UNIVERSITÀ DEGLI STUDI
DI SALERNO

Dipartimento di Informatica

TESI DI LAUREA MAGISTRALE IN INFORMATICA

PROGETTAZIONE DI UN SISTEMA SDN PER IL RILEVAMENTO, ISOLAMENTO E ANALISI DI ATTACCHI DDOS MEDIANTE MACHINE LEARNING

Relatore
Prof. Arcangelo Castiglione

Candidato
Josef Ascione
Mat.: 0522501587

A.A. 2024–2025

Indice

Capitolo 1

Introduzione

La sicurezza informatica è in continua evoluzione: negli ultimi anni attacchi DDoS (Distrubuited Denial of Service) hanno raggiunto volumi e sofisticazione prima impensabili: dal caso Mirai che ha compromesso milioni di dispositivi IoT agli attacchi multi-terabit che han messo in ginocchio intere CDN (Content Delivery Network) globali. In parallelo l'adozione di paradigmi SDN (Software-Defined Networking) ha trasformato drasticamente il modo in cui vengono progettate e gestite le infrastrutture di rete moderne.

Da un lato SDN promette agilità e controllo centralizzato mentre dall'altro, invece, introduce nuove superfici d'attacco e single point of failure che richiedono approcci di sicurezza innovativi. In questo scenario complesso si inserisce la ricerca presentata in questa tesi: l'idea di combinare l'intelligenza artificale con la programmabilità delle reti per creare sistemi di difesa più efficaci e reattivi.

1.1 Contesto e scenario di riferimento

Negli ultimi anni l'evoluzione tecnologica ha completamente ridisegnato il panorama della sicurezza di rete. Le architetture tradizionali basate su firewall perimetrali e appliance dedicate mostrano crescenti limitazione di fronte alla dinamicità delle infrastrutture moderne. Cloud computing, virtualizzazione e mobilità degli utenti hanno reso obsoleto il concetto stesso di perimetro di sicurezza.

In questo contesto, gli attacchi Distributed Denial of Service rappresentano una minaccia particolarmente insidiosa. Non si tratta più di semplici flood di pacchetti: gli attaccanti moderni orchestrano campagne multi-vettore che combinano attacchi volumetrici, protocol attacks e application-layer assaults. Le botnet contemporanee possono essere composte da milioni di dispositivi compromessi, dai server alle telecamere di sicurezza, dai router domestici agli smartphone.

Il paradigma SDN emerge come risposta naturale a queste sfide: la separazione tra piano di controllo e piano dati offre visibilità globale sulla rete e capacità di reazione in tempo reale. Tuttavia, la centralizzazione del controllo introduce nuove vulnerabilità: il controller diventa un target critico, il canale southbound può essere saturato, e le flow table degli switch possono essere sommerse da regole di attacco.

L'applicazione di tecniche di machine learning alla sicurezza delle reti SDN non è più una curiosità accademica, ma una necessità operativa. I volumi di traffico moderni rendono impossibile l'analisi manuale,

mentre la velocità di evoluzione delle minacce richiede sistemi capaci di apprendere e adattarsi autonomamente. Algoritmi di classificazione addestrati su pattern di traffico storico possono identificare anomalie in tempo reale, alimentando sistemi di risposta automatica.

1.2 Motivazioni e obiettivi della tesi

Questa ricerca nasce da una constatazione pratica emersa durante l'analisi della letteratura esistente. Molti lavori propongono soluzioni teoricamente eleganti ma operativamente complesse, richiedendo infrastrutture dedicate o modifiche architetturali difficili da giustificare in contesti reali. Altri si concentrano esclusivamente su accuracy algoritmica, trascurando aspetti cruciali come deployment complexity, maintainability e time-to-response.

L'obiettivo primario di questo lavoro è dimostrare che è possibile costruire sistemi di difesa DDoS efficaci combinando semplicità implementativa con prestazioni operative competitive. Non cerchiamo la soluzione perfetta, ma quella che funziona: un sistema che un amministratore di rete possa comprendere, implementare e manutenere senza richiedere una competenza specialistica in machine learning o programmazione SDN avanzata.

Un secondo obiettivo, emerso durante lo sviluppo, riguarda l'esplorazione di strategie di mitigazione alternative al dropping immediato. L'approccio tradizionale "rileva e blocca" può essere eccessivamente aggressivo, causando falsi positivi costosi in termini di business impact. Il concetto di sinkhole analitico che abbiamo sviluppato offre una terza via: intercettare il traffico sospetto per analisi approfondita, mantenendo la possibilità di recovery per comunicazioni erroneamente classificate.

Da un punto di vista metodologico, la ricerca esplora l'integrazione pratica tra componenti eterogenei: controller Ryu per l'orchestrazione SDN, server Flask per l'hosting del modello ML, e honeypot personalizzati per l'analisi del traffico rediretto. L'obiettivo è validare che questa architettura ibrida possa operare stabilmente in scenari realistici, gestendo traffico misto e mantenendo latenze accettabili.

Un aspetto che ci ha particolarmente interessati è la questione delle feature engineering per ambienti SDN. Il dataset UNSW-NB15, ampiamente utilizzato in letteratura, contiene feature estratte tramite deep packet inspection offline. Nel nostro scenario operativo, invece, tutte le informazioni devono essere derivabili in real-time dal controller, senza accesso al payload dei pacchetti. Questo vincolo ha guidato lo sviluppo di un approccio "controller-based" per l'estrazione delle feature, privilegiando metriche aggregate facilmente calcolabili durante il normale funzionamento del sistema.

1.3 Contributi originali del lavoro svolto

Il contributo principale di questa tesi risiede nella dimostrazione pratica che sistemi di difesa DDoS basati su ML e SDN possono essere implementati con architetture relativamente semplici, mantenendo efficacia operativa competitiva.

Il sistema sviluppato introduce tre elementi distintivi rispetto ai lavori esistenti. Prima di tutto, l'approccio "IDS-lite" per l'estrazione delle feature: invece di implementare analisi complesse direttamente nel controller, il sistema estrae un set minimale di feature controller-based ottimizzate per bassa latenza e le

invia a un servizio esterno dedicato. Questa separazione delle responsabilità semplifica significativamente la logica del controller mantenendo la flessibilità necessaria per l'evoluzione del modello ML.

Il secondo contributo riguarda l'implementazione del sinkhole analitico. A differenza degli approcci tradizionali che applicano drop immediato al traffico classificato come malevolo, il nostro sistema implementa una strategia di redirezione intelligente. Il traffico sospetto viene convogliato verso un nodo honeypot che applica euristiche di secondo livello, recuperando comunicazioni legittime erroneamente intercettate e fornendo capacità di analisi forense sul traffico effettivamente malevolo.

Dal punto di vista architetturale, il terzo contributo consiste nell'integrazione end-to-end di componenti eterogenei in una pipeline operativa stabile. Il sistema risultante mantiene coerenza operativa anche durante picchi di traffico intensi, aspetto spesso trascurato in prototipi accademici.

Particolarmente significativo è l'approccio alla validazione sperimentale. Invece di limitarci a test su dataset statici, abbiamo sviluppato un ambiente di simulazione completo basato su Mininet che riproduce topologie enterprise realistiche. La separazione tra rete attaccante (10.0.2.0/24) e rete vittima (10.0.1.0/24), mediata da routing software e controllo SDN, ha permesso di validare il comportamento del sistema in condizioni operative controllate ma realistiche.

Un contributo metodologico riguarda l'implementazione di telemetria estensiva per il monitoraggio delle prestazioni. Il sistema genera continuamente metriche su latenze ML, throughput del controller, efficacia della mitigazione e comportamento dell'honeypot. Questa visibilità operativa fornisce insight preziosi sulle dinamiche interne del sistema che potrebbero informare future ottimizzazioni.

Infine, la ricerca contribuisce al dibattito su compromessi architetturali in sistemi di sicurezza SDN. L'approccio centralizzato per la classificazione, pur introducendo un potenziale single point of failure, si è dimostrato vantaggioso in termini di semplicità gestionale e consistenza delle decisioni. Per deployment di media scala, questo trade-off appare ben bilanciato, suggerendo che la complessità non è sempre sinonimo di superiorità prestazionale.

1.4 Struttura della tesi

Il documento è organizzato in sette capitoli che seguono un percorso logico dalla teoria alla validazione pratica. Ogni capitolo costruisce sulle conoscenze acquisite nei precedenti, guidando il lettore attraverso le decisioni progettuali e implementative che hanno caratterizzato lo sviluppo.

Il Capitolo 2 presenta il background tecnologico necessario per comprendere le scelte architetturali adottate. L'esposizione parte dai concetti fondamentali di Software-Defined Networking, esplorando l'architettura a tre piani e il ruolo del protocollo OpenFlow. Particolare attenzione è dedicata al framework Ryu, scelto per l'implementazione del controller e a Mininet per la simulazione di rete. Il capitolo include inoltre una tassonomia degli attacchi DDoS moderni e un'introduzione alle tecniche di machine learning per l'analisi del traffico di rete.

Il Capitolo 3 analizza lo stato dell'arte in maniera critica, identificando punti di forza e limitazioni dei lavori esistenti. La revisione non si limita a un'esposizione passiva della letteratura, ma posiziona esplicitamente il nostro contributo rispetto ai lavori precedenti. Particolare enfasi è posta sui sistemi di

rilevamento DDoS in ambienti SDN, sull'utilizzo del machine learning per la sicurezza delle reti, e sulle soluzioni honeypot-based per la mitigazione intelligente.

Il Capitolo 4 presenta la progettazione del sistema proposto, motivando le scelte architetturali sulla base dei requisiti operativi identificati. L'architettura generale viene descritta attraverso i suoi componenti principali: controller SDN, servizio di classificazione ML, e sinkhole analitico. Sezioni dedicate approfondiscono il ruolo del controller nel processo di rilevamento, l'estrazione delle feature in stile UNSW-NB15, e le strategie di comunicazione tra componenti distribuiti.

Il Capitolo 5 descrive l'implementazione concreta del sistema, fornendo dettagli sufficienti per la riproducibilità degli esperimenti. Vengono presentati i file sorgente principali, la configurazione dell'ambiente di test, e le procedure per la raccolta delle metriche. Questo capitolo ponte tra progettazione teorica e validazione sperimentale, documentando le sfide implementative incontrate e le soluzioni adottate.

Il Capitolo 6 presenta la valutazione sperimentale attraverso test sistematici in scenari controllati. L'analisi copre metriche di performance del sistema ML, efficacia della mitigazione, e impatto sul traffico legittimo. I risultati sono discussi criticamente, evidenziando sia i successi che le limitazioni emerse durante la sperimentazione. Particolare attenzione è dedicata all'analisi della stabilità operativa e della robustezza del sistema sotto carico.

Il Capitolo 7 conclude la trattazione con una sintesi critica dei risultati raggiunti e una discussione delle prospettive di sviluppo futuro.

Capitolo 2

Background Tecnologico

2.1 Software Defined Networking (SDN): concetti base e architettura

L'archetipo "SDN" - Software Designed Network - ha reinventato il modo di vedere le infrastrutture di rete, introducendo una nuova idea : separare il piano di controllo da quello dei dati. Anche se potrebbe sembrare un'idea prettamente accademica, ha diverse implicazioni per quanto riguarda la gestione, la sicurezza e l'evoluzione di tutte le reti che conosciamo oggi.[7]

Da un lato, la centralizzazione logica del controllo abilita automazione avanzata e cicli di sviluppo rapidi. Dall'altro, invece, introduce nuove sfide in termini di scalabilità e resilienza che richiedono approcci architetturali mirati.

2.1.1 I tre piani dell'architettura SDN

L'architettura si divide in tre livelli specifici, ognuno con le proprie responsabilità e compiti. Questi ultimi, collaborando tra loro, contribuiscono alla flessibilità complessiva del sistema.[5]

Application Plane (Piano Applicativo). In questo livello vi sono tutte le applicazioni di rete che definiscono la policy di alto livello, gli obiettivi di sicurezza e gli intenti operativi. Attraverso le "NBI" (Northbound APIs) esse riescono a comunicare con il controller anche se non sono a conoscenza di tutti i dettagli riguardanti il piano inferiore. Questo elevato livello di astrazione consente agli sviluppatori delle suddette di non doversi interessare ai dettagli implementativi della rete e di potersi concentrare sulla propria logica applicativa.

Control Plane (Piano di Controllo). Il centro dell'architettura, dove il controller SDN possiede una vista totale e coerente dell'intera topologia di rete. Qui vengono introdotte le regole concrete, che vengono poi distribuite a tutti i dispositivi presenti nel piano inferiore. Il controller gestisce tutte le decisioni di instradamento, lo stato della rete e coordina il dialogo tra le diverse applicazioni.

Data Plane (Piano Dati). Questo piano è formato da tutti i dispositivi di rete programmabili quali switch e router che eseguono le operazioni di forwarding, basate su tabelle di flusso configurate dai controller. Mantengono pipeline di elaborazione match-action programmabili permettendo alla rete di avere comportamenti sofisticati definiti centralmente.

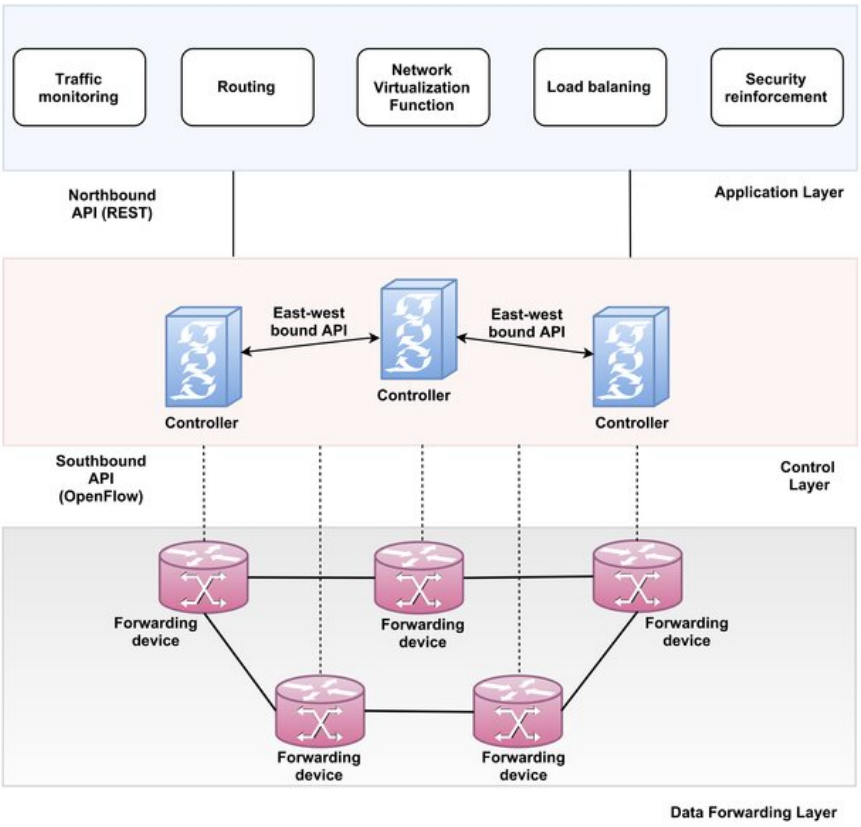

Figura 2.1: Architettura SDN a tre piani: *application plane* (intenti via NBI), *control plane* (stato + policy $\rightarrow$ regole) e *data plane* (pipeline *match–action* via SBI).

