UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA, ELETTRONICA E SISTEMISTICA

DIMES

CdL Magistrale in

Ingegneria

Informatica

Relazione Progetto - Network Security

Conflict Detector

Docente

Prof. [redacted]

Esercitatore

Ing. [redacted]

Studenti

Federica Cosenza [redacted]

Michele Purrone [redacted]

Antonino Vaccarella [redacted]

Anno Accademico 2023/2024

Indice

Abstract

Il **software-defined networking** (**SDN**) è una tecnologia di networking che utilizza i controller basati su software per gestire, automatizzare, fornire e programmare le risorse di rete per migliorarne la gestione e il controllo. La programmabilità resa possibile dal software-defined networking lo rende una piattaforma ideale per molteplici applicazioni, tra cui il deployment, i cambiamenti dinamici della topologia e la gestione decentralizzata delle reti nei data center multi-tenant.
L'idea progettuale consiste nel ricreare il framework di analisi delle policy di sicurezza **Brew**, costruito sul controller SDN OpenDaylight. Brew estende la classificazione dei conflitti delle regole firewall tradizionali alle regole di flusso del controller, automatizzando la risoluzione di conflitti a più livelli.

1. Introduzione

Il software-defined networking (SDN) è un approccio innovativo alla progettazione e all'implementazione delle reti, basato sulla separazione del controllo delle funzioni di rete dai dispositivi di rete stessi (switch, router, firewall, bilanciatori di carico, ecc.). Utilizzando il protocollo **OpenFlow**, gli switch SDN possono sfruttare la flessibilità data dall'accesso alle informazioni di intestazione di vari livelli dello stack OSI, permettendo di soddisfare funzionalità tradizionalmente realizzate da numerosi dispositivi fisici.

Qualsiasi implementazione di sicurezza da parte del tenant, come Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), Virtual Private Networks (VPN), Moving Target Defense (MTD), ecc., può essere realizzata installando nuove regole di flusso nell'ambiente basato su SDN. A differenza degli ambienti tradizionali dove le nuove regole possono essere aggiunte solo attraverso un amministratore, l'astrazione del piano dati dal piano di controllo consente alle applicazioni di introdurre nuove regole di flusso nel controller tramite un'API. Quando ciò avviene senza valutare le regole di flusso esistenti potrebbero sorgere conflitti. A complicare ulteriormente le cose, una topologia di rete dinamica aggiunge ulteriori complessità.

In questo lavoro si classificano tutti i potenziali conflitti tra le regole di flusso e si implementa un algoritmo sul controller per estrarre le regole e rilevare i conflitti tra di esse.

Il lavoro prende ispirazione dal framework di analisi delle policy di sicurezza Brew, costruito su un controller SDN basato su OpenDaylight (ODL), che sanifica efficacemente la tabella delle regole di flusso, evidenzia e rileva i potenziali conflitti.

1.1 Ubuntu

Ubuntu (https://www.ubuntu-it.org/) è una delle distribuzioni Linux più popolari e utilizzate a livello globale.
La versione utilizzata in questo progetto è *Ubuntu 18.04.6 LTS*.

1.2 Mininet

Mininet (https://mininet.org/ e https://github.com/mininet/mininet) è uno strumento essenziale nel campo delle reti software-defined e della ricerca sulle reti in generale. È un simulatore di rete open source che permette di creare e testare ambienti di rete complessi, configurazioni innovative, protocolli e applicazioni su una singola macchina, senza la necessità di hardware costoso o reti fisiche elaborate.
L'interfaccia di Mininet, accessibile tramite linea di comando o script Python, permette agli utenti di definire la topologia della rete, includendo switch e host, e configurando le connessioni tra di essi. Dopo aver stabilito la topologia, Mininet configura automaticamente gli switch e i router, avvia i container Linux che rappresentano gli host, e imposta le connessioni di rete tra tutti i componenti.

Architettura di Mininet

Mininet utilizza diverse componenti chiave per creare la sua simulazione:

1. **Switch Virtuali**: Mininet emula switch di rete utilizzando software come **Open vSwitch (OVS)**. Questi switch virtuali gestiscono il traffico di rete tra gli host virtuali e possono essere configurati per supportare diversi protocolli di rete e topologie.
2. **Host Virtuali**: gli host sono rappresentati da container Linux all'interno di Mininet. Ogni container ha il proprio spazio di rete e può eseguire applicazioni, generare traffico, e interagire con altri host nella rete virtuale.
3. **Controller SDN**: Mininet può essere utilizzato per simulare reti gestite da controller SDN, come ONOS o OpenDaylight. Gli utenti possono configurare Mininet per interagire con un controller SDN esterno, consentendo di testare come il controller gestisce e configura la rete virtuale.

4. **Topology Generator**: Mininet include un generatore di topologie che permette agli utenti di creare facilmente reti complesse. È possibile utilizzare script Python per definire e personalizzare la topologia della rete, inclusi il numero di switch e host, le connessioni tra di essi e le configurazioni degli switch.

Vantaggi

Tra i principali vantaggi di Mininet, spicca la sua capacità di simulare reti complesse senza la necessità di hardware fisico, riducendo significativamente i costi associati alla creazione di ambienti di rete e alla conduzione di esperimenti. Mininet è inoltre estremamente flessibile e scalabile, permettendo di adattare facilmente la topologia della rete, aggiungere nuovi componenti e modificare le configurazioni in base alle specifiche esigenze dei test.

Limitazioni

Tuttavia, è importante considerare alcune limitazioni. Sebbene Mininet emuli una rete in maniera virtuale, la simulazione di reti molto grandi o complesse può richiedere una quantità significativa di risorse hardware, come CPU e memoria, limitando così la dimensione e la complessità della rete che può essere simulata su una singola macchina. Inoltre, pur offrendo una simulazione realistica, non può replicare tutti gli aspetti delle reti fisiche, come dettagli specifici del comportamento dell'hardware o influenze ambientali.

1.3 OpenFlow

OpenFlow è uno dei protocolli più influenti e fondamentali nel campo delle reti software-defined. Sviluppato dal progetto Open Networking Foundation (ONF), OpenFlow rappresenta un'evoluzione significativa nel modo in cui le reti sono progettate, gestite e controllate. Prima dell'avvento di OpenFlow, le reti erano tradizionalmente gestite attraverso dispositivi hardware proprietari e configurazioni statiche, ma OpenFlow ha introdotto un paradigma innovativo che ha cambiato radicalmente questo approccio.
OpenFlow è un protocollo di comunicazione che consente la separazione tra il piano di controllo e il piano dati nelle reti. In un contesto di rete tradizionale, il piano di controllo e il piano dati sono strettamente integrati nei dispositivi di rete (come switch e router). Questo significa che il dispositivo decide autonomamente come gestire il traffico basandosi sulle proprie tabelle di routing e forwarding.
Con OpenFlow, invece, il piano di controllo e il piano dati sono separati: il primo è costituito da un controller SDN, un software centralizzato che gestisce le decisioni sulla rete. Il controller utilizza OpenFlow per comunicare con gli switch e gli altri

dispositivi di rete, inviando istruzioni su come trattare il traffico. Può modificare dinamicamente le regole di inoltro, aggiornare le tabelle di routing e gestire le politiche di rete in base alle condizioni attuali della rete e alle esigenze degli utenti. Il piano dati è invece costituito dai dispositivi di rete, come switch e router, che sono responsabili del forwarding dei pacchetti. Questi dispositivi seguono le istruzioni del controller SDN e utilizzano le tabelle di flusso per decidere come inoltrare i pacchetti attraverso la rete.

Tabelle di Flusso

Nell'ambito di OpenFlow, gli switch mantengono delle tabelle di flusso che contengono le regole su come gestire i pacchetti. Ogni regola nella tabella di flusso specifica un criterio di corrispondenza (come indirizzi IP, porte, protocolli) e un'azione da intraprendere (come inoltrare il pacchetto a una certa porta, modificarne il contenuto, ecc.). Quando un pacchetto arriva a uno switch, il dispositivo consulta la tabella di flusso per determinare come trattare il pacchetto.

Protocollo di Comunicazione

OpenFlow utilizza un protocollo di comunicazione tra il controller e gli switch: esso è responsabile del trasferimento delle istruzioni di controllo e delle informazioni di stato. OpenFlow specifica vari messaggi, tra cui:

- **Flow Mod**: usato dal controller per aggiungere, modificare o rimuovere le regole nella tabella di flusso degli switch.
- **Packet-In**: inviato dallo switch al controller quando un pacchetto non corrisponde a nessuna regola nella tabella di flusso, richiedendo al controller di decidere come gestirlo.
- **Packet-Out**: inviato dal controller allo switch per ordinare l'inoltro di un pacchetto specifico verso una o più porte.
- **Flow-Removed**: notifica al controller che una regola della tabella di flusso è stata rimossa.

