Han et al. «Evaluating and Improving Adversarial Robustness of Machine Learning-Based Network Intrusion Detectors». In: *IEEE Journal on Selected Areas in Communications* 39.8 (ago. 2021), 2632–2647. ISSN: 1558-0008. DOI: 10.1109/jsac.2021.3087242. URL: http://dx.doi.org/10.1109/JSAC.2021.3087242.

[11] Mohammad J. Hashemi, Greg Cusack e Eric Keller. «Towards Evaluation of NIDSs in Adversarial Setting». In: *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks*. Big-DAMA '19. Orlando, FL, USA: Association for Computing Machinery, 2019, pp. 14–21. ISBN: 9781450369992. DOI: 10.1145/3359992.3366642. URL: https://doi.org/10.1145/3359992.3366642.

[12] Ivan Homoliak et al. «Improving Network Intrusion Detection Classifiers by Non-payload-Based Exploit-Independent Obfuscations: An Adversarial Approach». In: *EAI Endorsed Transactions on Security and Safety* 5.17 (2018), e4. DOI: 10.4108/eai.10-1-2019.156245. URL: https://publications.eai.eu/index.php/sesa/article/view/194.

[13] Ansam Khraisat et al. «Survey of intrusion detection systems: techniques, datasets and challenges». In: *Cybersecurity* 2.1 (2019), p. 20. DOI: 10.1186/s42400-019-0038-7. URL: https://doi.org/10.1186/s42400-019-0038-7.

[14] Mohamed Amine Merzouk et al. «Investigating the practicality of adversarial evasion attacks on network intrusion detection». In: *Annals of Telecommunications* 77.11 (2022),

pp. 763–775. DOI: 10.1007/s12243-022-00910-1. URL: https://doi.org/10.1007/s12243-022-00910-1.

[15] Stratosphere Laboratory. *Malware Capture Facility Project*. https://www.stratosphereips.org/datasets-malware. Stratosphere IPS Project, Czech Technical University. Ultimo accesso: 29 giugno 2026. 2017.

[16] Andrea Venturi et al. «DReLAB - Deep REinforcement Learning Adversarial Botnet: A benchmark dataset for adversarial attacks against botnet Intrusion Detection Systems». In: *Data in Brief* 34 (2021), p. 106631. ISSN: 2352-3409. DOI: https://doi.org/10.1016/j.dib.2020.106631. URL: https://www.sciencedirect.com/science/article/pii/S2352340920315110.

[17] Andrea Venturi et al. «Hardening machine learning based network intrusion detection systems with synthetic netflows». In: *CEUR WORKSHOP PROCEEDINGS*. Vol. 3731. CEUR-WS. 2024.

[18] Miel Verkerken et al. «"What is the Problem Space?" Defining Host-space Adversarial Perturbations against Network Intrusion Detection Systems». In: *Proceedings of the ACM Asia Conference on Computer and Communications Security*. ASIA CCS '26. Bangalore, India: Association for Computing Machinery, 2026, 1043–1059. ISBN: 9798400723568. DOI: 10.1145/3779208.3807482. URL: https://doi.org/10.1145/3779208.3807482.

[19] R. Vinayakumar et al. «Deep Learning Approach for Intelligent Intrusion Detection System». In: *IEEE Access* 7 (2019), pp. 41525–41550. DOI: 10.1109/ACCESS.2019.2895334.

[20] João Vitorino, Nuno Oliveira e Isabel Praça. «Adaptative Perturbation Patterns: Realistic Adversarial Learning for Robust Intrusion Detection». In: *Future Internet* 14.4 (2022). ISSN: 1999-5903. DOI: 10.3390/fi14040108. URL: https://www.mdpi.com/1999-5903/14/4/108.

[21] João Vitorino, Isabel Praça e Eva Maia. «Towards adversarial realism and robust learning for IoT intrusion detection and classification». In: *Annals of Telecommunications* 78.7 (2023), pp. 401–412. ISSN: 1958-9395. DOI: 10.1007/s12243-023-00953-y. URL: https://doi.org/10.1007/s12243-023-00953-y.

[22] Haonan Yan et al. « Automatic Evasion of Machine Learning-Based Network Intrusion Detection Systems ». In: *IEEE Transactions on Dependable and Secure Computing* 21.01 (gen. 2024), pp. 153–167. ISSN: 1941-0018. DOI: 10.1109/TDSC.2023.3247585. URL: https://doi.ieeecomputersociety.org/10.1109/TDSC.2023.3247585.

# Ringraziamenti

Desidero ringraziare il Prof. Mirco Marchetti e l'Ing. Dimitri Galli per la disponibilità e il sostegno con cui mi hanno seguito durante lo svolgimento di questa tesi. Grazie a loro ho potuto approfondire il mondo della sicurezza informatica, acquisire nuove conoscenze e consolidare le competenze maturate nel percorso di studi. Ringrazio tutti i professori che hanno contribuito, durante tutto il percorso, alla mia crescita professionale e personale.

Desidero ringraziare mia mamma, mio padre e mia sorella, che mi hanno sostenuto in ogni tappa di questo percorso. Un pensiero va anche ai miei nonni, che, seppur lontani, mi hanno sempre fatto sentire il loro affetto. Ringrazio Salvio, la persona che mi ha ispirato e mi ha dato la motivazione per iniziare questo percorso.

Ringrazio Lisa, la mia anima gemella, che mi ha supportato e sopportato in questi anni. Per lei non credo esistano parole in grado di descrivere quanto mi sia stata vicina. Ringrazio la sua famiglia, a partire da Marilena e Vito, con i quali ho trascorso momenti di spensieratezza e da cui ho raccolto i più saggi consigli.

Desidero ringraziare tutta la mia famiglia e le persone a me più care, insieme ai miei amici e i compagni che ho incontrato durante questo percorso. Tutti voi avete contribuito a costruire la persona che sono oggi.