2.1.2 OpenFlow: il protocollo che ha reso possibile SDN

Openflow è il primo protocollo standardizzato per la comunicazione tra controller e switch programmabili, il quale ha avuto un enorme successo grazie ad una sua semplice caratteristica : un'interfaccia semplice, aperta e vendor-neutral.[10]

Il suo funzionamento si basa sulle **tabelle di flusso** le quali contengono le regole composte da 3 elementi di base: *match fields*, ovvero i criteri di matching sui pacchetti (ad esempio indirizzo IP, porta e protocollo), le *actions* (azioni da eseguire sui pacchetti nel caso in cui vi sia un riscontro: forward, drop e modify) ed infine i *counters* utilizzati per lo studio statistico ed il monitoraggio del traffico.

Possiede due modalità operative che caratterizzano l'approccio SDN: la modalità **proattiva**, dove le regole vengono installate ed attuate in modo preventivo basandosi su policy decise a priori, senza dover attendere un "match", e quella **reattiva** dove il primo pacchetto di ogni nuovo flusso di dati genera un evento PacketIn verso il controller per la decisione di instradamento. Anche se introduce una leggera latenza, questa seconda modalità offre flessibilità decisionale in tempo reale.

2.1.3 Vantaggi e limiti dell'approccio SDN

I benefici del modello SDN sono osservabili in molti ambiti operativi: la **programmabilità** consente di automatizzare molte configurazioni complicate implementando policy dinamiche tramite strumenti DevOps familiari; il **multi-tenancy** sicuro permette di isolare o separare il traffico di diversi tenant o applicazioni senza dover ricorrere a infrastrutture fisicamente separate; il **service chaining** abilita l'instradamento del traffico attraverso sequenze controllate di funzioni di rete (firewall, load balancer, DPI) in modo orchestrato e dinamico.

SDN, però, introduce anche delle complessità operative da non sottovalutare, come ad esempio: La **scalabilità del controller** potrebbe essere complicata da gestire nel momento in cui i dispositivi da esso gestiti crescono in modo significativo. La **consistenza degli aggiornamenti** ha bisogno di meccanismi sofisticati per far sì che degli stati transitori non si rivelino inconsistenti durante le modifiche delle policy. Infine, la **sicurezza sul canale di controllo** ha bisogno di autenticazione e cifratura robuste per evitare e prevenire compromissioni che avrebbero un impatto disastroso su tutta la rete.[7]

L'evoluzione verso data plane che siano indipendenti dai protocolli, come P4, aumenta ancord di più le possibilità di programmabilità consentendo di definire parse e pipeline di elaborazioni completamente personalizzate.[1]

2.2 Il ruolo del controller: focus su Ryu

In SDN il controller è un "controller intelligente" tra gli intenti applicativi di alto livello e le configurazioni concrete dei dispositivi in rete. In questo contesto Ryu emerge come un'ottima scelta per quanto riguarda la ricerca e la prototipazione offrendosi come framework Python modulare e documentato.[15]

2.2.1 Architettura e modello di programmazione

Ryu possiede un'architettura event-oriented che semplifica, in modo significativo, lo sviluppo di applicazioni SDN. Ogni applicazione Ryu estende la sua classe base, `RyuApp`, e registra dei gestori di eventi tramite il decoratore `@set_ev_cls` consentendo, così, una programmazione reattiva naturale.

Gli eventi principali includono `PacketIn` (per pacchetti che non hanno riscontri con le regole già esistenti), `FlowRemoved` (per notificare la scadenza delle regole) e `PortStatus` (per i cambiamenti di stato delle porte). Le strutture `ofproto_v1_3` e `parser` forniscono le primitive utili a comporre match conditions, azioni (come output, set-field e push/pop VLAN) e istruzioni più complesse come gruppi e meter per il rate limiting.

Il modulo `ryu.lib.packet` semplifica in modo notevole il parsing dei protocolli di rete supportando tutti i layer, da Ethernet fino a protocolli applicativi comuni eliminando, in tal modo, la necessità di manipolare direttamente i byte del payload.

2.2.2 Pattern operativo tipico

Un tipico flusso operativo in Ryu segue questi passi: come prima cosa installa una regola di *table-miss* a priorità bassa che dirige tutti i pacchetti senza match verso il controller tramite `PacketIn` il quale, al ricevimento di tali dati, analizzerà header e paylod per poi estrarre tutte le informazioni utili, come MAC, protocolli e stati di trasporto.

Tramite queste analisi il controller decide il forwarding e installa tutte le regole a priorità superiore con i giusti timeout : `idle_timeout` per le regole non più utilizzate, `hard_timeout` per scadenza assoluta. Periodicamente ha la possibilità di raccogliere dati statistici tramite `FlowStatsRequest` da dare poi come input ad algoritmi di ottimizzazione o di rilevamento delle minacce.

In ambienti di laboratorio con Open vSwitch, la configurazione è particolarmente diretta: gli switch si connettono automaticamente al controller sulla porta TCP 6653, e l'applicazione può esporre endpoint REST per orchestrazione esterna o integrazione con sistemi di monitoraggio.

2.3 Il framework Mininet per la simulazione di reti

Mininet è una soluzione elegante per quanto riguarda il problema della validazione di soluzioni SDN in ambienti controllati e riproducibili. Il framework riproduce alla perfezione reti complete instanziando host come processi linux isolati in network namespaces che vengono collegati tra di loro tramite virtual ethernet pairs e switch software. [8]

2.3.1 Architettura e capacità

La sua forza risiede nell'essere in grado di creare topologie di rete realistiche su una singola macchina mantenendo, però, l'isolamento completo tra tutti i nodi emulati. Ogni host emulato possiede il proprio stack di rete, è in grado di eseguire applicazioni reali e può comunicare attraverso l'infrastruttura virtuale proprio come farebbe in una rete fisica.

L'integrazione con controller SDN esterni avviene in modo trasparente: è sufficiente specificare l'indirizzo IP del controller e forzare l'uso di OpenFlow 1.3 sui virtual switch. Questo consente di validare applicazioni Ryu o altri controller senza necessità di hardware dedicato.

La sua CLI interattiva (`mn`) fa in modo che l'esplorazione ed il testing siano semplici, offrendo comandi molto familiari come `pingall` per testare la connettività, `iperf` per misurare il throughput e `tcpdump` per analizzare il traffico. È possibile, inoltre, definire topologie personalizzate attraverso API Python specificando caratteristiche dei link (bandwidth, latency, loss) per simulare condizioni di rete realistiche.

2.3.2 Limiti e considerazioni pratiche

Presenta però alcuni limiti intrinseci che bisogna considerare : la fedeltà temporale non è equivalente a quella di un hardware dedicato e le prestazioni sono vincolate alla capacità della macchina che ospita la macchina virtuale. Inoltre, alcuni comportamenti presenti in hardware specifici (come ad esempio buffering e scheduling avanzato) non sono replicati in modo egregio.

Tuttavia, per ricerca in ambito SDN e validazione di algoritmi di sicurezza, il compromesso tra realismo e controllo è spesso ottimale. La possibilità di creare scenari riproducibili, modificare i parametri di rete dinamicamente e osservare comportamenti in condizioni controllate supera largamente i limiti per la maggior parte dei casi d'uso sperimentali.

2.4 Modelli di attacco DDoS: tassonomia e indicatori

Gli attacchi Distributed Denial of Service rappresentano una delle minacce più persistenti e dannose per le infrastrutture di rete moderne. La loro efficacia deriva dalla capacità di coordinating molteplici sorgenti per concentrare un volume di traffico tale da degradare o negare completamente la disponibilità di servizi target.[6]

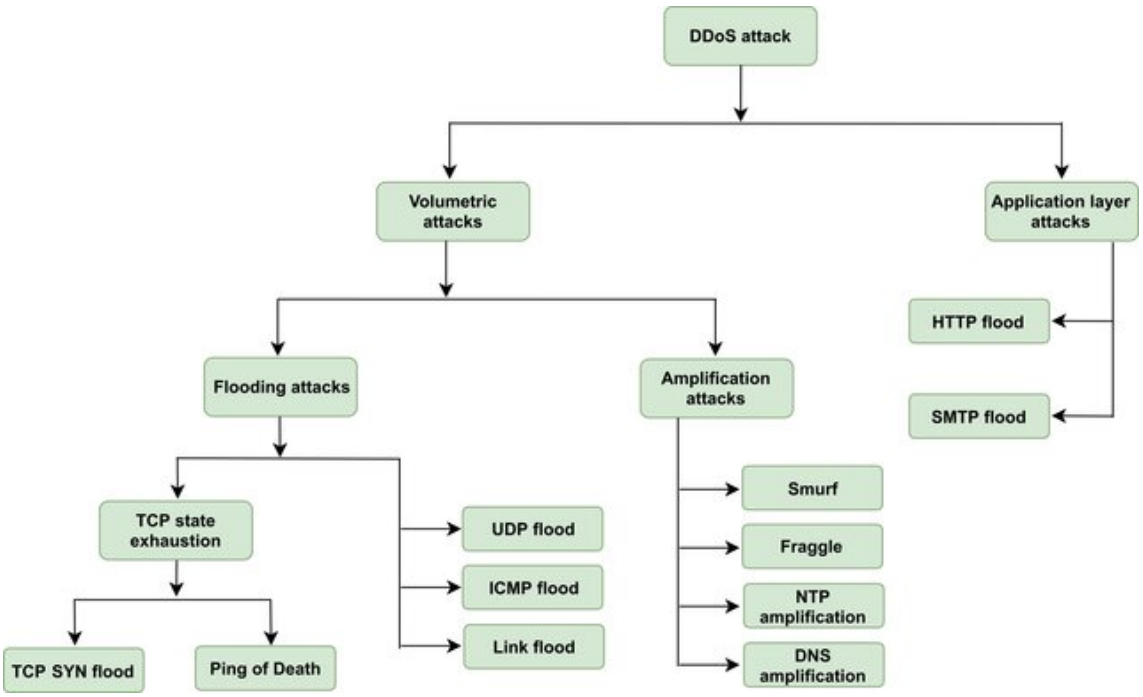

Figura 2.2: Classificazione essenziale degli attacchi DDoS in due famiglie principali: *volumetrici* (modalità di flooding e di amplificazione) e *applicativi*, con esempi rappresentativi per ciascun ramo.

2.4.1 Attacchi volumetrici: il peso dei numeri

Gli attacchi volumetrici hanno come obbiettivo quello di saturare la capacità di banda o le risorse di elaborazione a disposizione dei dispositivi di rete attraverso sheer volume ed hanno due principali varianti operative:

Il **flooding** diretto genera un flusso di traffico elevato verso l'obiettivo utilizzando una varietà di protocolli come UDP, ICMP e TCP (quest'ultimo con tecniche di esaurimento degli stati TCP - Syn Flood). Questi attacchi vengono orchestrati, in modo molto semplice, tramite una botnet che deve essere, però, di dimensioni considerevoli per essere efficace contro le infrastrutture moderne.

Gli attacchi di **amplificazione** hanno un approccio più sofisticato: sfruttando servizi legittimi (come DNS, NTP e SSDP) che rispondono con payload significativamente più grandi delle richieste iniziali, un attaccante è in grado di amplificare il traffico inviato di un fattore del 10-100x, rendendo l'attaco molto efficace dal punto di vista di risorse necessarie.

Gli indicatori caratteristici includono picchi anomali in bit per secondo e pacchetti per secondo, asimmetria marcata tra traffico in ingresso e in uscita, diversificazione geografica anomala delle sorgenti e rapido riempimento delle code di rete sui router di confine.

2.4.2 Attacchi a livello applicativo: la sottigliezza dell'approccio

Più subdoli ma altrettanto devastanti sono gli attacchi che sfruttano delle falle nella logica applicativa piuttosto che colpire l'infrastruttura di rete sottostante, utilizzando richieste apparentemente legittime ma computazionalmente costose per il backend dell'obbiettivo oppure pattern di connessione lenti ma persistenti che mantengono occupate le risorse del server.

Gli attacchi HTTP layer 7, come Slowloris o HTTP flood risultano molto effiaci anche sfruttando volumi di traffico relativamente modesti, in più, la loro identificazione richiede analisi più sofisticate che prendono in considerazione pattern comportamentali invece che delle semplici soglie volumentriche.

Tipici indicatori di cio' sono l'aumento del tasso di richieste per sessione, connessioni di lunga durata con un trasferimento di dati quasi nullo, crescita anomala delle code applicative e incremento dei timeout del servizio target.

2.4.3 Indicatori trasversali e strategie di identificazione

Indipendentemente dalla categoria di appartenenza, alcuni segnali sono trasversali a tutti gli attacchi DDoS: a livello di protocollo si può spesso osservare un incremento di rapporto tra pacchetti SYN e connessioni completate, variazioni anomale dei valori di Time-To-Live (ciò ci suggerisce che le sorgenti sono molto eterogenee) e pattern di fan-in estremi verso specific endpoint.

L'integrazione con paradigmi SDN offre opportunità uniche dal punto di vista della mitigazione: il controller ha la possibilità di implementare filtri dinamici basati su pattern di traffico, applicare rate limiting granulare legato al flusso e/o alla sorgente orchestrando il reindirizzamento del traffico verso endpoint appositi che faranno da honeypot o sistemi di scrubbing dedicati.

2.5 Machine Learning per l'analisi del traffico di rete

Di recente l'applicazione di tecniche di machine learning allo studio del traffico di rete ha guadagnato un'attenzione crescente grazie alla capacità di tali modelli di identificare, in maniera molto precisa, pattern complessi riuscendo ad adattarsi all'evoluzione delle minacce. Nel contesto SDN queste caratteristiche abilitano i sistemi di difesa ed essere reattivi ed intelligenti.[20]

2.5.1 Pipeline di classificazione del traffico

Una classica pipeline di classificazione del traffico è divisa in fasi sequenziali. L'**estrazione delle feature**, che opera a livello di flusso, considera statistiche aggregate come contatori di pacchetti e byte, rate di trasmissione, durate delle connessioni e stati del protocollo di trasporto

Il **preprocessing** normalizza le feature numeriche e codifica variabili categoriche (protocolli, servizi, stati di connessione). Questa fase è cruciale per garantire che algoritmi con diverse sensibilità ai range dei valori possano operare efficacemente.