1.4 ONOS

Open Network Operating System (https://opennetworking.org/onos/ e https://github.com/opennetworkinglab/onos) è un sistema operativo open source dedicato alla gestione e al controllo delle reti, progettato per rispondere alle esigenze delle moderne infrastrutture di rete su larga scala. Sviluppato dalla Open Networking Foundation (https://opennetworking.org), ONOS rappresenta una delle soluzioni più avanzate nel campo del *SDN* e della Network Functions Virtualization (NFV). La sua architettura e le sue funzionalità lo rendono particolarmente adatto in ambito

industriale ed è impiegato dai grandi operatori di telecomunicazioni, dove garantisce una gestione flessibile, scalabile e sicura delle risorse di rete.

Architettura del Sistema

ONOS è costruito su un'architettura a microservizi: ognuno di essi è un'unità indipendente che gestisce una parte del controllo della rete, come la gestione del traffico, la configurazione dei dispositivi, o la raccolta e l'analisi dei dati di rete. Il design modulare consente una grande flessibilità e scalabilità, permettendo al sistema di adattarsi e crescere in base alle esigenze specifiche di ogni ambiente di rete. I componenti fondamentali sono:

- **Il controller**, che si occupa di coordinare e gestire le operazioni della rete. Il piano di controllo è separato dal piano dati: il primo, gestito dal controller, si occupa di prendere decisioni strategiche su come i dati devono essere instradati e trattati, mentre il secondo, costituito da switch e router fisici o virtualizzati, esegue effettivamente l'instradamento dei dati.
- **Il framework di servizi**, che facilita l'integrazione e la gestione delle applicazioni di rete. Il framework, impiegato nella realizzazione dell'idea progettuale, consente di creare e implementare nuove applicazioni che possono interagire con il sistema operativo della rete senza dover modificare il cuore del sistema.
- **Il database distribuito**, il cui obiettivo è che le informazioni siano sempre disponibili e sincronizzate.

Interfacce e API

ONOS fornisce una serie di API RESTful e altre interfacce per facilitare l'integrazione con applicazioni di rete e strumenti di gestione: le API permettono agli sviluppatori di interagire con ONOS e di personalizzare il sistema in base alle loro necessità. La disponibilità di queste interfacce semplifica anche la creazione di applicazioni personalizzate per la gestione della rete, che possono essere utilizzate per monitorare le prestazioni, configurare i dispositivi o implementare nuove politiche di rete.

Installazione

La versione di ONOS di riferimento per la realizzazione dell'idea progettuale è la 2.6.0.

La versione di ONOS adottata richiede JAVA 11 (informazione consultabile sulla pagina https://wiki.onosproject.org/display/ONOS/Requirements). Si eseguono i seguenti comandi sul terminale:

```
sudo apt update
sudo apt install openjdk-11-jdk
```

e si imposta la variabile *$JAVA_HOME*:

```
sudo cat >> /etc/environment <<EOL
JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
EOL
```

La pagina https://wiki.onosproject.org/display/ONOS/Installing+on+a+single+machine presenta i passi da seguire per l'installazione:

```
sudo mkdir /opt
cd /opt
sudo wget -c https://repo1.maven.org/maven2/org/onosproject/onos-
releases/2.6.0/onos-2.6.0.tar.gz
```

Una volta scaricato l'archivio *tar*, occorre decomprimere l'archivio e rinominare la directory estratta in "*onos*".

```
sudo tar -xvf onos-2.6.0.tar.gz
sudo mv onos-2.6.0 onos
```

A questo punto è possibile avviare il servizio ONOS e la CLI con i comandi:

```
sudo /opt/onos/bin/onos-service start
sudo /opt/onos/bin/onos -l onos
```

Dalla CLI si attivano i due moduli essenziali per il protocollo OpenFlow e per la gestione del traffico:

```
app activate org.onosproject.openflow
app activate org.onosproject.fwd
```

Creazione di un’Applicazione

Il comando da eseguire per creare una nuova applicazione è:

```
mvn archetype:generate -DarchetypeGroupId=org.onosproject -
DarchetypeArtifactId=onos-bundle-archetype -DarchetypeVersion=2.2.1-
b2
```

avendo cura di cambiare opportunamente la versione *archetype* che potrebbe variare nel tempo.

Maven scaricherà le dipendenze necessarie e chiederà di impostare le proprietà “groupId” e “artifactId” del progetto, per poi finalizzare la creazione.

Installazione e Attivazione dell’Applicazione

È possibile compilare l'applicazione in un archivio di applicazioni ONOS o in un file .oar utilizzando il comando:

```
sudo mvn clean install
```

L’attivazione avviene caricando il file *.oar* e attivando l’applicazione tramite la GUI disponibile all’indirizzo http://127.0.0.1:8181/onos/ui.

Per rendere più semplice l’avvio di ONOS e il processo di compilazione, è stato creato uno script “run.sh” che automatizza il processo sopra descritto:

```
#!/bin/bash

PASSWORD="Arabella"

# Avvio di ONOS
gnome-terminal --tab -- bash -c "echo $PASSWORD | sudo -S
/opt/onos/bin/onos-service start; exec bash"

# Attesa di 15 secondi
sleep 15
```

```
# Avvio della CLI
gnome-terminal --tab -- bash -c "/opt/onos/bin/onos -l onos; exec
bash"

# Compilazione
gnome-terminal --tab -- bash -c "echo $PASSWORD | sudo -S mvn clean
install; exec bash"
```

Altri Controller

Nel corso dello sviluppo dell'idea progettuale, sono stati approfonditi anche i controller Floodlight, POX e Ryu, poi sostituiti in favore di ONOS:

- **Floodlight** (*https://github.com/floodlight/floodlight*) è un controller OpenFlow open-source per reti SDN: è programmabile, flessibile e supporta vari dispositivi e protocolli di rete. È sviluppato in linguaggio Java e beneficia di una comunità attiva che contribuisce al suo miglioramento continuo. È stato scartato in favore di ONOS poiché esso permette il caricamento di applicazioni personalizzate in modo più semplificato.
- **POX** (*https://github.com/noxrepo/pox*) è un controller OpenFlow leggero e open-source utilizzato principalmente per la ricerca e l'istruzione in reti SDN. Scritto in Python, è noto per la sua facilità d'uso e la capacità di permettere lo sviluppo rapido di applicazioni di rete. POX supporta varie funzionalità di rete, tra cui instradamento, sicurezza e gestione del traffico. È stato sostituito con ONOS a causa della maggiore familiarità con Java.
- **RYU** (*https://github.com/faucetsdn/ryu*) è un controller SDN open-source scritto in Python. Progettato per essere altamente modulare e flessibile, RYU fornisce un framework per la gestione delle reti software-defined. Supporta vari protocolli di rete, tra cui OpenFlow, NETCONF, e BGP. Come per POX, si è preferito adottare ONOS per la migliore dimestichezza con Java.

2. Lavori correlati

Sebbene l'adozione dell'SDN stia diventando sempre più diffusa, i meccanismi di sicurezza associati a queste tecnologie vengono ancora in parte trascurati rispetto alle loro applicazioni pratiche.

Verifica delle regole di flusso

In risposta alla necessità di verificare l'integrità delle regole di flusso, è stato proposto *VeriFlow*, uno strato intermedio tra il controller SDN e gli switch. Lo strumento consente la verifica in tempo reale delle regole di flusso inserite, prevenendo errori derivanti da firmware difettosi, problemi di comunicazione nel piano di controllo, difficoltà di raggiungibilità o aggiornamenti di configurazione errati.

Gestione dei conflitti diretti

La gestione efficace dei conflitti di policy è stata affrontata da diverse soluzioni: *Pyretic*, ad esempio, è in grado di gestire i conflitti di policy diretti posizionandoli in un insieme di regole prioritarie, simile alla tabella di flusso di OpenFlow.
Un altro framework che si occupa della gestione dei conflitti tra le regole di flusso è Brew. In particolare, esso:

- Include tecniche per la prioritizzazione globale delle regole di flusso.
- Fornisce strategie per la risoluzione non assistita di questi conflitti.

3. Conflitti sulle regole di flusso

Le regole di flusso in una rete SDN sono istruzioni che vengono utilizzate per controllare il comportamento del traffico di rete all'interno di uno switch o di un router. Queste regole determinano come devono essere gestiti i pacchetti che transitano attraverso i dispositivi di rete.
Le regole di flusso possono includere diverse componenti:

- **Match**: le condizioni che un pacchetto deve soddisfare per corrispondere alla regola di flusso. Ad esempio, possono includere indirizzi IP sorgente e destinazione, numeri di porta, tipo di protocollo, ecc.
- **Azione**: le operazioni che devono essere eseguite sui pacchetti che corrispondono alla regola di flusso. Le azioni possono includere l'inoltro del pacchetto a una determinata porta, la modifica dell'intestazione, il drop del pacchetto, ecc.
- **Priorità**: un valore che indica l'importanza della regola. Le regole con una priorità più alta vengono applicate prima rispetto a quelle con una priorità più bassa se più regole corrispondono allo stesso pacchetto.
- **Tempo di vita (*Timeout*)**: specifica per quanto tempo una regola di flusso deve rimanere attiva prima di essere rimossa.

Una **flow table** (tabella di flusso) è una tabella in cui ogni voce rappresenta una regola di flusso, utilizzata per memorizzare e gestire le regole che determinano come i pacchetti di rete devono essere trattati e inoltrati all'interno del dispositivo.
La tabella è costituita da:

- **Match fields (*Campi di corrispondenza*)**: i criteri che un pacchetto deve soddisfare per corrispondere alla regola. Questi campi possono includere indirizzi IP, numeri di porta, tipo di protocollo, ecc.
- **Action (*Azione*)**: le operazioni che devono essere eseguite sui pacchetti che corrispondono ai campi di corrispondenza. Le azioni possono includere l'inoltro a una specifica porta, la modifica dei campi del pacchetto, il *drop* del pacchetto, ecc.
- **Priority (*Priorità*)**: un valore che determina l'importanza della regola.

Quando un pacchetto arriva ad uno switch, il dispositivo confronta i campi del pacchetto con le voci nella *flow table* per determinare quale regola di flusso si applica. Se il pacchetto corrisponde a una regola, vengono eseguite le azioni specificate. Se non c'è corrispondenza, il pacchetto può essere inoltrato al controller SDN per ulteriori istruzioni o gestito in base ad una regola predefinita.

TABLE 1
Flow Table Example

Rule #	Priority	Source MAC	Dest MAC	Source IP	Dest IP	Protocol	Source Port	Dest Port	Action
1	51	*	*	10.5.50.0/24	10.211.1.63	tcp	*	*	forward
2	50	*	*	10.5.50.5	10.211.1.63	tcp	*	80	forward
3	52	*	*	10.5.50.5	10.211.1.0/24	tcp	*	*	forward
4	53	*	*	10.5.50.0/24	10.211.1.63	tcp	*	*	drop
5	54	*	*	10.5.50.5	10.211.1.63	tcp	*	*	drop
6	51	*	*	10.5.50.0/16	10.211.1.63	tcp	*	*	drop
7	55	*	*	10.5.50.5	10.211.1.0/24	tcp	*	80-90	drop
8	57	11:11:11:11:11:ab	11:11:aa:aa:11:11	*	*	*	*	*	forward
9	58	*	*	*	*	tcp	*	80	drop

Figura 1: Esempio di Flow Table

3.1 Conflitti

I conflitti nelle regole di rete si riferiscono a situazioni in cui diverse regole di flusso interagiscono in modi che possono causare comportamenti imprevisti o indesiderati. I conflitti possono causare problemi come:

1. **Ambiguità**: se due regole di flusso specificano azioni diverse per lo stesso insieme di condizioni, il sistema può non sapere quale regola applicare.
2. **Loop di instradamento**: un conflitto può portare a loop infiniti in cui i dati vengono instradati ripetutamente tra gli stessi punti.
3. **Perdita di dati**: i conflitti possono causare la perdita di pacchetti di dati se il sistema scarta le informazioni non conformi ad una regola definita.
4. **Sovraccarico del sistema**: i conflitti possono causare un uso inefficiente delle risorse di rete, portando ad un sovraccarico e ad una diminuzione delle prestazioni del sistema.

L'idea progettuale considera le seguenti tipologie di conflitti:

Redundancy (*Ridondanza*)

Si verifica quando due o più regole di flusso sono essenzialmente identiche o si sovrappongono senza aggiungere valore significativo. Una regola r_i è ridondante rispetto a r_j se:

1. lo spazio degli indirizzi di ***i*** è un sottoinsieme dello spazio degli indirizzi di ***j***;
2. il protocollo è lo stesso;
3. l'azione è la stessa.

Shadowing (Ombreggiamento)

Si verifica quando una regola con priorità più alta "copre" o "ombreggia" una regola con priorità più bassa. In altre parole, la regola con priorità più alta si applica ai pacchetti che corrispondono anche alla regola con priorità più bassa, ma con un'azione diversa.
Una regola r_i è ombreggiata da r_j se:

1. la priorità di ***i*** è inferiore a quella di ***j***;
2. lo spazio degli indirizzi di ***i*** è un sottoinsieme dello spazio degli indirizzi di ***j***;
3. il protocollo è lo stesso;
4. l'azione è diversa.

Generalization (*Generalizzazione*)

La generalizzazione è il conflitto complementare allo *Shadowing*.
Una regola r_i è generalizzata dalla regola r_j se:

1. la priorità di ***i*** è maggiore rispetto a quella di ***j***;
2. *tutto* lo spazio degli indirizzi di ***j*** è un sottoinsieme dello spazio degli indirizzi di ***i***;
3. il protocollo è lo stesso;
4. l'azione è diversa.

Correlation (*Correlazione*)

Il conflitto si verifica quando due regole si applicano a spazi di indirizzi sovrapposti ma con due azioni diverse.
Una regola r_i è correlata a r_j se:

1. lo spazio degli indirizzi di ***i*** si interseca con lo spazio degli indirizzi di ***j***;
2. il protocollo è lo stesso;
3. la priorità è la stessa;
4. l'azione è diversa.

Overlap (Sovrapposizione)

Si verifica quando due regole si applicano a spazi di indirizzi sovrapposti e specificano la stessa azione.
Una regola r_i si sovrappone a r_j se:

1. lo spazio degli indirizzi di ***i*** si interseca con lo spazio degli indirizzi di ***j***;
2. il protocollo è lo stesso;
3. l'azione è la stessa.

4. Architettura

Il componente centrale è il controller ONOS, che aggiorna la topologia di rete in tempo reale, monitorando lo stato degli switch e dei link.
Il capitolo fornisce una panoramica ad alto livello sull'architettura dell'applicazione "**Conflict Detector**", presentando i due moduli "**Flow Extraction**" e "**Conflict Detection**", proseguendo poi con un'analisi approfondita delle classi che formano l'applicazione: **AppComponent**, **MyRule** e **MyScore**.

4.1 Flow Extraction

Il primo modulo, corrispondente alla prima porzione di codice della classe **AppComponent**, si occupa di estrarre le nuove regole immesse su ONOS. Intercetta tutte le regole iniettate nel controller e le inserisce all'interno di una struttura dati dove resteranno in attesa di essere processate. Di default, ONOS non accetta regole mal formate o con errori di sintassi, quindi ai fini del progetto si è supposto che l'estrazione della regola non debba occuparsi di verificare la sua correttezza.

4.2 Conflict Detection

Il modulo principale si occupa del rilevamento e gestione dei conflitti tra le regole inserite. Le regole vengono processate iterativamente e valutate in base ai diversi tipi di conflitti che potrebbero generare.

Ogni regola è associata a un indirizzo IP, che viene utilizzato per tracciare il comportamento dell'utente. In caso di conflitti, l'utente riceve una penalità crescente in base alla gravità e alla quantità di conflitti.

Ogni utente ha un livello di "**trustiness**" (*fiducia*), che viene ricalcolato a ogni iterazione del processo di valutazione. Il sistema applica un meccanismo di ricompensa e penalità per modificare dinamicamente il livello di fiducia, basato su parametri come il tempo e le azioni precedenti. Gli utenti con un livello di fiducia troppo basso verranno bloccati dopo più tentativi falliti, con un numero crescente di tentativi che porta a periodi di blocco progressivamente più lunghi. Il funzionamento del meccanismo di **trustiness** verrà approfondito ampiamente nel capitolo successivo.
L'obiettivo perseguito dall'idea progettuale è che si riesca a discriminare il comportamento malevolo di un ipotetico attaccante da un utente legacy, che potrebbe causare solo inavvertitamente un conflitto.

L’applicazione propone un modello di individuazione dell’utente malevolo basato sull’approccio a *black list*. Si memorizza una mappa che associa agli indirizzi IP degli utenti promotori un oggetto rappresentante un *punteggio*, che verrà descritto dettagliatamente nel capitolo seguente.
Se un utente genera un conflitto per la prima volta, il suo indirizzo IP viene aggiunto nella blacklist e viene bloccato per 2 secondi.
Se genera un conflitto due volte di fila, verrà bloccato per 4 secondi. Se genera tre conflitti di seguito verrà bloccato per 8 secondi.
Infine, se genera quattro conflitti di seguito verrà bloccato per sempre e il suo indirizzo IP non verrà mai rimosso dalla blacklist.