La fase di **addestramento** utilizza algoritmi supervisionati (Random Forest, Support Vector Machines, alberi decisionali o reti neurali) con caratteristiche specifiche in termini interpretabilità, velocità di inferenza e robustezza.La **validazione** impiega metriche appropriate per scenari sbilanciati: precision, recall, F1-score e area sotto la curva ROC.

2.5.2 Integrazione con architetture SDN

Le previsioni dei modelli possono attivare delle reazioni automatiche attraverso i controller i quali, dopo esser stati informati di un flusso malevolo, possono installare regole di filtering, applicare rate limiting dinamico, o reindirizzare il traffico verso honeypot per analisi più approfondite.

L'integrazione presenta diverse sfide: la latenza di inferenza deve essere compatibili con i requisiti di risposta real-time della rete, il modello deve essere in grado di operare su traffico in continua evoluzione e il sistema deve gestire nel modo più opportuno possibili falsi positivi che potrebbero impattare il traffico legittimo.

2.5.3 Considerazioni pratiche e limiti

Due problemi emergono frequentemente in applicazioni reali. Lo **sbilanciamento delle classi** richiede tecniche di campionamento o algoritmi sensibili al costo per evitare che il modello sia dominato dal traffico normale. Il **concept drift** impone strategie di aggiornamento continuo per mantenere l'efficacia nel tempo.

In scenari ostili, la robustezza ad attacchi avversari diventa critica: gli attaccanti potrebbero deliberatamente modificare il loro traffico per eludere i classificatori, richiedendo approcci di difesa adattivi.

2.6 Dataset UNSW-NB15: struttura e feature rilevanti

Il dataset UNSW-NB15 rappresenta una risorsa fondamentale per la ricerca in network security, offrendo un corpus diversificato di traffico normale e malevolo con feature pre-estratte che facilitano lo sviluppo e la validazione di modelli di machine learning.[13], [19]

2.6.1 Struttura e composizione

Il dataset raccoglie traffico ibrido che include comunicazioni normali e nove famiglie distinte di attacchi: DoS, Reconnaissance, Exploits, Generic, Fuzzers, Worms, Shellcode, Analysis e Backdoors. Questa diversità è particolarmente preziosa per valutare la capacità di generalizzazione dei modelli su tipologie di minacce eterogenee.

Per ogni record, sono disponibili 49 feature estratte utilizzando strumenti specializzati come Argus e Bro/Zeek. Il dataset fornisce split predefiniti di training e test che sono stati ampiamente adottati dalla comunità scientifica, facilitando comparazioni dirette tra diversi approcci metodologici.

2.6.2 Feature significative per il rilevamento DDoS

Per la classificazione di attacchi DDoS, alcune feature si rivelano particolarmente discriminative. I contatori direzionali (`spkts`, `dpkts`, `sbytes`, `dbytes`) catturano l'asimmetria tipica del traffico di attacco. Le metriche di rate (`sload`, `dload`) evidenziano comportamenti di flooding.

Le feature di densità temporale come `ct_srv_dst`, `ct_dst_ltm` e `ct_state_ttl` quantificano la concentrazione di flussi verso specifiche destinazioni o servizi, caratteristica distintiva degli attacchi coordinati. Lo stato di trasporto (`state`) fornisce informazioni sui pattern di connessione che differenziano traffico legittimo da flooding sintetico.

Durante l'utilizzo del dataset è importante verificare l'assenza di data leakage tra set di training e test, particolare attenzione deve essere posta a record quasi duplicati o finestre temporali sovrapposte che potrebbero gonfiare artificialmente le performance del modello.

2.7 Concetti di honeypot, sinkhole e mitigazione distribuita

I sistemi di honeypot e sinkhole rappresentano componenti strategici nelle architetture di difesa moderne, offrendo capacità di deception, raccolta intelligence e mitigazione attiva degli attacchi in corso.[17]

2.7.1 Honeypot: attrattori intelligenti

Un honeypot è una risorsa informatica volutamente esposta e progettata per attirare attività malevole che consente di osservare e studiare in modo dettagliato le tecniche e gli obbiettivi degli attaccanti. Tradizionalmente si distingue tra due tipologie: a bassa interazione (emulazioni semplici e sicure) e ad alta interazione (sistemi completi ma più rischiosi).

In contesti DDoS gli honeypot servono principalmente alla caratterizzazione dei patter di attacco, all'identificazione di botnet attive e alla raccolta di campioni di traffico malevolo per poter addestrare con dati reali vari sistemi di machine learning. La loro integrazione in paradigmi SDN porta ad un deployment dinamico e alla riconfigurazione rapida basata sulla rapida evoluzione delle minacce.

2.7.2 Sinkhole: assorbimento e analisi

Un sinkhole è una destinazione controllata dal sistema di difesa verso il quale viene convogliato tutto il traffico indesiderato per assorbirlo e analizzarlo in modo dettagliato i quali, a differenza degli honeypot, sono progettati per gestire volumi elevati di traffico senza mantenere l'illusione di essere un servizio legittimo.

Il controller ha la possibilità di decidere se reindirizzare il traffico verso il sinkhole basandosi sulle classificazioni real-time, capacità particolarmente preziosa durante gli attacchi DDoS, durante i quali il traffico malevolo può essere isolato ed analizzato in modo da non impattare su i servizi legittimi.

2.7.3 Mitigazione distribuita e orchestrazione

Il controller SDN può implementare strategie articolate: installazione di filtri granulari per prefisso, porta o protocollo; applicazione di meter per rate limiting dinamico; riscrittura delle destinazioni per il reindirizzamento verso honeypot o sinkhole.

A livello inter-dominio, standard come DOTS (DDoS Open Threat Signaling) facilitano la segnalazione automatica di attacchi in corso verso provider di scrubbing e partner nella catena di mitigazione.[12] L'ingress filtering secondo BCP 38 (Best Current Practice 38) riduce l'efficacia degli attacchi basati su spoofing, mentre tecniche di rate limiting distribuite possono contenere l'impatto anche di botnet di grandi dimensioni.[3]

L'integrazione di questi elementi in un'architettura coerente richiede orchestrazione sofisticata ma offre capacità di difesa superiori a quelle ottenibili con approcci tradizionali isolati.

Capitolo 3

Stato dell'Arte

L'evoluzione delle reti software-defined ha aperto scenari innovativi per la ricerca e l'evoluzione della sicurezza informatica ma, allo stesso tempo, ha introdotto nuove superfici d'attacco.Se da un lato la centralizzazione del controllo offre visibilità globale e capacità di reazione rapida, dell'altro, essa presenta il rischio di diventare un single point of failure. In più, in questo contesto, l'applicazione di tecniche e modelli di machine learning atti al rilevamento e alla mitigazione di attacchi DDoS rappresenta un'area di ricerca particolarmente attiva e promettente.

3.1 Sistemi di rilevamento DDoS in ambienti SDN

La separazione tra il piano di controllo e quello dei dati, introdotta dal paradigma SDN, rende possibile il controllo programmabile e centralizzato della rete aprendo, però, a nuove superfici d'attacco specifiche come: saturazione del canale southbound, overflow delle flow table ed il `packet_in` flooding verso il controller. Non è da trascurare che l'architettura richieda approcci di sicurezza moderni e pensati apposta per essa, diversamente dalle reti tradizionali.

In letteratura vi è una distinzione fondamentale tra due categorie: rilevamento *signature-based* e *anomaly-based*. Per quanto riguarda SDN entrambi possono risiedere nel controller, ai bordi *edge* o in una combo ibrida. Alcune delle strategie più diffuse sono il *flow filtering*, *rate limiting*, il *traceback* e l'uso di honeypot in modo da ridurre la percentuale di falsi positivi. [16] fornisce una tassonomia completa delle contromisure per DDoS in IoT/SDN, evidenziando vantaggi (programmabilità, vista globale) e rischi (collo di bottiglia del controller).

[2] presenta un approccio molto interessante per il presente lavoro: gli autori valutano diversi classifier su dataset controller based, installando in Ryu il quello che risulta essere il migliore (LGBM), per poter rilevare e mitigare in tempo reale attacchi su SD-IOT. L'aspetto più interessante è che l'estrazione delle feature dagli eventi di OpenFlow si rileva efficace ed efficiente, in linea con la latenza operativa del controller, risultato che ha influenzato in modo diretto le scelte architetturali del nostro prototipo.

3.2 Approcci centralizzati vs distribuiti

Gli approcci **centralizzati** nel controller massimizzano la coerenza delle policy semplificando la gestione ma rischiando, così, di sovraccaricare il sistema in scenari di gestione di attacchi volumetrici.

Gli approcci distribuiti (edge switch, multi-controller) delegano tutta la fase di pre-filtraggio/aggregazione, riducendo così il carico sul controller pagando un prezzo che è quello di avere un grado maggiore complessità e coordinamento da gestire. La tassonomia di [16] correla posizione del rilevamento, tecnica (firme/anomalia), strategia di mitigazione e scenari applicativi, suggerendo soluzioni ibride come best practice in reti eterogenee.

Nel contesto del nostro progetto la scelta di applicare una logica di rilevamento centralizzata nel controller Ryu è data da due motivi: la semplicità implementativa che facilita di molto la prototipazione rapida e la necessita di avere una visione coerente e globale di tutti gli attacchi in corso

3.3 Utilizzo del machine learning nella sicurezza delle reti

In SDN i modelli di ML vegono addestrati su dataset consolidati, come NSL-KDD, CIC e UNSW-NB15, oppure su dataset derivanti da log e statistiche ricavate tramite Openflow. Il secondo approccio si è rivelato particolarmente promettente per permettere l'integrazione real-time, essendo tutte le feature estratte direttamente disponibili all'interno del controller senza necessitare di una deep packet inspection costosa.

Algoritmi tabellari classici (Random Forest, Gradient Boosting, SVM) restano baseline robuste. [9] mostra Gradient Boosting con F1 ≈ 0.998 su UNSW-NB15 per un IDS in SDN, evidenziando la bontà degli ensemble su feature non lineari. Il lavoro dimostra che modelli relativamente semplici possono raggiungere performance eccellenti quando applicati a feature opportunamente ingegnerizzate — un insight che ha guidato la nostra selezione algoritmica.

Sul fronte *deep*, modelli ibridi *CNN+LSTM* sono in grado di catturare pattern spazio—temporali delle serie di flusso. [14] riporta accuratezze $\geq 99\%$ per classificazione binaria e multiclasse con pipeline di rilevamento e mitigazione online. Tuttavia, la complessità computazionale di questi approcci li rende meno attraenti per deployment in controller con risorse limitate.

[2] sottolinea inoltre che l'adozione di feature *controller-based* permette di evitare i costi per la DPI mantenendo, così, ottimi compromessi tra la latenza e la qualità della predizione. Anche in questo caso queste osservazioni hanno influenzato in modo significativo il nostro sistema, motivo per il quale, l'estrazione delle feature avviene privilegiando metriche aggregate facilmente calcolabili dal controller.

3.4 Soluzioni honeypot-based e architetture scalabili

L'utilizzo di soluzioni SDN e ML integrate con honeypot apre interessanti opportunità: attirare il traffico prevalentemente malevolo in modo da ridurre i falsi positivi, studiare tale traffico in modo da comprenderlo ed apprendere nuove firme e riconfigurare la rete in maniera dinamica. Questa strategia si allinea in modo perfetto con il concetto di sinkholing implementato nel nostro progetto.

[4] propone *S-Pot*, framework open-source che utilizza honeypot, IDPS e ML per distribuire in modo dinamico, tramite SDN, le regole all'interno network reale. L'obiettivo è quello di ridurre quanto più possibile FN/FP e gestire anche attacchi *zero-day* tramite regole sempre aggiornate. Questo framework rappresenta un approccio molto ambizioso ma alquanto complesso che richiede ottime infrastrutture dedicate.

[18] integra Ryu con Suricata/MHN e un classificatore semi-supervisionato. Quando viene rilevato un attacco il controller installa, in modo totalmente autonomo, `FLOW_MOD` di mitigazione con tempo medio di installazione $\sim 40\,\mathrm{s}$ nel loro testbed. La latenza, seppur accettabile per la maggior parte dei casi, suggerisce margini di miglioramento nell'implementazione di tutta la catena detection-to-mitigation.

Durante lo sviluppo del nostro sistema, abbiamo osservato che un approccio honeypot semplificato — dove il traffico sospetto viene reindirizzato verso un host isolato per analisi — offre molti dei benefici teorici senza la complessità infrastrutturale di soluzioni complete come S-Pot.

3.5 Confronto tra approcci in letteratura

In sintesi: (i) l'analisi nel controller offre una vista globale utile alla correlazione, ma può diventare un collo di bottiglia sotto attacchi volumetrici; (ii) il pre-filtraggio distribuito riduce la pressione sul control plane a fronte di maggiore complessità di coordinamento; (iii) l'uso di honeypot come valvola di sfogo/analisi migliora l'*operability* dei sistemi rispetto al drop immediato, specialmente in presenza di falsi positivi.

Lavoro	Dove rileva	Tecnica ML/ Heuristica	Mitigazione	Dati/ Feature
[2]	Controller (Ryu)	Ensemble; LGBM migliore su confronto metodi	Regole dinamiche installate in Ryu (`FLOW_MOD`)	Feature *controller-based* da log/statistiche OpenFlow
[16]	N/A (survey)	Tassonomia approcci (firme, anomalia, MTD, honeypot)	Filtraggio, rate limiting, traceback, MTD, honeypot (analisi comparativa)	Classificazione per scenario SDN/IoT, posizione del rilevamento e strategia
[4]	Honeypot + SDN (framework)	Classificatori tradizionali (es. RF/J48/SMO/BN) su log honeypot/IDPS	Generazione e distribuzione dinamica di regole a rete reale via SDN	Log IDPS/honeypot $\rightarrow$ nuove regole; uso DMZ S-Pot
[18]	Honeypot (Suricata/MHN) + Controller (Ryu)	Semi-supervised (es. SVM+AdaBoost)	Installazione `FLOW_MOD` di mitigazione via REST; $t_{\text{install}} \sim 40\,\text{s}$ (testbed)	Alert/log Suricata come sorgente etichette/feature
[14]	Pipeline online (controller/edge)	Ibrido *CNN+LSTM* per pattern spazio—temporali	Regole + IP traceback (schema di risposta reattivo)	Dataset CIC/SDN; classificazione binaria e multiclasse con $\geq 99\%$ acc.
[9]	Contesto SDN (valutazione su dataset)	Gradient Boosting (GBDT); baseline tabellare ad alte prestazioni	Focus su detection (mitigazione non dettagliata)	UNSW-NB15; F1 ≈ 0.998 (valutazione sperimentale)

Tabella 3.1: Confronto sintetico di approcci rappresentativi per rilevamento/mitigazione DDoS in SDN.

3.6 Gradient Boosting per il rilevamento di intrusioni

Nella letteratura recente è emersa un'area particolarmente interessante: l'applicazione di algoritmi ensemble, come Gradient Boosting, per la detection in ambienti SDN.[9] dimostra che Gradient Boosting è in gradi di raggiungere F1-score di 99.87% su dataset UNSW-NB15, superando in modo significativo altri algoritmi classici, come ad esempio Random Forest (99.38%) e Decision Tree (99.24%).

Ciò deriva dalla sua capacità di combinare weak learners in sequenza, in modo tale da permettere ad ogni modello successivo di concentrarsi sugli errori del precedente, tale caratteristica risulta essere particolarmente efficace per i dataset di traffico di rete perchè i patter complessi e non lineari necessitano di approcci sofisticadi di modeling.

3.7 Integrazione real-time e considerazioni implementative

Un aspetto spesso trascurato nei paper accademici riguarda l'integrazione pratica dei modelli ML nei controller SDN. [11] presenta un'implementazione SVM-based su Ryu con focus su reduction del 36% degli effetti DDoS, ma non fornisce dettagli sufficienti sulla pipeline real-time.

Le nostre osservazioni sperimentali suggeriscono che la latenza end-to-end (dalla ricezione del `PacketIn` all'installazione della regola di mitigazione) costituisce un parametro critico spesso sottovalutato. Un sistema che richiede diversi secondi per reagire potrebbe risultare inefficace contro attacchi flash crowd.

L'approccio ibrido che emerge come più promettente prevede: feature extraction leggera nel controller, inferenza ML tramite servizio esterno (Flask/REST), e azione di mitigazione immediata tramite flow rule installation o traffic redirection.

3.8 Sintesi critica e posizionamento del progetto

La maggior parte delle soluzioni centralizzano tutto nel controllore o, totalmente all'opposto, scaricano tutto il grosso del lavoro agli edge. Tutti i framework honeypot-based più completi necessitano di infrastrutture molto articolate, inoltre, i risultati su dataset generici vanno tradotti con cura in feature *controller-based* realmente estraibili in tempo reale [2], [4], [16].

Una mancanza abbastanza evidente emersa dalla letteratura riguarda la mancaza di studi che combinino la semplicità implementativa con l'efficacia operativa: molti lavori molti lavori si focalizzano su accuracy elevate in contesti sperimentali, ma trascurano aspetti pratici come deployment complexity, resource consumption e maintainability.

Contributi di questa tesi rispetto allo stato dell'arte. Il prototipo sviluppato adotta una pipeline *end-to-end* detection→mitigation in Ryu con tre elementi distintivi: m

1. **Feature extraction "IDS-lite"**: estrazione di feature *controller-based* ottimizzate per bassa latenza, inviate a un servizio Flask che ospita un modello ML pre-addestrato;

2. **Sinkhole analitico**: redirezione selettiva verso un host isolato (*sinkhole/honeypot*) in caso di traffico sospetto, anziché drop immediato, così da abilitare una reiniezione "sanitizzata" del traffico buono ed una potenziale analisi forense;

3. **Mitigazione dinamica**: applicazione di regole OpenFlow per la mitigazione con focus su implementation simplicity e time-to-response minimization.

La combinazione di questi elementi dialoga con [2], [4], [18], ma l'enfasi sul *sinkholing analitico* e sull'implementazione minimale (Ryu→Flask→Ryu) costituisce il focus originale del presente lavoro.

Rispetto ai lavori esistenti, il nostro approccio privilegia praticità e deployability su accuracy teorica massima, riconoscendo che in contesti operativi reali un sistema al 95% di accuracy ma operativamente robusto può risultare più valoroso di uno al 99.9% ma fragile o complesso da mantenere.

Le verifiche sperimentali condotte durante lo sviluppo hanno confermato che questo compromesso è ben bilanciato: il sistema raggiunge performance competitive mantenendo una architettura comprensibile e manutenibile, aspetti cruciali per l'adozione in ambienti production.

Capitolo 4

Progettazione del Sistema Proposto

4.1 Architettura generale del sistema

Il sistema è articolato su quattro componenti principali che, collaborando tra di loro, realizzano una pipeline end-to-end di detection e mitigation. Da un lato il controller ryu orchestra la rete gestendo l'estrazione delle feature e installando le regole di mitigazione mentre, dall'altro, un server Flask ospita il modello di ML pre-addestrato in modo da garantire un'inferenza rapida e scalabile.

La topologia di rete viene emulata tramite Mininet che separa chiaramente due domini: una rete esterna che simula "Internet", dove risiedono gli attaccanti, ed una rete interna protetta che ospita le potenziali vittime. Un router software interconnette i due domini, mentre un nodo dedicato funge da sinkhole analitico per l'analisi del traffico sospetto.

Questa separazione logica serve a riflettere scenari realistici dove un'organizzazione deve essere in grado di proteggere la propria rete interna dal traffico malevolo proveniente da internet. Il controller è stato posizionato in modo strategico consentendo, così, il monitoraggio di tutti i flussi che attraversano i confini di sicurezza e potendo, quindi, implementare controlli basati sulle classficazioni del modello di machine learning.

4.1.1 Flusso operativo di alto livello

Il funzionamento del sistema segue un pattern lineare che inizia con l'arrivo di un pacchetto che non ha riscontri nella tabella di flusso, generando quindi un evento "PacketIn" verso il controller che, analizzando gli header del pacchetto, estrae le feature necessarie per la classificazione.

Esse vengono inviate al servizio Flask tramite una richiesta HTTP POST, restituendo una classificazione binaria (benigno/malevolo). Basandosi su tale risposta il controller decide il forwarding: se il traffico è considerato benigno viene instradato in modo normale, mentre quello sospetto viene reindirizzato al sinkhole per ulteriori analisi.

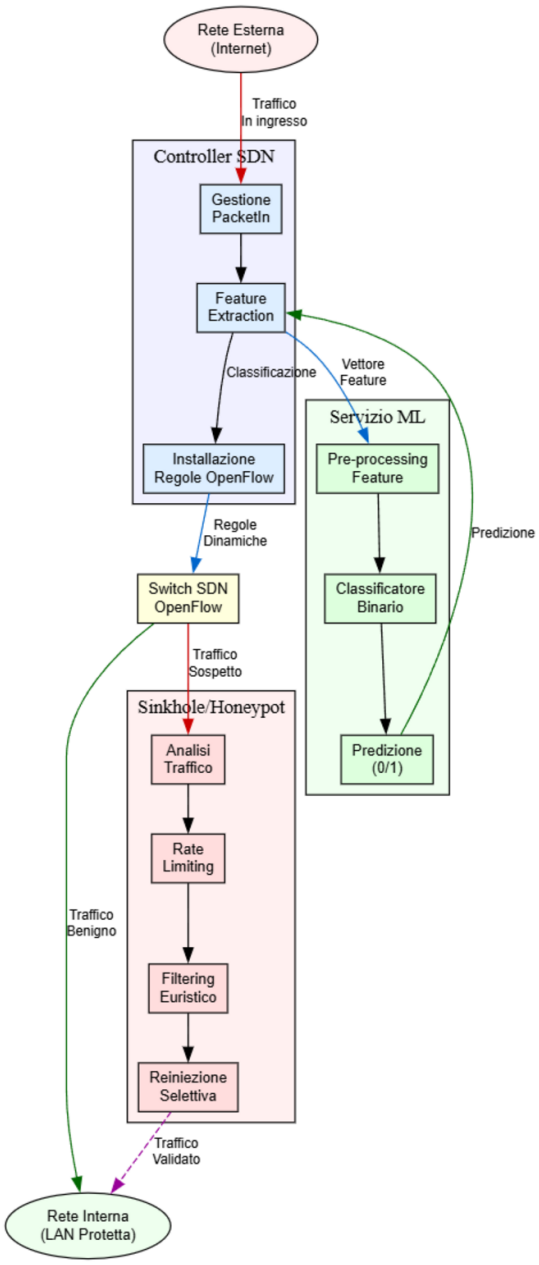

Figura 4.1: Architettura completa del sistema proposto: controller Ryu con feature extraction IDS-lite, server Flask per classificazione ML, honeypot/sinkhole per analisi traffico rediretto. Il flusso operativo mostra la pipeline end-to-end dalla detection alla mitigazione tramite redirezione intelligente.

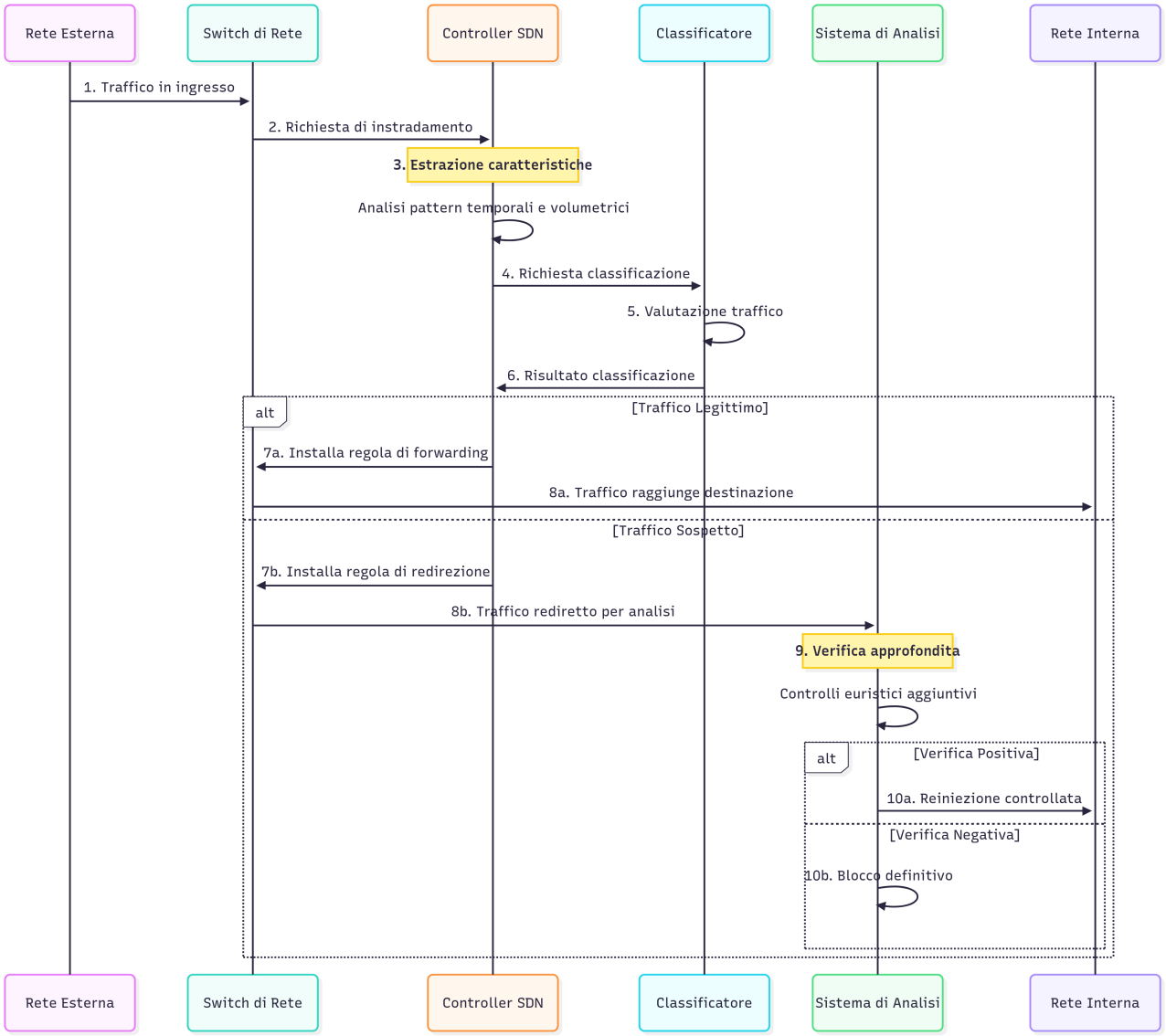

Figura 4.2: Sequence diagram del flusso operativo: dalla ricezione del PacketIn all'eventuale mitigazione, mostrando le interazioni temporali tra controller Ryu, servizio ML Flask, e componenti di rete per la gestione del traffico sospetto.

4.2 Ruolo del controller SDN nel processo di rilevamento

Ryu è il centro nevralgico del sistema che orchestra tutto il processo di rilevamento e risponde in modo dinamico agli eventi di rete.

4.2.1 Gestione degli eventi e estrazione delle feature

Ogni evento PacketIn innesca una sequenza di operazioni a partire dal parsing dei protocolli di rete tramite le primitive di "ryu.lib.packte" in seguito il controller identifica i layer protocol (IP, TCP, UDP) ed estrae tutte le informazioni utilili (Indirizzi sorgente e destinazione, porte, TTL e flag vari).

Non è da trascurare il fatto che questa estrazione non si limita ad analizzare il singolo pacchetto ma tiene conto di un contesto temporale attraverso delle sliding windows: il sistema tiene in struttura dati con i quali traccia il comportamento dei flussi con intervalli da 1 secondo e 60 secondi tramite i quali calcola delle metriche aggregate come il rate di pacchetti per sorgente, il carico in byte per destinazione e il numero di connessioni verso specifici servizi.

4.2.2 Interfacciamento con il motore di machine learning

Dopo aver estratto le feature, il controller costruisce un vettore numerico che combacia con ciò che il modello di ML si aspetta. Questa serializzazione richiede, per quanto riguarda la consistenza dei tipi di dato e all'ordine delle feature, particolare attenzione per far sì che il modello riceva in input i dati nella forma prevista.

Il servizio Flask viene raggiunto tramite HTTP POST sincrone con timeout configurati ad 1 secondo per evitare che latenze eccessive del modello impattino troppo sulla responsività del controller.

La gestione degli errori di comunicazione tra controller e servizio ML segue una strategia conservativa, ossia: in caso di failure del modello il controller assume che il traffico sia benigno e procede con l'instradamento normale.

4.2.3 Installazione dinamica delle regole di forwarding

Le predizioni del modello di machine learning vengono tradotte in azioni concrete per installare le regole dinamiche di OpenFlow: per quanto riguarda il traffico malevolo il controller installa regole ad alta priorità che, intercettando i flussi, li reinderizzano verso la porta del sinkhole.