4.3 Struttura di AppComponent

La classe **AppComponent** racchiude la logica dell’applicazione ed è formata dai seguenti metodi:

Activate

L'annotazione *@Activate* indica che il metodo viene chiamato quando il componente viene attivato. L'applicazione viene registrata presso il *CoreService* con il nome "*Conflict Detector*": ogni applicazione ONOS ha bisogno di un *ApplicationId* univoco per gestire regole di flusso e altre operazioni di rete.

```
ApplicationId appId = coreService.registerApplication("Conflict
Detector");
flowRuleService.removeFlowRulesById(appId);
```

Un nuovo *thread* viene avviato per eseguire le operazioni di monitoraggio e gestione dei flussi.

```
new Thread(() -> {
    // codice interno
}).start();
```

Viene recuperata una lista di tutti i dispositivi presenti nella rete utilizzando *deviceService.getDevices()*.
Per ogni dispositivo, vengono recuperate le regole di flusso (*FlowEntry*) associate tramite *flowRuleService.getFlowEntries(d.id())*.

```
Iterable<Device> startDevices = deviceService.getDevices();
Iterable<FlowEntry> startRules = null;
```

```
for (Device d : startDevices) {
    startRules = flowRuleService.getFlowEntries(d.id());
}
```

Per ogni regola di flusso trovata in precedenza, viene creata una nuova istanza di *MyRule*, una classe personalizzata che gestisce i dettagli della regola di flusso.
La lista memorizza tutte le regole attuali nel sistema, così da poterle confrontare successivamente con nuove regole.

```
List<MyRule> startFlowList = new ArrayList<>();
for (FlowEntry f : startRules) {
    startFlowList.add(new MyRule(f.id().value(), f, false));
}
```

Successivamente vengono inizializzate le strutture dati:

- **conflictList**: contiene le regole di flusso che risultano in conflitto e che potrebbero essere rimosse.
- **flowList**: contiene le regole di flusso attualmente attive nel sistema.
- **users**: una mappa per tenere traccia degli utenti (identificati tramite indirizzo IP) e i loro punteggi.
- **bannedIp**: un insieme che contiene gli indirizzi IP che sono stati bannati a causa di comportamenti malevoli.
- **scheduler**: un esecutore programmato che gestisce il reinserimento degli indirizzi IP bannati dopo un certo periodo di tempo.

```
List<MyRule> conflictList = new ArrayList<>();
List<MyRule> flowList = new ArrayList<>();
HashMap<String, MyScore> users = new HashMap<>();
ScheduledExecutorService scheduler =
Executors.newScheduledThreadPool(1);
Set<String> bannedIp = new HashSet<>();
```

L'intero ciclo che monitora e gestisce le regole di flusso è implementato nel thread.
Prima di eseguire nuove operazioni, il sistema rimuove tutte le regole di flusso contrassegnate come "rimuovibili" (metodo *isRemovable()*).

```
while(true) {
    try {
        // Rimozione delle regole in conflitto
        for (MyRule f : conflictList) {
            if (f.isRemovable()) {
                flowList.remove(f);
```

```
            }
        }
        conflictList.clear();
```

Il sistema aggiorna la lista dei dispositivi e delle regole di flusso a ogni ciclo. Le regole che erano già presenti all'inizio dell'esecuzione vengono rimosse, in modo che solo le nuove regole vengano analizzate per eventuali conflitti.

```
Iterable<Device> devices = deviceService.getDevices();
Iterable<FlowEntry> rules = null;
for (Device d : devices) {
    rules = flowRuleService.getFlowEntries(d.id());
}
for (FlowEntry f : rules) {
    if(!flowList.contains(new MyRule(f.id().value(), f, false))){
        flowList.add(new MyRule(f.id().value(), f, false));
    }
}
flowList.removeAll(startFlowList);
```

Il cuore del metodo sta nella gestione dei conflitti (*Overlap*, *Redundancy*, *Shadowing*, *Generalization*, *Correlation*) attraverso la logica di incremento del *punishment*:

```
if (!overlap(startFlowList, f).isEmpty() && !f.isRemovable()) {
    punishment += 0.05;
    flowRuleService.removeFlowRules(f.getFlow());
    users.get(ip).startTimer();
    conflictList.add(f);
}
```

Ogni tipo di controllo verifica se la regola corrente (“*f*”) entra in conflitto con quelle iniziali.

- **Punizione**: se la regola è in conflitto, viene aumentato il livello di "*punishment*" associato all'utente che ha inserito la regola.
- **Rimozione della regola**: se viene rilevato un conflitto, la regola viene rimossa dal sistema.

La fiducia di un utente viene aggiornata ad ogni ciclo in base alla formula per il calcolo della *trustiness*.

```
users.get(ip).setPunishment(punishment);
     if(punishment < 0.0) {
     users.get(ip).setPunishment(0.0);
```

```
     }
     else if(punishment > 1.0) {
users.get(ip).setPunishment(1.0);
                        }

                        double oldTrustness =
users.get(ip).getTrustness();
                        double newTrustness = 1.0;

                        double oldAlpha = users.get(ip).getAlpha();
                        double newAlpha = 0.0;

                        if(oldTrustness < 0.0) {
                            oldTrustness = 0.0;
                        }else if(oldTrustness > 1.0) {
                            oldTrustness = 1.0;
                        }



                        // Calcolo della trustness
                        double beta = 0.001;
                        double lambda = 0.7;
                        newAlpha = oldAlpha * (1 + beta *
oldTrustness);
                        double reward = newAlpha *
(users.get(ip).getTime()) + 0.001;
                        if(reward > 1.0) {
                            reward = 1.0;
                        }
                        else if(reward < 0.0) {
                            reward = 0.0;
                        }
                newTrustness = oldTrustness + (reward - punishment);

                        if(newTrustness < 0.0) {
                            newTrustness = 0.0;
                        }else if(newTrustness > 1.0) {
                            newTrustness = 1.0;
                        }
```

```
                        users.get(ip).setTrustness(newTrustness);
                        users.get(ip).setPunishment(0.0);
                        users.get(ip).setAlpha(newAlpha);
```

Se la fiducia di un utente scende sotto una certa soglia, l'indirizzo IP viene bannato temporaneamente (gestito dallo "*scheduler*") o permanentemente.

```
//Meccanismo di blocco dell'IP sospetto
for (Map.Entry<String, MyScore> pair : users.entrySet()) {

if (pair.getKey().equals(ip) && pair.getValue().getTrustness() <
0.7) {
pair.getValue().setAttempts(pair.getValue().getAttempts() + 1);
if (pair.getValue().getAttempts() == 1) {
//log.info("L'utente con IP: {}" + ip + " viene riconosciuto come
attaccante per la prima volta!", ip);
flowRuleService.removeFlowRules(f.getFlow());
users.get(ip).setBannedRules(users.get(ip).getBannedRules() + 1);
//log.info("L'utente con IP: {}" + ip + " ha {} " +
users.get(ip).getBannedRules() + " regole bannate");
conflictList.add(f);
f.setRemovable(true);
bannedIp.add(pair.getKey());

scheduler.schedule(() -> {
bannedIp.remove(pair.getKey());
}, 2, TimeUnit.SECONDS);
break myLoop;
}
if (pair.getValue().getAttempts() == 2) {
//log.info("L'utente con IP: {}" + ip + " viene riconosciuto come
attaccante per la seconda volta!", ip);
flowRuleService.removeFlowRules(f.getFlow());
users.get(ip).setBannedRules(users.get(ip).getBannedRules() + 1);
//log.info("L'utente con IP: {}" + ip + " ha {} " +
users.get(ip).getBannedRules() + " regole bannate");
```

```
conflictList.add(f);
f.setRemovable(true);
bannedIp.add(pair.getKey());

scheduler.schedule(() -> {
bannedIp.remove(pair.getKey());
}, 4, TimeUnit.SECONDS);
break myLoop;
}
if (pair.getValue().getAttempts() == 3) {
//log.info("L'utente con IP: {}" + ip + " viene riconosciuto come
attaccante per la terza volta!", ip);
flowRuleService.removeFlowRules(f.getFlow());
users.get(ip).setBannedRules(users.get(ip).getBannedRules() + 1);
//log.info("L'utente con IP: {}" + ip + " ha {} " +
users.get(ip).getBannedRules() + " regole bannate");
conflictList.add(f);
f.setRemovable(true);
bannedIp.add(pair.getKey());

scheduler.schedule(() -> {
bannedIp.remove(pair.getKey());
}, 8, TimeUnit.SECONDS);
break myLoop;
}
if (pair.getValue().getAttempts() >= 4) {
//log.info("L'utente con IP: {}" + ip + " viene riconosciuto come
attaccante, sarà bloccato per sempre!", ip);
flowRuleService.removeFlowRules(f.getFlow());
users.get(ip).setBannedRules(users.get(ip).getBannedRules() + 1);
log.info("L'utente con IP: {} " + ip + " ha " +
users.get(ip).getBannedRules() + " regole bannate");
conflictList.add(f);
f.setRemovable(true);
bannedIp.add(pair.getKey());
break myLoop;
}
}
}
```

longToIp

Questo metodo converte un indirizzo IP rappresentato come stringa in un formato leggibile.

subnet

Gestisce il controllo di sottoreti, applicando maschere di rete a indirizzi IP per determinare se appartengono alla stessa rete.