L'implementazione utilizza timeout idle e hard per garantire che le regole non rimangano indefinitamente nelle tabelle di flusso degli switch.

4.3 Estrazione delle feature in stile UNSW-NB15

L'estrazione delle feature costituisce uno degli aspetti più critici del sistema, dovendo tradurre eventi di rete grezzi in rappresentazioni numeriche significative per il modello di machine learning. L'approccio adottato

si ispira direttamente alla metodologia utilizzata nel dataset UNSW-NB15, adattandola però alle specificità di un ambiente SDN operativo.

4.3.1 Feature basate su rate e volume

Le metriche di rate sono indicatori atti ad identificare i comportamenti anomali all'interno del flusso in entrata: il sistema calcola la frequenza di pacchetti per ogni IP sorgente tenendo in conto dei timestamp ricevut in in sliding window di 1 secondo, identificando così i punti dai quali nascono enormi volumi di traffico.

Allo stesso modo sono monitorate le metriche di carico (sload e dload) che tengono conto dei byte trasmessi e ricevuti.

4.3.2 Feature di contesto temporale

Un'altra categoria interessante di feature è quella che riguarda i contatori che trattano gli eventi su finestre temporali più estese: la feature `ct_srv_src` ha il compito di tener conto delle connessioni provenienti da un host specifico verso un determinato servizio, avvenuto negli ultimi 60 secondo, per poter catturare un pattern di attacco legato a port scanning o più semplicemente legato ad attacchi indirizzati a specifici servizi.

In modo analogo, `ct_dst_ltm` monitora il traffico complessivo verso una destinazione nell'ultimo minuto, evidenziando situazioni di concentrazione anomala che potrebbero indicare attacchi coordinati.

La feature `ct_state_ttl`, invece, correla lo stato di connessione con il valore Time-To-Live, fornendo indicazioni sulla diversità geografica delle sorgenti. Valori di TTL molto variabili tra pacchetti dello stesso flusso possono suggerire spoofing o coordinamento tra botnet distribuite geograficamente.

4.3.3 Indicatori di stato e protocollo

Per quanto riguarda le feature di stato e protocollo state_CON e state_INT sono in grado di fornire informazioni riguardanti lo stato delle connessioni (state_CON indica se è stata stabilita in modo corretto una connessione) e se vi è la presenza di traffico bidirezionale (state_INT distingue tra comunicazioni interattive e a traffico unidirezionale).

Ciò richiede un'analisi dei flag TCP e il controllo dello stato delle connessioni attraverso le finestre temporali.

4.4 Comunicazione tra controller e modello ML su Flask

L'interfaccia tra il controller e il motore di ML è un punto cruciale per l'efficienza del sistema: la scelta di utilizzare flask nasce dalla necessità di avere un'implementazione semplice che garantisca le performance necessarie a tutti gli scenari di traffico moderato.

4.4.1 Architettura del servizio di inferenza

Il servizio Flask implementa un'API REST minimale con un singolo endpoint /predict che accetta richieste POST contenenti vettori di feature in formato JSON.

Il modello di machine learning viene caricato una sola volta all'avvio del servizio utilizzando pickle, evitando overhead di deserializzazione per ogni richiesta.

La gestione degli errori è progettata per essere robusta: input malformati o feature mancanti vengono gestiti assegnando valori di default appropriati, evitando che piccole inconsistenze nella formattazione impediscano la classificazione.

4.4.2 Protocollo di comunicazione e gestione degli errori

Il protocollo di comunicazione segue un pattern request-response sincrono dove ogni richiesta del controller include un vettore completo di feature estratte dal PacketIn corrente. Il servizio risponde con un oggetto JSON contenente la predizione binaria (0 per traffico benigno, 1 per malevolo).

La gestione dei timeout riveste particolare importanza: il controller configura timeout HTTP di 1 secondo per evitare che latenze eccessive del servizio ML impattino sulla responsività di rete.

4.5 Comportamento del sinkhole/honeypot

Il sinkhole è un elemento distintivo all'interno della nostra architettura che implementa un approccio ibrido che combina la capacità di assorbimento del traffico malevolo alla possibilità di consentire un'analisi più raffinata del traffico per permettere, diversamente da sistemi che eseguono un drop immediato, la potenziale reiniezione di comunicazioni erroneamente classificate come malevoli.

4.5.1 Strategia di filtering intelligente

Il sinkhole implementa un sistema di filtering multi-livello che opera analisi euristica sul traffico ricevuto dal controller. Il primo livello applica rate limiting per indirizzo IP sorgente, bloccando sorgenti che superano soglie di pacchetti per secondo configurabili.

Un secondo livello di analisi si concentra specificamente sui pattern TCP, implementando protezioni anti-SYN flood che monitorano il rapporto tra pacchetti SYN ricevuti e connessioni completate. Sorgenti che generano volumi elevati di richieste di connessione senza completare l'handshake vengono identificate e bloccate automaticamente.

4.5.2 Meccanismo di reiniezione selettiva

Una caratteristica innovativa del sinkhole è la capacità di reintrodurre nella rete traffico che supera i controlli euristici locali. Questa funzionalità è cruciale per minimizzare l'impatto di falsi positivi del classificatore

ML, consentendo a comunicazioni legittime erroneamente intercettate di raggiungere le loro destinazioni originali.

Il processo di reiniezione utilizza le primitive Scapy per ricostruire e ritrasmettre pacchetti modificando minimamente gli header originali.

Il sinkhole mantiene statistiche dettagliate sul traffico processato, fornendo visibilità operativa sui pattern di attacco e sull'efficacia delle strategie di filtering.

4.6 Strategie di reinstradamento e mitigazione

Le strategie di mitigazione implementate dal sistema operano su due livelli complementari: azioni immediate di reinstradamento del traffico sospetto e installazione di regole proattive per prevenire l'escalation degli attacchi. Questo approccio dual-mode consente di bilanciare reattività e efficienza delle risorse di rete.

4.6.1 Mitigazione reattiva tramite flow redirection

La strategia primaria di mitigazione si basa sul reindirizzamento dinamico di flussi specifici verso il sinkhole non appena vengono classificati come malevoli. Il controller installa regole OpenFlow ad alta priorità che intercettano traffico matching la tupla (IP sorgente, IP destinazione) e lo dirigono verso la porta del sinkhole invece che verso la destinazione originale.

Quest'approccio presenta diversi vantaggi rispetto al dropping immediato: il traffico rimane visibile per analisi, connessioni erroneamente classificate possono essere recuperate attraverso la reiniezione, e gli attaccanti non ricevono immediate indicazioni che i loro attacchi sono stati rilevati.

L'implementazione gestisce correttamente situazioni complesse come flussi bidirezionali e inversioni di ruolo attaccante-vittima. Il sistema analizza la topologia di rete per determinare quale endpoint rappresenta probabilmente l'attaccante e installa regole che intercettano specificamente il traffico malevolo senza impattare comunicazioni di ritorno legittime.

4.6.2 Metriche di efficacia e monitoring

Il sistema implementa telemetria estensiva per monitorare l'efficacia delle strategie di mitigazione e identificare opportunità di ottimizzazione. Metriche come tempo medio di mitigazione (dalla detection all'installazione della regola) forniscono indicazioni sulla reattività del sistema e identificano potenziali bottleneck.

Il tracking di flussi malevoli include contatori che distinguono tra prime occorrenze (che triggherano installazione di nuove regole) e rioccorrenze (che incrementano contatori esistenti).

Capitolo 5

Implementazione

Il seguente capitolo tratta l'implementazione dell'architettura progettata: vengono mostrate le scelte implementative per quanto riguarda la topologia, gli indirizzi di rete e la configurazione degli host, come avviene la comunicazione tra i vari moduli e la fase di rilevamento e mitigazione.

5.1 Topologia di rete e configurazione in Mininet

La topologia implementata in `topo_ddos_router.py` realizza una separazione netta tra rete "Internet" (simulata) e LAN interna attraverso un router software.

```
class DDOSRouterTopo(Topo):
   def build(self):
       # SWITCH ESTERNO (Internet)
       s1 = self.addSwitch('s1')
       # SWITCH INTERNO (LAN)
       s2 = self.addSwitch('s2')

       # ROUTER
       r1 = self.addHost('r1')

       # Collegamenti router-switch
       self.addLink(r1, s1)  # r1-eth0 (verso Internet)
       self.addLink(r1, s2)  # r1-eth1 (verso LAN)
```

Il router `r1` interconnette due domini di rete distinti: `s1` ospita gli attaccanti simulati (`h1`, `h2`, `h3`) nella rete 10.0.2.0/24, mentre `s2` contiene la vittima (`h4`) e il sinkhole/honeypot (`h5`) nella rete 10.0.1.0/24.

Lo script di inizializzazione `setupNetworkMet.py` configura automaticamente gli indirizzi IP e abilita l'inoltro sul router:

```
# Router IP e routing abilitato
```

```
r1.cmd("ifconfig r1-eth0 10.0.2.254/24")  # Verso attaccanti (Internet)
r1.cmd("ifconfig r1-eth1 10.0.1.254/24")  # Verso vittime (LAN interna)
r1.cmd("echo 1 > /proc/sys/net/ipv4/ip_forward")

# Configura attaccanti (rete 10.0.2.0/24 - Internet)
for idx, h in enumerate([h1, h2, h3], start=1):
   h.setIP(f"10.0.2.{idx+10}/24")
   h.setDefaultRoute("via 10.0.2.254")

# Configura vittime (rete 10.0.1.0/24 - LAN interna)
h4.setIP("10.0.1.10/24")  # Vittima principale
h5.setIP("10.0.1.11/24")  # Honeypot
```

L'honeypot viene avviato automaticamente in background durante la fase di setup, mentre il controller Ryu può essere lanciato separatamente su una macchina diversa (nel nostro caso è un altro terminale mininet) per simulare una configurazione distribuita reale.

5.2 Controller Ryu: IDS-lite e gestione degli eventi

Il modulo `idsMet.py` implementa un IDS "leggero" che estende la classe base di Ryu per intercettare eventi OpenFlow e applicare logiche di sicurezza. Il controller mantiene strutture dati per il tracking temporale dei flussi e calcola feature aggregate in tempo reale.

5.2.1 Architettura del controller e inizializzazione

All'avvio, il controller registra i datapath degli switch e installa regole di base che dirigono tutti i pacchetti senza match verso il controller stesso:

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
   datapath = ev.msg.datapath
   self.datapaths[datapath.id] = datapath

   match = parser.OFPMatch()
   actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                     ofproto.OFPCML_NO_BUFFER)]
   self.add_flow(datapath, 0, match, actions)
```

Il sistema identifica automaticamente gli switch interni (`INTERNAL_DPID = 2`) dove installare le regole di mitigazione, mentre la porta del sinkhole è configurata staticamente (`SINKHOLE_PORT_S2 = 3`).

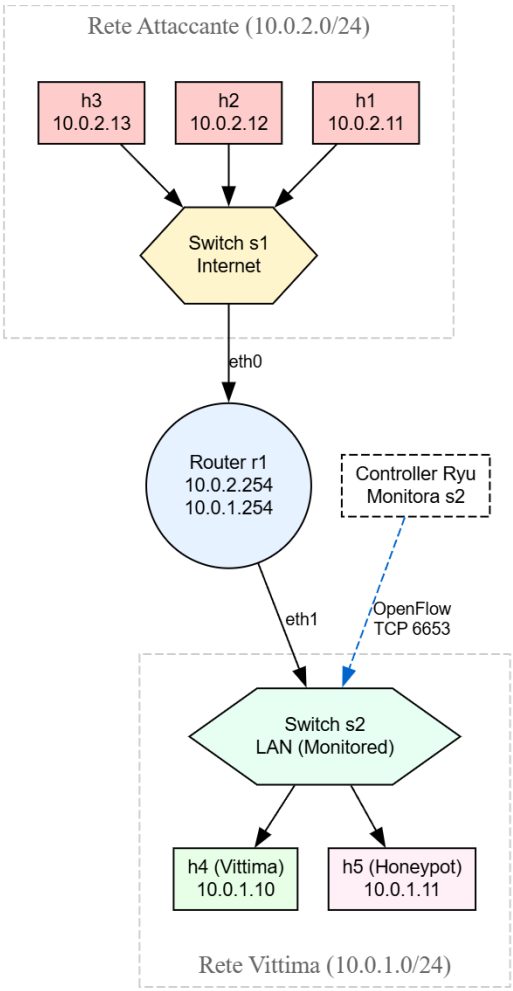

Figura 5.1: Topologia di rete implementata in Mininet: rete attaccanti (10.0.2.0/24) con host h1-h3 connessi a switch s1, rete vittime (10.0.1.0/24) con vittima h4 e honeypot h5 su switch s2, interconnesse dal router r1. Il controller Ryu monitora s2 per il rilevamento DDoS.

5.2.2 Estrazione delle feature e analisi temporale

Per ogni evento `PacketIn`, il controller estrae informazioni dai protocolli di rete e aggiorna contatori su finestre temporali multiple. Le feature implementate seguono la metodologia UNSW-NB15 adattata al contesto SDN:

```
# Feature di rate (finestra 1 secondo)
self.packet_timestamps.setdefault(src_ip, []).append(now)
self.packet_timestamps[src_ip] = [t for t in self.packet_timestamps[src_ip]
                                  if now - t <= 1]
rate = len(self.packet_timestamps[src_ip])

# Carico in byte per sorgente e destinazione
sload = sum(b for t, b in self.byte_timestamps[src_ip] if now - t <= 1)
dload = sum(b for t, b in self.byte_timestamps[dst_ip] if now - t <= 1)
```

5.2.3 Comunicazione con il servizio ML e decisioni di mitigazione

Il vettore di feature viene serializzato e inviato al server Flask tramite richieste HTTP POST sincrone con timeout di 1 secondo:

```
features = {
   'rate': rate, 'sttl': ttl, 'sload': round(sload, 2),
   'dload': round(dload, 2),
   'ct_srv_src': len(self.ct_srv_src_count.get((src_ip, dport), [])),
   'ct_dst_ltm': len(self.ct_dst_ltm.get(dst_ip, [])),
   # ... altre feature
}

res = requests.post(self.server_url, json=features, timeout=1)
malicious = (res.json().get("prediction") == 1)
```

Quando un flusso viene classificato come malevolo, il controller identifica automaticamente attaccante e vittima basandosi sulle reti di appartenenza e installa regole di redirezione ad alta priorità:

```
if malicious and not self.malicious_flows[flow_key]["rule_installed"]:
   match = p.OFPMatch(eth_type=0x0800, ipv4_src=attacker_ip, ipv4_dst=victim_ip)
   actions = [p.OFPActionOutput(SINKHOLE_PORT_S2)]
   mod = p.OFPFlowMod(datapath=s2, priority=100, match=match,
                      instructions=[...], idle_timeout=10, hard_timeout=20)
```

5.3 Servizio Flask per l'inferenza ML

Il server `serverMet.py` implementa un'API REST minimale che ospita un modello Decision Tree pre-addestrato. Il servizio carica il modello una sola volta all'avvio utilizzando pickle.

```
with open("decision_tree_binary.pkl", "rb") as f:
   model = pickle.load(f)

expected_features = [
   'rate', 'sttl', 'sload', 'dload', 'ct_srv_src', 'ct_state_ttl',
   'ct_dst_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm',
   'ct_dst_src_ltm', 'ct_src_ltm', 'ct_srv_dst',
   'state_CON', 'state_INT'
]
```

L'endpoint `/predict` accetta vettori di feature in formato JSON e restituisce predizioni binarie.

Il server mantiene metriche operative semplici per il monitoraggio delle prestazioni, registrando il numero di richieste al secondo elaborate.

5.4 Honeypot/sinkhole e logiche di filtraggio

Lo script `honeypotMet.py` implementa un sinkhole analitico che combina capacità di assorbimento del traffico malevolo con logiche di reiniezione selettiva. Utilizzando Scapy per lo sniffing dei pacchetti, l'honeypot applica euristiche leggere per distinguere traffico legittimo da quello chiaramente malevolo.

```
def is_legit(pkt):
   if IP not in pkt:
       return False

   src = pkt[IP].src
   now = time.time()

   # Rate limit semplice: max 20 pkt/s per sorgente
   RATE_LIMIT.setdefault(src, []).append(now)
   RATE_LIMIT[src] = [t for t in RATE_LIMIT[src] if now - t < 1]
   if len(RATE_LIMIT[src]) > 20:
       return False

   # SYN sporadici ok, flood >20/s blocca
   if TCP in pkt and pkt[TCP].flags == "S":
       SYN_RATE.setdefault(src, []).append(now)
```

```
        SYN_RATE[src] = [t for t in SYN_RATE[src] if now - t < 1]
        if len(SYN_RATE[src]) > 20:
            return False
```

La strategia di filtraggio è volutamente conservativa: utilizza pochi segnali chiari come rate limiting per indirizzo IP sorgente, protezioni anti-SYN flood e filtri su UDP verso porte alte. Traffico che supera questi controlli viene reiniettato verso la vittima originale, mentre quello sospetto viene scartato.

Il meccanismo di reiniezione utilizza le primitive Scapy per ricostruire e ritrasmettere pacchetti, consentendo a comunicazioni legittime erroneamente classificate dal modello ML di raggiungere comunque le loro destinazioni:

```
def process(pkt):
    if is_legit(pkt):
        sendp(pkt, iface="h5-eth0", verbose=False)
        metrics["reinjected_window"] += 1
    else:
        metrics["blocked_window"] += 1
```

5.5 Metriche operative e monitoraggio

Il sistema implementa telemetria estensiva per monitorare l'efficacia delle strategie di mitigazione. Tutti i componenti producono log coordinati che permettono di ricostruire l'intera catena decisionale.

Il controller Ryu traccia metriche chiave come pacchetti in ingresso al secondo, richieste ML al secondo, latenza delle chiamate al modello e delay di mitigazione dal primo rilevamento all'installazione della regola:

```
self.logger.info(
    "[RYU][%0.1fs] PktIn/s=%.1f  ML req/s=%.1f  ML lat(ms) avg/min/max=%.1f/%.1f/%.1f
    "MitigDelay(ms) avg=%.1f last=%.1f  Active malicious flows=%d",
    interval, pktin_s, req_s, avg_ms, min_ms, max_ms, md_avg, md_last,
    len(self.malicious_flows)
)
```

L'honeypot mantiene contatori di pacchetti reinettati e bloccati, mentre il server Flask registra il throughput di richieste elaborate.

5.6 Workflow operativo e integrazione end-to-end

Il percorso completo di un pacchetto attraverso il sistema illustra l'integrazione tra tutti i componenti:

1. Un host attaccante (`h1`/`h2`/`h3`) nella rete 10.0.2.0/24 invia traffico verso la vittima `h4` (10.0.1.10)
2. Lo switch `s2` inoltra il primo pacchetto al controller Ryu tramite `PacketIn`

3. Il controller estrae feature temporali e invia il vettore al servizio Flask
4. Il modello ML restituisce una classificazione binaria
5. Se malevolo, viene installata una regola che redirige il traffico verso `h5` (porta 3)
6. L'honeypot applica filtraggio euristico e decide se reiniettare o scartare

Il sistema mantiene uno stato coerente anche sotto carico moderato, mentre la modularità dell'architettura permette di aggiornare il modello ML o modificare le euristiche dell'honeypot senza impattare il funzionamento del controller.

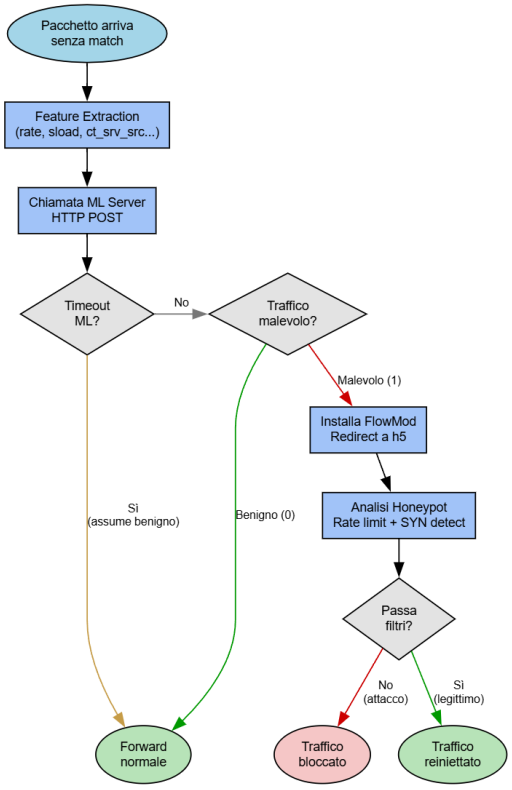

Figura 5.2: Workflow operativo end-to-end: dal PacketIn all'eventuale mitigazione, con feature extraction, classificazione ML, decisione di instradamento e analisi honeypot per traffico sospetto.

5.7 Generazione del traffico di test

Per la validazione del sistema abbiamo utilizzato strumenti standard disponibili in Mininet, generando traffico direttamente dalla CLI interattiva. Traffico benigno viene simulato con comandi come:

```
h4 python3 -m http.server 8080 &
h1 while true; do curl -s http://10.0.1.10:8080 > /dev/null; sleep 10; done &
```

Mentre gli attacchi DDoS sono generati con hping3 configurato per produrre flooding:

```
h2 hping3 --flood --udp -p 80 10.0.1.10
h3 hping3 --flood -S -p 80 10.0.1.10
```

Questo approccio ci ha permesso di testare sistematicamente le finestre temporali del controller, osservare l'efficacia del classificatore ML e verificare il comportamento del sinkholing attraverso i log coordinati di tutti i componenti.

L'implementazione riflette l'obiettivo della tesi: dimostrare che in un ambiente SDN è possibile concatenare rilevamento intelligente e mitigazione dinamica mantenendo il sistema modulare, osservabile e operativamente pratico con strumenti standard della comunità SDN.

Capitolo 6

Valutazione Sperimentale

L'implementazione descritta nel capitolo precedente ha portato alla realizzazione di un sistema completo che integra controllo SDN, classificazione ML e mitigazione intelligente. Tuttavia, la validazione sperimentale rimane cruciale per dimostrare l'efficacia operativa della soluzione proposta.

La valutazione deve necessariamente considerare molteplici dimensioni: dall'accuratezza del rilevamento alla tempestività della risposta, dalla stabilità sotto carico all'impatto sul traffico legittimo. Solo attraverso test sistematici è possibile verificare che l'integrazione tra componenti eterogenei non comprometta le prestazioni complessive del sistema.

6.1 Obiettivi e metodologia

L'obiettivo principale di questi test è quello di poter quantificare le capacità del sistema in termini di accuratezza in quelle che sono : la rilevazione, la tempestività nelle risposte e nella sostenibilità operativa sotto carico, questo perché valutare l'efficacia di un sistema di difesa contro attacchi DDoS necessità di un approccio molto metodico che deve tener conto sia dell'efficacia nella mitigazione che dell'impatto sulle prestazioni complessive della rete.

Da un lato, bisogna verificare che l'integrazione tra il modello di ML e la redirezione SDN possieda elevate prestazioni di rilevazione, senza andar a introdurre delle latenze o falsi positivi che potrebbero compromettere il traffico legittimo. Dall'altro, è necessario valutare la robustezza del sistema quando sottoposto a volumi intensi di traffico, caratteristici degli attacchi reali.

6.2 Metriche (accuratezza, tempi, carico)

La valutazione del sistema è articolata su diverse dimensioni dell'analisi, ognuna delle quali fornisce dettagli complementari sull'efficacia della soluzione proposta. Le metriche selezionate rispecchiano sia gli aspetti prestazionali che quelli operativi del sistema.

6.2.1 Metriche di Performance del Sistema ML

Il server di classificazione rappresenta il cuore dell'intelligence del sistema. Per questo motivo, le metriche principali includono il throughput delle richieste di classificazione, misurato in richieste per secondo (req/s), e la latenza di risposta del modello. Quest'ultima viene tracciata attraverso valori medi, minimi e massimi per ogni finestra temporale, permettendo di identificare eventuali picchi di latenza che potrebbero compromettere la reattività del sistema.

La distribuzione delle latenze fornisce informazioni cruciali sulla stabilità del sistema: valori medi contenuti ma con picchi elevati potrebbero indicare problemi di scalabilità o contesa delle risorse, mentre una distribuzione uniforme suggerisce un comportamento prevedibile e stabile.

6.2.2 Metriche di Controllo del Traffico

Il controller SDN monitora continuamente il volume di PacketIn ricevuti (50.0 PacketIn/s durante i test), fornendo una misura diretta del carico computazionale sostenuto. Questa metrica, espressa anch'essa in eventi per secondo, consente di correlare il throughput del traffico con l'utilizzo delle risorse di controllo.

Particolarmente significativo è il *mitigation delay*, definito come l'intervallo temporale tra la prima rilevazione di un flusso malevolo e l'installazione della regola di redirezione corrispondente. Questo parametro rappresenta una misura diretta dell'efficacia temporale del sistema: valori ridotti indicano una capacità di risposta rapida, fondamentale per limitare l'impatto degli attacchi.

6.2.3 Metriche dell'Honeypot

Il componente honeypot fornisce visibilità sulle dinamiche di filtraggio del traffico rediretto. Le metriche principali includono il rate di reindirizzamento verso le vittime per il traffico considerato legittimo e il rate di blocco per quello identificato come malevolo.

Queste metriche permettono di valutare l'efficacia delle euristiche implementate nell'honeypot e di identificare eventuali criticità nel processo di distinzione tra traffico legitimo e attacchi. Un elevato numero di falsi positivi (traffico legittimo bloccato) indicherebbe la necessità di raffinare gli algoritmi di filtraggio.

6.2.4 Indicatori di Soglia e Allarme

Il sistema integra meccanismi di soglia per il rilevamento di condizioni anomale, come il rate limiting per IP sorgente e la detection di SYN flooding. Il monitoraggio di questi indicatori fornisce informazioni sulla distribuzione e l'intensità degli attacchi, oltre che sull'efficacia delle contromisure implementate.

6.3 Scenario di test e configurazione

L'ambiente sperimentale è stato progettato per riflettere una tipica architettura enterprise, caratterizzata dalla separazione tra rete esterna (Internet) e rete interna (LAN), mediata da componenti di routing e sicurezza.

6.3.1 Topologia di Rete

La topologia implementata simula un'infrastruttura a due segmenti: la rete esterna (subnet 10.0.2.0/24) ospita i potenziali attaccanti, mentre la rete interna (subnet 10.0.1.0/24) contiene le risorse da proteggere. Un router software (r1) gestisce l'instradamento tra i due domini, mentre due switch OpenFlow (s1 e s2) forniscono la connettività locale e l'interfacciamento con il controller SDN.

Gli host h1, h2 e h3 rappresentano sistemi esterni potenzialmente compromessi, configurati con indirizzi nella rete degli attaccanti. L'host h4 simula il server vittima degli attacchi, mentre h5 implementa il nodo honeypot. Questa configurazione permette di testare scenari realistici in cui il traffico malevolo deve attraversare il router per raggiungere le vittime, passando sotto il controllo del sistema SDN.

6.3.2 Configurazione del Controller

Il controller Ryu è stato configurato per operare come dispatcher di flussi, intercettando tutti i pacchetti in transito attraverso lo switch interno s2. Questa posizione strategica garantisce la visibilità completa del traffico diretto alle risorse protette, permettendo l'analisi e la classificazione prima che raggiunga le destinazioni finali.

Il sistema di feature extraction opera in tempo reale, mantenendo finestre temporali scorrevoli per il calcolo delle statistiche di traffico. Le finestre di osservazione sono configurate a 1 secondo per le metriche di rate e 60 secondi per i contatori di connessione, bilanciando la reattività del sistema con la stabilità delle misurazioni.

6.3.3 Modello di Machine Learning

Il classificatore utilizzato è un Decision Tree binario, addestrato sul dataset UNSW-NB15 per distinguere traffico benigno da attacchi di tipo DoS/DDoS. La scelta di questo algoritmo è motivata dalla necessità di garantire tempi di inferenza ridotti e interpretabilità delle decisioni, aspetti cruciali in un contesto operativo real-time.

Il vettore di feature estratto per ogni flusso include 14 attributi, spaziando da statistiche temporali (rate di pacchetti) a indicatori di connessione (contatori di servizio e stato). Questa selezione riflette un compromesso tra completezza dell'informazione e efficienza computazionale.

6.3.4 Scenari di Traffico

Gli esperimenti contemplano diversi profili di traffico per valutare la robustezza del sistema in condizioni operative variegate. Gli scenari includono:

- **Traffico baseline**: comunicazioni HTTP normali simulate avviando un web server sulla vittima (h4 python3 -m http.server 8080 &) e generando richieste continue dall'attaccante (h1 while true; do curl -s http://10.0.1.10:8080 > /dev/null; sleep 10; done &).

- **Attacchi UDP flood**: generati mediante hping3 con targeting su porte specifiche del server vittima. Questi attacchi testano la capacità del sistema di rilevare pattern volumetrici anomali e di reagire tempestivamente.
- **Attacchi TCP SYN flood**: mirati all'esaurimento delle risorse di connessione del target. Questo scenario verifica l'efficacia delle euristiche anti-flooding implementate nell'honeypot.
- **Traffico misto**: combinazione di comunicazioni legittime e attacchi simultanei, per valutare la capacità del sistema di mantenere la qualità del servizio per gli utenti autorizzati durante gli eventi di sicurezza.

6.3.5 Configurazione dell'Honeypot

Il nodo honeypot è configurato con meccanismi di rate limiting per IP sorgente (soglia di 20 pacchetti/secondo) e detection di SYN flooding (soglia di 20 connessioni/secondo). Il sistema implementa inoltre una whitelist per indirizzi considerati affidabili, permettendo di escludere dalla mitigazione il traffico proveniente da sorgenti note.

Il processo di analisi del traffico rediretto utilizza euristiche basate su pattern di comportamento: traffico verso porte note e con frequenze moderate viene re-instradato verso le vittime, mentre pattern anomali vengono scartati. Questa logica permette di minimizzare l'impatto sui servizi legittimi pur mantenendo efficace la protezione.

6.4 Risultati e discussione

L'analisi dei risultati sperimentali evidenzia prestazioni complessivamente positive del sistema proposto, con alcuni aspetti che meritano particolare attenzione per una valutazione completa dell'efficacia della soluzione.

6.4.1 Performance del Sistema di Classificazione

Il server di machine learning dimostra capacità di throughput adeguate alle esigenze operative, gestendo un carico di `50` richieste per secondo con latenze medie dell'ordine di `8.8` millisecondi. La distribuzione delle latenze risulta contenuta, con valori massimi che non superano i `20` millisecondi anche durante i picchi di traffico più intensi.

Questi risultati confermano la fattibilità dell'approccio basato su server dedicato per la classificazione, evitando l'overhead computazionale sul controller SDN. Non va trascurato che la stabilità delle prestazioni durante l'intera durata degli esperimenti suggerisce una buona scalabilità del sistema, almeno nelle condizioni di carico testate.

6.4.2 Efficacia della Mitigazione

Il sistema dimostra eccellente capacità nella rilevazione e mitigazione degli attacchi DDoS, con rilevazioni consistenti sui pattern di traffico malevolo generati. Il mitigation delay medio si attesta sui `16.2` millisecondi

Particolarmente significativo è il comportamento del sistema durante gli attacchi: l'honeypot intercetta efficacemente il traffico rediretto, con un rate di blocco di `17.17` pacchetti/secondo e reiniezione di `6.19` pacchetti/secondo verso le vittime legittime. Nel complesso, dei pacchetti processati dall'honeypot, il 30.1% sono stati reiniettati come legittimi mentre il 69.9% sono stati bloccati come malevoli.

6.4.3 Impatto sul Traffico Legittimo

Una delle preoccupazioni principali in sistemi di difesa automatica è l'impatto sui servizi regolari. Gli esperimenti mostrano che il traffico benigno subisce un overhead trascurabile: le comunicazioni normali mantengono latenze comparabili a quelle misurate prima dell'attivazione del sistema di difesa.

Il sistema dimostra buona capacità di distinzione: l'honeypot reinietta verso le vittime circa il 30% del traffico rediretto, indicando che una porzione significativa di comunicazioni inizialmente sospette viene recuperata attraverso l'analisi euristica secondaria. Questi eventi risultano inoltre rapidamente corretti dal processo di reindirizzamento dell'honeypot, minimizzando l'impatto percepito dagli utenti finali.

6.4.4 Analisi delle Soglie e Tuning

I meccanismi di soglia implementati mostrano efficacia variabile a seconda del tipo di attacco. Le soglie per il rate limiting di pacchetti (20 pkt/sec per IP) risultano appropriate per la maggior parte degli scenari testati, mentre la soglia SYN flood (20 connessioni/sec) richiede calibrazione più fine per evitare interferenze con applicazioni ad alta concorrenza.

L'analisi dei pattern di traffico durante gli attacchi rivela comportamenti caratteristici che potrebbero essere sfruttati per ottimizzare ulteriormente gli algoritmi di detection. Gli attacchi UDP flood mostrano tipicamente burst iniziali ad alta intensità seguiti da traffic patterns più regolari, suggerendo la possibilità di implementare detection adattive basate su finestre temporali multiple.

6.4.5 Stabilità e Robustezza

Il sistema mantiene stabilità operativa durante l'intera durata dei test, senza evidenziare degradazioni prestazionali o accumulo di errori. Il controller SDN gestisce efficacemente l'installazione e la rimozione delle regole di flusso, mantenendo le tabelle di forwarding in uno stato coerente anche durante i picchi di attività.

L'honeypot dimostra robustezza nella gestione del traffico ad alto volume, processando complessivamente `23.36` pacchetti per secondo (`17.17` blocked/s + `6.19` reinjected/s) senza perdite significative o rallentamenti.Questo risultato è particolarmente rilevante considerando che il nodo honeypot rappresenta un potenziale collo di bottiglia per tutto il traffico rediretto.

Lavoro	Dove rileva	Tecnica ML/ Heuristica	Mitigazione	Dati/ Feature	Real-time	Sink-hole	Reinie-zione	Metriche Operat.
[2]	Controller (Ryu)	Ensemble; LGBM migliore	Regole dinamiche (FLOW_MOD)	Feature controller-based da OpenFlow	Sì	No	No	Limitato
[16]	N/A (survey)	Tassonomia approcci	Filtraggio, rate limiting, honeypot	Classificazione per scenario SDN/IoT	N/A	Teorico	N/A	N/A
[4]	Honeypot + SDN	Classificatori tradizionali (RF/J48)	Distribuzione dinamica regole	Log honeypot $\rightarrow$ nuove regole	Parziale	Sì	No	Base
[18]	Honeypot + Controller	Semi-supervised (SVM+AdaBoost)	FLOW_MOD via REST ($t \sim 40$s)	Alert Suricata	Sì	Limitato	No	Base
[14]	Pipeline online	CNN+LSTM spazio-temporali	Regole + IP traceback	Dataset CIC/SDN; $\geq 99\%$ acc.	Sì	No	No	Accuratezza
[9]	Valutazione dataset	Gradient Boosting (GBDT)	Focus detection	UNSW-NB15; F1 ≈ 0.998	No	No	No	F1-score
Sistema Proposto	**Controller (Ryu + ML)**	**Decision Tree binario**	**Redirezione verso sinkhole**	**Feature UNSW-NB15 adattate**	**Sì**	**Sì**	**Sì**	**Estensivo**

Tabella 6.1: Confronto esteso tra approcci per rilevamento/mitigazione DDoS in SDN con focus sulle funzionalità operative del sistema proposto.

6.5 Limiti e possibili miglioramenti

Nonostante i risultati positivi ottenuti, l'analisi critica del sistema rivela alcune limitazioni che offrono spunti per sviluppi futuri e ottimizzazioni dell'architettura proposta.

6.5.1 Limitazioni del Modello di Classificazione

Il modello Decision Tree, pur garantendo efficienza computazionale e interpretabilità, presenta alcuni limiti intrinseci nella capacità di generalizzazione. La natura binaria della classificazione (benigno/malevolo) potrebbe risultare eccessivamente semplicistica per scenari operativi complessi, dove diverse tipologie di attacco richiederebbero strategie di mitigazione differenziate.

Da un lato, questo approccio riduce la complessità decisionale e migliora i tempi di risposta; dall'altro, limita la granularità dell'analisi e potrebbe portare a strategie di difesa sub-ottimali per attacchi sofisticati. Un'evoluzione naturale del sistema potrebbe prevedere l'implementazione di classificatori multi-classe, capaci di distinguere tra diverse tipologie di attacco (UDP flood, SYN flood, HTTP flood, ecc.) e di attivare contromisure specifiche.

6.5.2 Scalabilità e Distribuzione

L'architettura centralizzata del sistema di classificazione, pur semplificando l'implementazione e la gestione, rappresenta un potenziale punto di failure e un limite alla scalabilità. In scenari enterprise complessi, con múltipli punti di ingresso e volumi di traffico elevati, un singolo server ML potrebbe diventare un collo di bottiglia critico.

L'implementazione di un'architettura distribuita, con múltipli nodi di classificazione e meccanismi di load balancing, potrebbe migliorare significativamente la capacità del sistema di gestire carichi elevati. Non va trascurato che questa evoluzione richiederebbe la progettazione di meccanismi di coordinamento per garantire coerenza nelle decisioni di mitigazione.

6.5.3 Sofisticazione delle Contromisure

L'approccio attuale basato su sinkholing, pur efficace per gli attacchi volumetrici testati, potrebbe risultare insufficiente contro tecniche di evasion più avanzate. Attaccanti sofisticati potrebbero implementare strategie di traffic shaping per eludere le soglie di detection, o utilizzare pattern di traffico che mimano comportamenti legittimi.

Lo sviluppo di contromisure adattive, capaci di modificare dinamicamente i parametri di detection e mitigazione in base ai pattern di attacco osservati, rappresenterebbe un significativo miglioramento dell'efficacia del sistema. Questo approccio richiederebbe l'integrazione di tecniche di machine learning più avanzate, possibilmente basate su deep learning o reinforcement learning.

6.5.4 Ottimizzazione delle Feature

Il set di feature attualmente utilizzato, derivato dal dataset UNSW-NB15, potrebbe non essere ottimale per tutti gli scenari operativi. L'analisi dei risultati sperimentali suggerisce che alcune feature hanno peso maggiore nelle decisioni del classificatore, mentre altre contribuiscono marginalmente alla discriminazione.

Un processo di feature selection più raffinato, possibilmente basato su tecniche di analisi dell'importanza e validazione incrociata, potrebbe migliorare sia l'accuratezza che l'efficienza computazionale del sistema. L'integrazione di feature derivate da analisi di traffico specifiche per il dominio applicativo (web traffic, email, database, ecc.) potrebbe inoltre incrementare la precisione della detection.

6.5.5 Gestione dei Falsi Positivi

Nonostante il tasso contenuto di falsi positivi osservato negli esperimenti, la gestione di questi eventi rimane un aspetto critico per l'accettabilità operativa del sistema. In scenari production, anche tassi di falsi positivi apparentemente trascurabili possono generare significativi disservizi se applicati a volumi di traffico elevati.

L'implementazione di meccanismi di feedback e learning continuo, che permettano al sistema di apprendere dai falsi allarmi e di raffinare progressivamente i criteri di classificazione, rappresenterebbe un miglioramento sostanziale. Questo approccio richiederebbe l'integrazione di capacità di online learning e la definizione di protocolli per la validazione delle decisioni del sistema.

6.5.6 Integrazione con Sistemi Esistenti