4.4 Struttura di MyRule

A causa dell'interazione limitata presente nelle regole di flusso proposte da ONOS, è stata sviluppata una classe personalizzata *MyRule* con l'obiettivo di aggiungere delle funzionalità per adattarsi meglio allo scopo dell'idea progettuale.

La classe presenta:

- Un campo *id*, con analoghe funzioni dell'identificativo della regola di flusso nativa di ONOS;
- Un campo *flow*, che integra la regola di flusso di ONOS;
- Un campo *removable*, con lo scopo di impostare la rimozione della regola.

Il resto della classe presenta il costruttore, i metodi accessori, l'*equals* e l'*hashCode*.

4.5 Struttura di MyScore

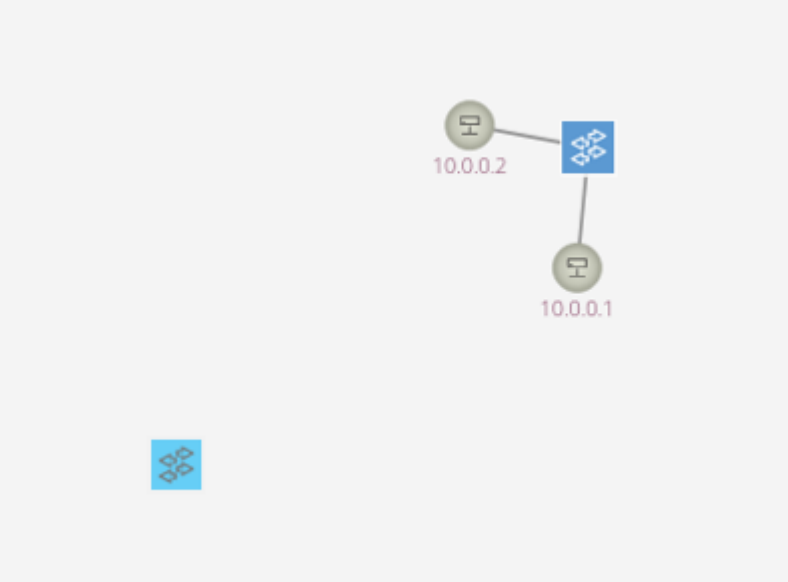
La classe MyScore definisce un oggetto che tiene traccia di una serie di metriche relative al comportamento di un utente. I campi caratterizzanti sono:

- *trustiness*: il valore rappresenta il "livello di fiducia" dell'utente;
- *alpha*: un parametro che rappresenta il coefficiente di guadagno che influenza il calcolo della fiducia;
- *punishment*: un valore che rappresenta la quantità di "punizione" inflitta all'utente;
- *attempts*: tiene traccia del numero di tentativi da parte dell'utente di caricare regole in conflitto o regole dannose;
- *loadedRules*: conta il numero di regole che l'utente ha caricato con successo;
- *bannedRules*: conta il numero di regole che sono state rimosse dal sistema;
- *startTime*: il tempo relativo all'inizio del monitoraggio dell'utente.

Il resto della classe presenta i costruttori, i metodi accessori, e un metodo *getTime()* che calcola il tempo trascorso in secondi dall'ultimo reset del timer, impiegato per tenere traccia del tempo che passa da quando l'utente è stato penalizzato per l'ultima volta.

4.6 Use Case

La topologia predefinita è quella minima, che comprende uno switch kernel OpenFlow (identificato con "*s1*") collegato a due host ("*h1*" e "*h2*"), oltre al controller di riferimento ("*c0*").



Con il comando:

```
sudo mn --mac --switch ovsk,protocols=OpenFlow13 --
controller=remote, ip=127.0.0.1, port=6653 --link=tc
```

è possibile avviare la CLI di Mininet. La topologia è adesso visualizzabile con il comando *net*.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Il caricamento delle regole di flusso avviene tramite il comando *curl*. Le regole sono state caricate singolarmente (per verificare la presenza di conflitti) o mediante script bash *addRule.sh* per il testing del sistema in risposta a un'elevata quantità di regole:

```
#!/bin/bash

URL="http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002"
```

```
AUTH="onos:rocks"
CONTENT_TYPE="application/json"
ACCEPT="application/json"

generate_rule() {
  local priority=$1
  local ethType=$2
  local inPort=$3
  local protocol=$4
  local ipv4Src=$5
  local ipv4Dst=$6
  local metadata=$7
  local outputPort=$8

  if [ -n "$outputPort" ]; then
    instructions="{\"type\": \"OUTPUT\", \"port\": \"$outputPort\"}"
  else
    instructions=""
  fi

  curl -X POST -u $AUTH \
    --header "Content-Type: $CONTENT_TYPE" \
    --header "Accept: $ACCEPT" \
    -d "{
      \"priority\": $priority,
      \"timeout\": 0,
      \"isPermanent\": true,
      \"deviceId\": \"of:0000000000000002\",
      \"treatment\": {
        \"instructions\": [$instructions]
      },
      \"selector\": {
        \"criteria\": [
          {\"type\": \"ETH_TYPE\", \"ethType\": \"$ethType\"},
          {\"type\": \"IN_PORT\", \"port\": \"$inPort\"},
          {\"type\": \"IP_PROTO\", \"protocol\": $protocol},
          {\"type\": \"IPV4_SRC\", \"ip\": \"$ipv4Src\"},
          {\"type\": \"IPV4_DST\", \"ip\": \"$ipv4Dst\"},
          {\"type\": \"METADATA\", \"metadata\": \"$metadata\"}
```

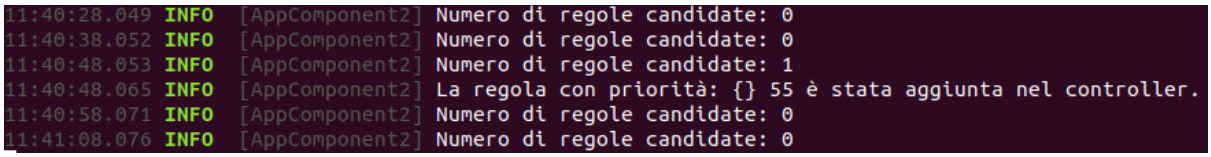
```
        ]
      }
    }" $URL
}

priority=60
for i in {1..95}; do
  generate_rule $priority "0x0800" "3" 6 "10.5.50.$i/32"
"10.211.1.$((i % 64))/32" "16777216$((i % 10))" ""
  priority=$((priority + 1))
done
```

1° caso: Funzionamento standard

L'applicazione è stata caricata e attivata su ONOS.
Nello **Use Case** base di funzionamento supponiamo che ogni regola caricata non generi conflitto. Un esempio di regola (da caricare con il comando *curl* mediante chiamata REST all'API di ONOS) è:

```
curl -X POST -u onos:rocks --header "Content-Type: application/json"
--header "Accept: application/json" -d '{
  "priority": 55,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000002",
  "treatment": {
     },
  "selector": {
     "criteria": [
     {
     "type": "ETH_TYPE",
     "ethType": "0x0800"
     }
     ]
  }
}' "http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002"
```

La regola viene processata da Conflict Detector e aggiunta senza creare conflitti.



2° caso: Presenza di conflitti

Nel secondo scenario proposto dall'idea progettuale, le regole candidate generano conflitti con una o più regole già presenti nel controller. Il paragrafo si occuperà di fornire una panoramica sui vari conflitti per come sono stati discussi precedentemente, con degli esempi pratici di regole di flusso che li causano.

Al fine di distinguere nella stampa le regole che generano conflitti, si è scelto di utilizzare la priorità (come elemento distintivo della regola), dato che può essere scelta dall'utente ed è più semplice da visualizzare.

Overlap

Un conflitto di tipo *overlap* (sovrapposizione) è generato come descritto di seguito.

La prima regola di flusso aggiunta nel controller è:

```
curl -X POST -u onos:rocks --header "Content-Type: application/json"
--header "Accept: application/json" -d '{
  "priority": 51,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000002",
   "treatment": {
    "instructions": [

    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
{
        "type": "IN_PORT",
        "port": "3"
      },
```

```
   {
        "type": "IP_PROTO",
        "protocol": 6
      },
  {
        "type": "IPV4_SRC",
        "ip": "10.5.50.0/32"
      },
{
        "type": "IPV4_DST",
        "ip": "10.211.1.63/32"
      },

{
"type": "METADATA",
 "metadata": "167772164"
}

    ]
  }
}' "http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002"
```

L'inserimento della regola va a buon fine e l'output generato è il seguente:

```
11:40:28.049 INFO  [AppComponent2] Numero di regole candidate: 0
11:40:38.052 INFO  [AppComponent2] Numero di regole candidate: 0
11:40:48.053 INFO  [AppComponent2] Numero di regole candidate: 1
11:40:48.065 INFO  [AppComponent2] La regola con priorità: {} 55 è stata aggiunta nel controller.
11:40:58.071 INFO  [AppComponent2] Numero di regole candidate: 0
11:41:08.076 INFO  [AppComponent2] Numero di regole candidate: 0
```

Per generare *overlap*, la seconda regola candidata da aggiungere deve avere la stessa azione, lo stesso protocollo della prima e lo spazio degli indirizzi deve intersecarsi.

```
curl -X POST -u onos:rocks --header "Content-Type: application/json"
--header "Accept: application/json" -d '{
  "priority": 55,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000002",
   "treatment": {
    "instructions": [
```

```
    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
{
        "type": "IN_PORT",
        "port": "3"
      },
   {
        "type": "IP_PROTO",
        "protocol": 6
      },
  {
        "type": "IPV4_SRC",
        "ip": "10.5.50.5/32"
      },
{
        "type": "IPV4_DST",
        "ip": "10.211.1.0/24"
      },

{
"type": "METADATA",
 "metadata": "167772163"
}

    ]
  }
}' "http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002"
```

L'applicazione riconosce il conflitto e visualizza in output:

```
7:33:36.129 INFO  [AppComponent] Numero di regole candidate: 1
7:33:40.132 INFO  [AppComponent] La regola con priorità: {} 55 è sovrapposta alla regola con priorità: {}51
```

Redundancy

Un conflitto di tipo *redundancy* (ridondanza) è generato nel seguente modo.

La prima regola di flusso aggiunta nel controller è:

```
curl -X POST -u onos:rocks --header "Content-Type: application/json"
--header "Accept: application/json" -d '{
  "priority": 51,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000002",
   "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "4"
      }
    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
      {
        "type": "IN_PORT",
        "port": "3"
      },
      {
        "type": "IP_PROTO",
        "protocol": 6
      },
        {
        "type": "IPV4_SRC",
        "ip": "10.5.50.0/24"
      },
      {
        "type": "IPV4_DST",
```

```
        "ip": "10.211.1.63/32"
      },

      {
            "type": "METADATA",
            "metadata": "167772161"
      }

    ]
  }
}' "http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002"
```

L'inserimento della regola va a buon fine e l'output visualizzato è il seguente:

```
12:01:38.875 INFO  [AppComponent2] Numero di regole candidate: 0
12:01:48.882 INFO  [AppComponent2] Numero di regole candidate: 0
12:01:58.896 INFO  [AppComponent2] Numero di regole candidate: 1
12:01:58.900 INFO  [AppComponent2] La regola con priorità: {} 51 è stata aggiunta nel controller.
```

Per generare un conflitto di tipo *redundancy*, è necessario che il protocollo e l'azione corrispondano e che lo spazio degli indirizzi della prima regola sia un sottoinsieme dello spazio degli indirizzi della seconda. Nel caso in esame, ad esempio, l'indirizzo IP sorgente della prima regola è *10.5.50.5/32,* ovvero un sottoinsieme dell'indirizzo sorgente della seconda (*10.5.50.0/24*).

```
curl -X POST -u onos:rocks --header "Content-Type: application/json"
--header "Accept: application/json" -d '{
  "priority": 50,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000002",
   "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "4"
      }
    ]
  },
```

```
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
{
        "type": "IN_PORT",
        "port": "3"
      },
   {
        "type": "IP_PROTO",
        "protocol": 6
      },
  {
        "type": "IPV4_SRC",
        "ip": "10.5.50.5/32"
      },
{
        "type": "IPV4_DST",
        "ip": "10.211.1.1/32"
      },

{
"type": "METADATA",
 "metadata": "167772162"
}

    ]
  }
}' "http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002"
```

L'applicazione riconosce il conflitto e visualizza in output:

```
11:36:30.559 INFO  [AppComponent2] La regola con priorità: {} 51 è stata aggiunta nel controller.
11:36:40.561 INFO  [AppComponent2] Numero di regole candidate: 0
11:36:50.565 INFO  [AppComponent2] Numero di regole candidate: 1
11:36:50.579 INFO  [AppComponent2] La regola con priorità: {} 50 è ridondante con la regola con priorità: {}51
```

Shadowing

Per generare un conflitto *shadowing* (ombreggiamento) le due regole devono avere lo stesso protocollo, azioni differenti e appartenere allo stesso spazio di indirizzi.

La prima regola inserita è:

```
curl -X POST -u onos:rocks --header "Content-Type: application/json"
--header "Accept: application/json" -d '{
  "priority": 51,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000002",
   "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "3"
      }
    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
      {
        "type": "IN_PORT",
        "port": "3"
      },
      {
        "type": "IP_PROTO",
        "protocol": 6
      },
        {
        "type": "IPV4_SRC",
        "ip": "10.5.50.0/24"
      },
```

```
      {
         "type": "IPV4_DST",
         "ip": "10.211.1.63/32"
       },

      {
            "type": "METADATA",
            "metadata": "167772161"
      }

    ]
  }
}' "http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002"
```

```
12:37:13.480 INFO  [AppComponent2] Numero di regole candidate: 0
12:37:23.481 INFO  [AppComponent2] Numero di regole candidate: 1
12:37:23.491 INFO  [AppComponent2] La regola con priorità: {} 51 è stata aggiunta nel controller.
```

La seconda regola candidata, inoltre, ha una priorità maggiore della precedente (53 > 51): è quindi possibile concludere che la prima regola viene ombreggiata dalla seconda.

```
curl -X POST -u onos:rocks --header "Content-Type: application/json"
--header "Accept: application/json" -d '{
  "priority": 53,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000002",
   "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "5"
      }
    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
```

```
      },
      {
         "type": "IN_PORT",
         "port": "3"
      },
      {
         "type": "IP_PROTO",
         "protocol": 6
      },
        {
         "type": "IPV4_SRC",
         "ip": "10.5.50.0/24"
      },
      {
         "type": "IPV4_DST",
         "ip": "10.211.1.63/32"
      },

      {
            "type": "METADATA",
            "metadata": "167772165"
      }

    ]
  }
}' "http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002"
```

L'applicazione riconosce il conflitto e l'output restituito è:

```
12:37:13.480 INFO  [AppComponent2] Numero di regole candidate: 0
12:37:23.481 INFO  [AppComponent2] Numero di regole candidate: 1
12:37:23.491 INFO  [AppComponent2] La regola con priorità: {} 51 è stata aggiunta nel controller.
12:37:33.495 INFO  [AppComponent2] Numero di regole candidate: 0
12:37:43.499 INFO  [AppComponent2] Numero di regole candidate: 1
12:37:43.528 INFO  [AppComponent2] La regola con priorità: {} 54 ombreggia la regola con priorità: {}51
```

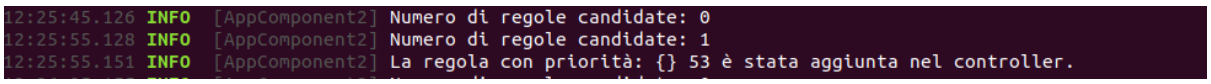
Correlation

Un conflitto di tipo *correlation* (correlazione) si genera aggiungendo come prima regola di flusso:

```
curl -X POST -u onos:rocks --header "Content-Type: application/json"
--header "Accept: application/json" -d '{
  "priority": 51,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000002",
   "treatment": {
    "instructions": [
       {
        "type": "OUTPUT",
        "port": "3"
      }
    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
      {
        "type": "IN_PORT",
        "port": "3"
      },
      {
        "type": "IP_PROTO",
        "protocol": 6
      },
       {
        "type": "IPV4_SRC",
        "ip": "10.5.50.0/24"
      },
      {
        "type": "IPV4_DST",
        "ip": "10.211.1.63/32"
```

```
      },

      {
            "type": "METADATA",
            "metadata": "167772161"
      }

    ]
  }
}' http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002
```

L'aggiunta della prima regola candidata va a buon fine.