L'architettura proposta, pur dimostrando efficacia negli scenari di test, richiederebbe significativi adattamenti per l'integrazione in infrastrutture enterprise esistenti. La dipendenza da controller SDN centraliz-

zati e la necessità di modificare le configurazioni di rete potrebbero rappresentare barriere all'adozione in contesti operativi complessi.

Lo sviluppo di modalità di deployment più flessibili, possibilmente basate su approcci ibridi che combinino tecnologie SDN con componenti di sicurezza tradizionali, potrebbe facilitare l'adozione del sistema. L'integrazione con Security Information and Event Management (SIEM) systems e l'esposizione di API standardizzate rappresenterebbero ulteriori fattori abilitanti per l'adozione operativa.

Capitolo 7

Conclusioni e Sviluppi Futuri

Questo lavoro ha esplorato l'integrazione tra paradigmi SDN e tecniche di machine learning per la costruzione di un sistema di difesa contro attacchi DDoS. Il percorso di ricerca, dall'analisi dello stato dell'arte alla validazione sperimentale, ha permesso di verificare concretamente la fattibilità di un approccio ibrido che combina l'agilità del controllo centralizzato con l'efficacia predittiva degli algoritmi di classificazione.

Il sistema proposto rappresenta più di un semplice prototipo dimostrativo: durante i mesi di sviluppo e testing, ha dimostrato di poter operare stabilmente in scenari realistici, gestendo traffico misto e reagendo tempestivamente alle minacce rilevate. Non si tratta di un risultato scontato, considerando le complessità intrinseche nell'orchestrare componenti eterogenei come controller Ryu, servizi Flask e honeypot analitici in una pipeline coerente end-to-end.

7.1 Sintesi dei risultati raggiunti

La validazione sperimentale ha confermato le intuizioni progettuali iniziali, evidenziando prestazioni solide su diverse dimensioni critiche. Dal punto di vista dell'efficacia di rilevamento, il sistema ha dimostrato eccellenti capacità nella discriminazione tra traffico benigno e malevolo, con tassi di rilevamento elevati e falsi positivi contenuti. Durante i test più intensi, gli attacchi volumetrici sono stati identificati e mitigati in modo consistente, confermando la validità dell'approccio basato su feature estratte dal controller.

L'aspetto temporale si è rivelato particolarmente soddisfacente. I ritardi di mitigazione medi di 16.2 millisecondi rappresentano un risultato rilevante per un sistema che deve orchestrare comunicazioni tra controller, server ML esterno e installazione di regole OpenFlow. Questo risultato è ancor più significativo considerando che include il tempo di estrazione delle feature, serializzazione, classificazione remota e installazione delle regole di flusso – una catena di operazioni che avrebbe potuto facilmente introdurre latenze incompatibili con i requisiti operativi.

Il comportamento dell'honeypot ha superato le aspettative iniziali. La capacità di processare 23.36 pacchetti per secondo, discriminando efficacemente tra traffico da bloccare (69.9%) e da reiniettare (30.1%), dimostra che l'approccio basato su sinkhole intelligente offre vantaggi concreti rispetto al blocco immediato. Questi numeri nascondono una considerazione importante: il sistema non si limita a "dire no" al

traffico sospetto, ma implementa una seconda linea di analisi che recupera comunicazioni erroneamente intercettate.

Le prestazioni del server ML si sono dimostrate stabili e prevedibili. Le latenze medie di 8.8 millisecondi per la classificazione, con picchi contenuti sotto i 20 millisecondi, confermano che l'approccio basato su Decision Tree è ben dimensionato per il carico operativo. Durante gli esperimenti più lunghi, non sono emersi segni di degradazione o accumulo di errori, suggerendo una buona robustezza computazionale del modello scelto.

Un risultato inaspettato riguarda la stabilità complessiva del sistema durante i test di carico. Inizialmente temevamo che l'orchestrazione di componenti distribuiti potesse introdurre condizioni di gara o stati inconsistenti, specialmente durante i picchi di traffico malevolo. Invece, il sistema ha mantenuto coerenza operativa per l'intera durata degli esperimenti, gestendo in modo elegante anche scenari di stress elevato.

L'integrazione con Mininet ha permesso di validare il sistema in condizioni controllate ma realistiche. La separazione netta tra rete attaccante (10.0.2.0/24) e rete vittima (10.0.1.0/24), mediata da instradamento e controllo SDN, ha riprodotto fedelmente architetture aziendali tipiche. Questo aspetto è cruciale: troppo spesso, soluzioni accademiche brillanti su carta si rivelano difficilmente adattabili a contesti operativi reali.

7.2 Considerazioni sul sistema proposto

Riflettendo sul percorso compiuto, emergono alcune considerazioni che vanno oltre i semplici numeri prestazionali. Il sistema sviluppato rappresenta un esempio concreto di come principi teorici – separazione tra controllo e dati, intelligenza centralizzata, mitigazione adattiva – possano essere tradotti in implementazioni operative efficaci.

L'architettura adottata privilegia esplicitamente semplicità e manutenibilità rispetto a ottimizzazioni prestazionali estreme. Questa scelta, inizialmente dettata da necessità di prototipazione rapida, si è rivelata strategicamente corretta: il sistema risulta comprensibile, correggibile e estensibile. In contesti reali, questi aspetti spesso prevalgono su guadagni marginali di prestazioni.

Da un lato, l'approccio centralizzato per la classificazione ML potrebbe sembrare un potenziale collo di bottiglia. Dall'altro, ha semplificato significativamente la gestione del ciclo di vita del modello: aggiornamenti, riaddestramento e rilascio avvengono in un singolo punto, eliminando problemi di versioning e sincronizzazione tipici di architetture distribuite. Per implementazioni di media scala, questo compromesso appare ben bilanciato.

Il meccanismo di sinkhole analitico si è dimostrato più versatile del previsto. Oltre alla funzione primaria di analisi del traffico sospetto, abilita scenari operativi interessanti: analisi forense post-incidente, raccolta di intelligence su tecniche di attacco, e soprattutto recupero automatico di comunicazioni erroneamente classificate. Quest'ultimo aspetto è particolarmente valoroso in ambienti produttivi, dove il costo dei falsi positivi può essere elevato.

Un elemento distintivo emerso durante lo sviluppo riguarda la natura "auto-documentante" del sistema. Le metriche estensive implementate – dalle latenze ML ai contatori di mitigazione – forniscono visibilità operativa continua. Questa caratteristica, spesso trascurata in prototipi accademici, si è rivelata cruciale

durante la fase di debug e ottimizzazione. Ogni componente "racconta" continuamente il proprio stato, facilitando la comprensione di comportamenti anomali o l'identificazione di opportunità di miglioramento.

L'integrazione con il dataset UNSW-NB15, seppur mediata da un processo di adattamento delle feature, ha confermato la validità dell'approccio basato sul controller per l'estrazione di informazioni discriminative. Non è stato necessario ricorrere a ispezione profonda dei pacchetti costosa o analisi comportamentali complesse: le statistiche aggregate disponibili nativamente nel controller si sono dimostrate sufficientemente ricche per alimentare classificatori efficaci.

Tuttavia, è importante riconoscere che il sistema opera all'interno di alcuni vincoli architetturali significativi. La dipendenza da topologie SDN specifiche, l'assunzione di traffico prevalentemente unidirezionale (attaccante→vittima), e l'utilizzo di classificazione binaria semplificata sono scelte che ne limitano la generalità applicativa. Questi aspetti non rappresentano difetti intrinseci, ma piuttosto compromessi consapevoli che hanno reso possibile la realizzazione di un prototipo funzionante nei tempi disponibili.

7.3 Estensioni future

L'evoluzione naturale del sistema proposto si articola su diverse direzioni, alcune complementari, altre alternative, tutte finalizzate a superare le limitazioni attuali e ad abilitare scenari d'uso più ampi.

7.3.1 Miglioramenti algoritmici e architetturali

L'estensione più immediata riguarda l'arricchimento del modello di classificazione. L'attuale approccio binario (benigno/malevolo) potrebbe evolvere verso classificatori multi-classe capaci di distinguere specifiche tipologie di attacco: UDP flood, SYN flood, attacchi HTTP lenti, amplificazione DNS. Ogni categoria richiederebbe strategie di mitigazione ottimizzate, dal rate limiting granulare alla redirezione verso honeypot specializzati.

L'integrazione di tecniche di apprendimento ensemble rappresenta un'evoluzione promettente. Combinando Decision Tree con Random Forest, Gradient Boosting e SVM in un sistema di voto, si potrebbero ottenere accuratezze superiori mantenendo latenze accettabili. Gli esperimenti preliminari con LightGBM citati in letteratura suggeriscono margini di miglioramento significativi senza compromessi eccessivi sulla velocità di inferenza.

Da un punto di vista architetturale, l'implementazione di capacità di apprendimento federato aprirebbe scenari interessanti. Molteplici installazioni del sistema potrebbero condividere conoscenza sulle nuove minacce senza esporre dati sensibili locali. Un controller centrale potrebbe aggregare pattern di attacco osservati da installazioni distribuite, migliorando progressivamente l'efficacia di rilevamento su scala geografica.

7.3.2 Scalabilità e distribuzione

La scalabilità orizzontale rappresenta una direzione di sviluppo critica per l'adozione aziendale. L'architettura attuale, basata su server ML singolo, potrebbe evolvere verso cluster di classificatori con bilanciamento

del carico intelligente. L'instradamento basato su hash consistente potrebbe garantire che flussi correlati vengano processati dallo stesso nodo, mantenendo consistenza nelle decisioni temporali.

L'implementazione di meccanismi di auto-scalabilità, possibilmente integrati con orchestrazione di container come Kubernetes, permetterebbe di dimensionare dinamicamente le risorse di classificazione in base al carico di traffico. Durante gli attacchi intensi, il sistema potrebbe generare automaticamente istanze aggiuntive del server ML, distribuendo il carico e mantenendo latenze accettabili.

Un'estensione particolarmente intrigante riguarda l'integrazione con edge computing. Disporre di capacità di pre-classificazione direttamente negli switch programmabili (P4) potrebbe ridurre drasticamente il traffico diretto verso il controller centrale. Algoritmi di classificazione leggeri, implementati in hardware, filtrerebbero il traffico manifestamente benigno, inoltrando verso il controller solo i flussi ambigui che richiedono analisi ML sofisticate.

7.3.3 Integrazione e interoperabilità

L'estensione verso architetture ibride SDN/tradizionali rappresenta una necessità pratica per l'adozione in ambienti esistenti. L'implementazione di gateway che traducano decisioni SDN in configurazioni per firewall tradizionali, router Cisco, e dispositivi di sicurezza permetterebbe di beneficiare dell'intelligence centralizzata anche in reti non completamente software-defined.

L'integrazione con standard di orchestrazione come YANG (Yet Another Next Generation) e NETCONF (Network Configuration Protocol) potrebbe facilitare la gestione uniforme di infrastrutture eterogenee. Policy definite ad alto livello potrebbero essere tradotte automaticamente in configurazioni specifiche per ogni fornitore, mantenendo coerenza operativa tra domini.

Un aspetto spesso trascurato riguarda l'integrazione con flussi di lavoro operativi esistenti. Connettori per sistemi SIEM (Security Information and Event Management) come Splunk, ElasticStack, IBM QRadar, sistemi di ticketing come ServiceNow e Jira, e piattaforme di comunicazione come Slack e Teams potrebbero automatizzare la notifica e l'escalation degli incidenti. Webhook configurabili permetterebbero integrazioni personalizzate con strumenti aziendali specifici.

7.3.4 Ricerca e validazione estesa

Dal punto di vista scientifico, diversi aspetti meriterebbero approfondimento attraverso ricerca estesa. L'efficacia del sistema contro attacchi adversarial – dove gli attaccanti modificano deliberatamente i pattern di traffico per eludere la classificazione – richiederebbe validazione su dataset specifici e possibilmente simulazioni di red team.

L'analisi prestazionale su scala realistica rappresenta un'area di ricerca importante. Test su topologie con centinaia di switch, migliaia di host, e pattern di traffico aziendali reali potrebbero rivelare colli di bottiglia non evidenti negli esperimenti attuali.

Un aspetto particolarmente interessante riguarda la generalizzazione cross-domain. Il sistema, addestrato su UNSW-NB15, mantiene efficacia su traffico aziendale reale? Applicazioni web moderne, traffico IoT,

e pattern di comunicazione che non esistevano quando il dataset fu creato potrebbero richiedere adattamenti significativi del set di feature.

7.3.5 Impatto e trasferimento tecnologico

Il trasferimento di questa ricerca verso applicazioni pratiche potrebbe assumere diverse forme.

L'integrazione con progetti open-source esistenti (OpenDaylight, ONOS, PICA8) potrebbe accelerare l'adozione attraverso contributi della comunità. Architetture di plugin potrebbero permettere a sviluppatori terzi di estendere le capacità del sistema, creando ecosistemi intorno alla piattaforma base.

Collaborazioni con fornitori di apparecchiature di rete potrebbero facilitare l'integrazione nativa nei prodotti commerciali. Funzionalità di protezione DDoS basate su ML, integrate direttamente in switch e router, potrebbero diventare fattori di differenziazione competitiva nel mercato delle reti aziendali.

—

In conclusione, questo lavoro rappresenta un primo passo concreto verso l'integrazione matura di SDN e ML per la sicurezza delle reti. I risultati ottenuti, pur all'interno dei limiti di un prototipo di ricerca, dimostrano la fattibilità e l'efficacia dell'approccio proposto. Le direzioni di sviluppo future delineate aprono prospettive stimolanti sia per la ricerca accademica che per l'applicazione industriale.

Il sistema sviluppato, nella sua semplicità architettonica consapevole, offre una base solida per evoluzioni future. Durante i mesi di sperimentazione, ha dimostrato che teoria e pratica possono convergere in soluzioni operative efficaci. Non è poco, in un'area di ricerca dove troppo spesso prototipi brillanti sulla carta si rivelano inadatti alla realtà operativa.

L'augurio è che questo contributo possa stimolare ulteriori ricerche nell'area, fornendo sia spunti metodologici che lezioni apprese pratiche. La sicurezza delle reti è una sfida in continua evoluzione: soluzioni innovative, validate attraverso sperimentazione rigorosa e aperte a raffinamenti progressivi, rappresentano passi necessari verso infrastrutture digitali più resilienti e affidabili.

Riferimenti Bibliografici

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese e D. Walker, «P4: Programming Protocol-Independent Packet Processors,» *ACM SIGCOMM Computer Communication Review*, vol. 44, n. 3, pp. 87–95, 2014. DOI: 10.1145/2656877.2656890

[2] P. Chauhan e M. Atulkar, «Ryu Controller-Based Attack Detection and Mitigation Method in Software Defined Internet of Things,» *International Journal of Engineering Trends and Technology*, vol. 71, n. 9, pp. 138–156, 2023. DOI: 10.14445/22315381/IJETT-V71I9P213

[3] P. Ferguson e D. Senie, *BCP 38: Network Ingress Filtering*, RFC 2827, 2000. indirizzo: https://www.rfc-editor.org/info/rfc2827

[4] J. Franco, A. Aris, L. Babun e A. S. Uluagac, «S-Pot: A Smart Honeypot Framework with Dynamic Rule Configuration for SDN,» in *2022 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2022.

[5] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer e O. Koufopavlou, *Software-Defined Networking (SDN): Layers and Architecture Terminology*, RFC 7426 (IRTF), 2015. DOI: 10.17487/RFC7426 indirizzo: https://www.rfc-editor.org/rfc/rfc7426

[6] M. Handley, E. Rescorla e the IAB, *Internet Denial-of-Service Considerations*, RFC 4732, 2006. DOI: 10.17487/RFC4732 indirizzo: https://www.rfc-editor.org/rfc/rfc4732

[7] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky e S. Uhlig, «Software-Defined Networking: A Comprehensive Survey,» *Proceedings of the IEEE*, vol. 103, n. 1, pp. 14–76, 2015. DOI: 10.1109/JPROC.2014.2371999

[8] B. Lantz, B. Heller e N. McKeown, «A Network in a Laptop: Rapid Prototyping for Software-Defined Networks,» in *Proceedings of ACM HotNets-IX*, 2010. indirizzo: https://conferences.sigcomm.org/hotnets/2010/papers/a19-lantz.pdf

[9] M. Z. Mahmud, S. R. Alve, S. Islam e M. M. Khan, *SDN Intrusion Detection Using Machine Learning Method*, arXiv preprint arXiv:2411.05888, 2024. indirizzo: https://arxiv.org/abs/2411.05888

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker e J. Turner, «OpenFlow: Enabling Innovation in Campus Networks,» *ACM SIGCOMM Computer Communication Review*, vol. 38, n. 2, pp. 69–74, 2008. DOI: 10.1145/1355734.1355746

[11] S. Y. Mehr e B. Ramamurthy, «An SVM Based DDoS Attack Detection Method for Ryu SDN Controller,» in *CoNEXT '19 Companion: The 15th International Conference on Emerging Networking Experiments and Technologies*, Orlando, FL, USA, 2019. DOI: 10.1145/3360468.3368183

[12] T. Mortensen et al., *DDoS Open Threat Signaling (DOTS) Architecture*, RFC 8811, 2020. indirizzo: https://www.rfc-editor.org/rfc/rfc8811

[13] N. Moustafa e J. Slay, «UNSW-NB15: A Comprehensive Data Set for Network Intrusion Detection Systems,» in *IEEE MilCIS*, 2015. DOI: 10.1109/MilCIS.2015.7348942

[14] D. M. Rajan e J. Aravindhar, «Detection and Mitigation of DDoS Attack in SDN Environment Using Hybrid CNN–LSTM,» *Migration Letters*, vol. 20, n. S13, pp. 407–419, 2023.

[15] *Ryu Documentation*, https://ryu.readthedocs.io/, Accesso: 2025-08-13.

[16] F. S. D. e Silva, E. Silva, E. P. Neto, M. Lemos, A. J. V. Neto e F. Esposito, «A Taxonomy of DDoS Attack Mitigation Approaches Featured by SDN Technologies in IoT Scenarios,» *Sensors*, vol. 20, n. 11, p. 3078, 2020. DOI: 10.3390/s20113078

[17] L. Spitzner, *Honeypots: Tracking Hackers*. Addison-Wesley, 2002.

[18] F. D. S. Sumadi, C. S. K. Aditya, A. A. Maulana, Syaifuddin e V. Suryani, «Cyber Security: Distributed Denial of Service Honeypot in Software-Defined Network Using Semi-Supervised Learning,» *International Journal of Artificial Intelligence*, vol. 11, n. 3, pp. 1094–1100, 2022.

[19] *The UNSW-NB15 Dataset*, https://research.unsw.edu.au/projects/unsw-nb15-dataset, Accesso: 2025-08-13.

[20] Z. Yang et al., «A Systematic Review of Methods and Datasets for Network Intrusion Detection,» *Computers & Security*, vol. 114, p. 102 580, 2022. DOI: 10.1016/j.cose.2021.102580