Per generare *correlation*, la seconda regola candidata da aggiungere deve avere la stessa priorità e lo stesso protocollo della prima, le azioni devono essere diverse e lo spazio degli indirizzi deve intersecarsi.

```
curl -X POST -u onos:rocks --header "Content-Type: application/json"
--header "Accept: application/json" -d '{
  "priority": 51,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000002",
   "treatment": {
     "instructions": [
    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
      {
        "type": "IN_PORT",
        "port": "3"
```

```
      },
      {
         "type": "IP_PROTO",
         "protocol": 6
      },
        {
         "type": "IPV4_SRC",
         "ip": "10.5.50.0/16"
      },
      {
         "type": "IPV4_DST",
         "ip": "10.211.1.63/32"
      },

      {
            "type": "METADATA",
            "metadata": "167772161"
      }

    ]
  }
}' http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002
```

L'output visualizzato è:

```
12:26:15.160 INFO  [AppComponent2] Numero di regole candidate: 1
12:26:15.166 INFO  [AppComponent2] La regola con priorità: {} 52 è correlata alla regola con priorità: {} 53
```

Generalization

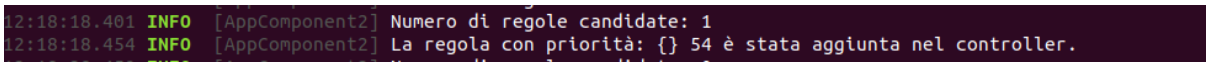
Per generare un conflitto di *generalization* (generalizzazione) è necessario che le due regole abbiano lo stesso protocollo, azioni differenti e debbano appartenere allo stesso spazio degli indirizzi.

La prima regola proposta è:

```
curl -X POST -u onos:rocks --header "Content-Type: application/json"
--header "Accept: application/json" -d '{
  "priority": 54,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000002",
   "treatment": {
    "instructions": [

    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
{
        "type": "IN_PORT",
        "port": "3"
      },
   {
        "type": "IP_PROTO",
        "protocol": 6
      },
  {
        "type": "IPV4_SRC",
        "ip": "10.5.50.5/32"
      },
{
        "type": "IPV4_DST",
        "ip": "10.211.1.63/32"
```

```
      },

{
"type": "METADATA",
 "metadata": "167772164"
}

    ]
  }
}' http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002
```

L'output visualizzato è:



La seconda regola candidata è:

```
curl -X POST -u onos:rocks --header "Content-Type: application/json"
--header "Accept: application/json" -d '{
  "priority": 51,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000002",
   "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "4"
      }
    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
{
        "type": "IN_PORT",
        "port": "3"
```

```
      },
   {
        "type": "IP_PROTO",
        "protocol": 6
      },
  {
        "type": "IPV4_SRC",
        "ip": "10.5.50.0/24"
      },
{
        "type": "IPV4_DST",
        "ip": "10.211.1.63/32"
      },

{
"type": "METADATA",
 "metadata": "167772164"
}

    ]
  }
}' http://127.0.0.1:8181/onos/v1/flows/of%3A0000000000000002
```

L'output restituito è:

```
12:19:08.494 INFO  [AppComponent2] Numero di regole candidate: 0
12:19:18.497 INFO  [AppComponent2] Numero di regole candidate: 1
12:19:18.522 INFO  [AppComponent2] La regola con priorità: {} 51 generalizza la regola con priorità: {}54
```

3° caso: Presenza di un attaccante

L'assunzione di base del terzo *use case* è che l'indirizzo IP sorgente dell'utente che propone una regola sia contenuto all'interno del campo *METADATA* della stessa. In particolare, come richiesto da ONOS, l'indirizzo è memorizzato come Long:

```
{
"type": "METADATA",
 "metadata": "167772161"
}
```

dalla pagina web https://ipaddress.standingtech.com/online-long-integer-to-ip-converter è possibile verificare che l'indirizzo appena presentato corrisponda all'IP "10.0.0.1". All'interno dell'applicazione è presente il metodo *longToIp* che si occupa di riconvertire il Long in stringa:

```
public String longToIp(String ipString) {
    long ip = Long.parseLong(ipString.split( regex: ":")[1],  radix: 16);
    return ((ip >> 24) & 0xFF) + "."
            + ((ip >> 16) & 0xFF) + "."
            + ((ip >> 8) & 0xFF) + "."
            + (ip & 0xFF);
}
```

Lo scenario è quello in cui diversi utenti sono a conoscenza delle credenziali *onos:rocks* per poter proporre una regola.

5. Testing

org.onosproject.cpman è un modulo del progetto ONOS specificamente legato alla gestione delle prestazioni e al monitoraggio del sistema.

Per avviarlo occorre eseguire:

```
sudo /opt/onos/bin/onos -l onos
app activate org.onosproject.cpman
```

Sono state analizzate le variazioni dell'uso di memoria RAM prima e dopo la generazione di un conflitto di *Overlap*.

```
onos > cpman-stats-list 10.0.2.15 memory
```

Primo output (in assenza di conflitti):

- **MEMORY_FREE**: 112.959.488 byte (circa 108 MB)
- **MEMORY_FREE_RATIO**: 3%
- **MEMORY_USED**: 2.830.942.208 byte (circa 2,64 GB)
- **MEMORY_USED_RATIO**: 95%

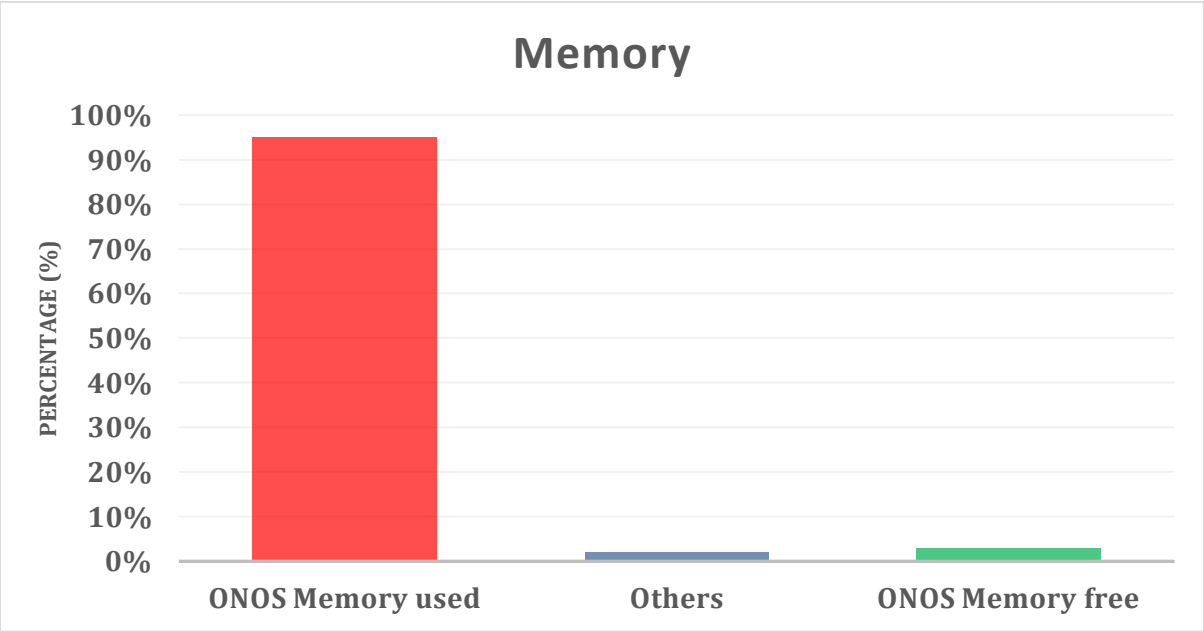


Figura 2: Memoria usata – senza conflitti

Secondo output (conflitto in corso):

- **MEMORY_FREE**: 118.239.232 byte (circa 113 MB)
- **MEMORY_FREE_RATIO**: 4%
- **MEMORY_USED**: 2.817.933.312 byte (circa 2,63 GB)
- **MEMORY_USED_RATIO**: 95%

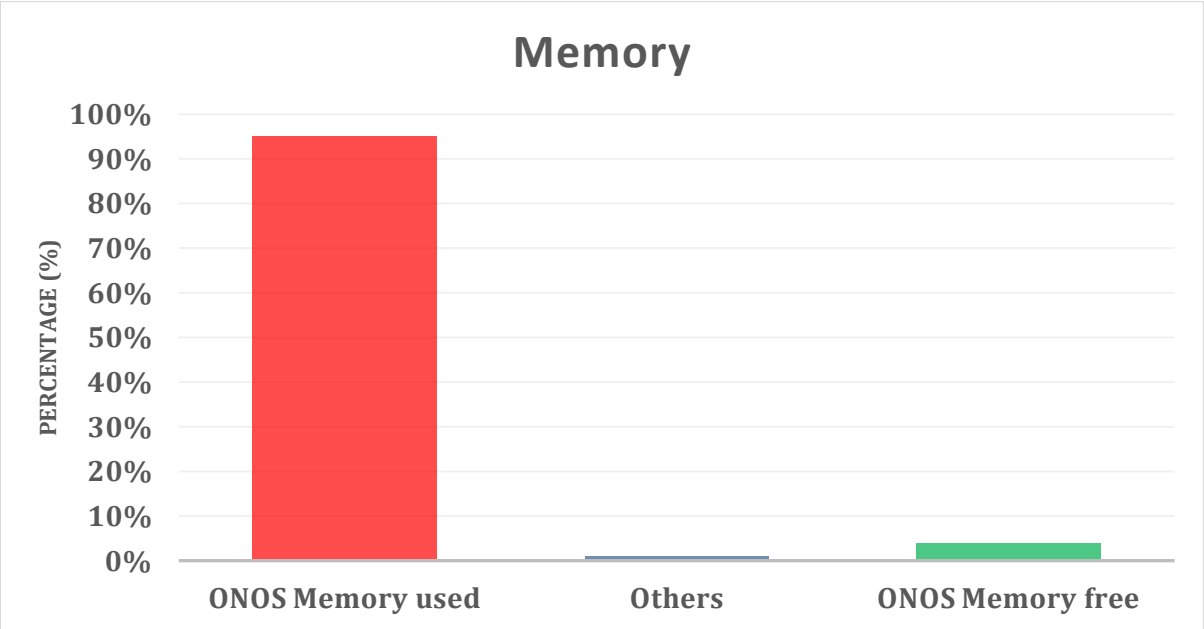


Figura 3: Memoria usata – con conflitti

Il rapporto di memoria utilizzata è rimasto al 95%, suggerendo che con la presenza di un solo conflitto la variazione è minima rispetto alla capacità totale di memoria.

Sono state anche analizzate le variazioni delle metriche di utilizzo della CPU prima e dopo l'inserimento di 95 regole, alcune delle quali in conflitto.
È stato utilizzato il comando:

```
onos > cpman-stats-list 10.0.2.15 cpu
```

Primo Output (senza regole conflittuali)
metricType=**CPU_IDLE_TIME**, latestValue=91, averageValue=0, latestTime=1723975433
metricType=**CPU_LOAD**, latestValue=71, averageValue=0, latestTime=1723975433
metricType=**SYS_CPU_TIME**, latestValue=3, averageValue=0, latestTime=1723975433
metricType=**USER_CPU_TIME**, latestValue=4, averageValue=0, latestTime=1723975433
metricType=**TOTAL_CPU_TIME**, latestValue=0, averageValue=0, latestTime=1723975433

Secondo Output (dopo l'inserimento delle regole)
metricType=**CPU_IDLE_TIME**, latestValue=93, averageValue=0, latestTime=1723976693
metricType=**CPU_LOAD**, latestValue=189, averageValue=2, latestTime=1723976693
metricType=**SYS_CPU_TIME**, latestValue=2, averageValue=0, latestTime=1723976693
metricType=**USER_CPU_TIME**, latestValue=8, averageValue=0, latestTime=1723976693
metricType=**TOTAL_CPU_TIME**, latestValue=0, averageValue=0, latestTime=1723976693

Il carico della CPU è aumentato drasticamente da 71 a 189. Quindi, i conflitti nell'inserimento delle regole hanno causato un aumento del carico sulla CPU e delle risorse utilizzate, confermando che l'elaborazione delle regole conflittuali ha richiesto una quantità significativa di tempo di CPU.

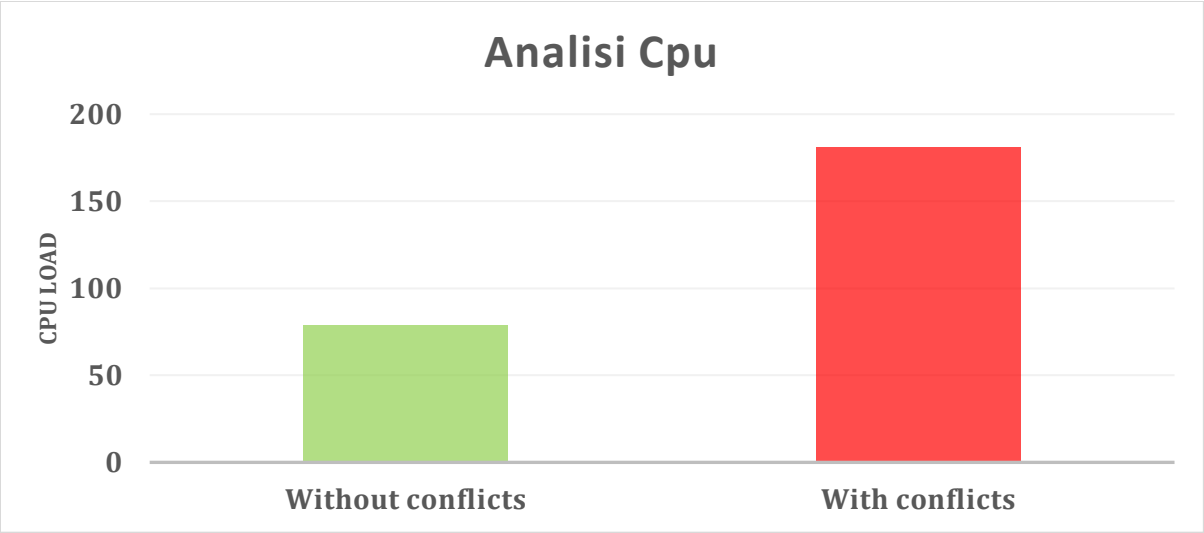


Figura 4: Confronto CPU

È stato effettuato un ulteriore confronto tra l'inserimento di regole conflittuali con l'applicazione Conflict Resolver e l'inserimento di regole senza l'applicazione.

L'output riscontrato è il seguente:

```
metricType=CPU_IDLE_TIME, latestValue=98, averageValue=0,
latestTime=1723978690
metricType=CPU_LOAD, latestValue=146, averageValue=0,
latestTime=1723978690
metricType=SYS_CPU_TIME, latestValue=0, averageValue=0,
latestTime=1723978690
metricType=USER_CPU_TIME, latestValue=22, averageValue=0,
latestTime=1723978690
```

```
metricType=TOTAL_CPU_TIME, latestValue=0, averageValue=0,
latestTime=1723978690
```

Il confronto più significativo riguarda il carico della CPU.

Output con l'applicazione: 189
Output senza l'applicazione: 146

Quindi, il carico della CPU è diminuito dal 189 al 146. Sebbene il carico sia ancora relativamente alto rispetto ai valori normali, esso è notevolmente ridotto rispetto al picco osservato precedentemente: ciò è dovuto al fatto che ONOS applica la propria politica di risoluzione dei conflitti (sovrascrive con l'ultima regola inserita), mentre l'applicazione gestisce tutti i tipi di conflitti.

5.1 Trustiness

Nel sistema in esame, la fiducia (o "*Trustiness*") viene valutata dinamicamente per monitorare e gestire il comportamento dei nodi coinvolti. Per garantire una gestione adeguata delle risorse e mantenere un elevato livello di cooperazione tra i partecipanti, è stato implementato un meccanismo di fiducia basato su un bilanciamento di ricompensa e punizione. Questo approccio si fonda su un'analisi costante delle azioni dei nodi e su una valutazione delle loro prestazioni in base a criteri predefiniti.

La fiducia di un nodo è un numero double definito tra 0 e 1. Viene inizialmente impostata a un valore neutrale (1) e poi aggiornata dinamicamente durante l'interazione con altri nodi del sistema. Ogni interazione è soggetta a valutazione: se un nodo esegue correttamente il proprio compito, contribuisce positivamente al sistema e, di conseguenza, ottiene una ricompensa. Viceversa, se il nodo esegue male il compito o agisce in modo dannoso, subisce una punizione. Anche i valori di *Punishment* e *Reward* variano tra 0 e 1.

La formula generale utilizzata per aggiornare il livello di fiducia T_S di un nodo S è la seguente:

$$T_S\,(t) \;=\; T_S\,(t-1) \;+\; (R_S(t) - P_S(t))$$

Dove:

- $T_S(t)$ rappresenta il livello di fiducia della sorgente S al tempo t;

- $R_S(t)$ è la ricompensa assegnata alla sorgente S al tempo t;
- $P_S(t)$ è la punizione assegnata alla sorgente S al tempo t e consiste nella somma dei conflitti generati in una finestra temporale;

Quando la somma dei conflitti è pari a zero, allora si ha una situazione di *Full Reward*. Quando la somma dei conflitti è maggiore del *Reward*, allora la differenza $\big(R_S(t) - P_S(t)\big)$ è un numero negativo. Infine, quando la somma dei conflitti è minore del valore di *Reward*, si ottiene una situazione di *Partial Reward*.

5.2 Punishment

Il meccanismo di *Punishment* assegna un punteggio da 0.05 a 0.5 ad ogni nodo. La severità della punizione dipende dalla gravità del comportamento negativo.
I conflitti con i loro punteggi sono riassunti nella seguente tabella:

CONFLITTI	**PUNTEGGI**
Shadowing	0.5
Correlation	0.3
Generalization	0.2
Redundancy	0.1
Overlap	0.05

L'algoritmo tiene conto della storia pregressa di ogni nodo, favorendo un approccio flessibile che consente un adattamento dinamico della fiducia in base all'evoluzione del comportamento del nodo nel tempo.

5.3 Reward

Le ricompense vengono attribuite ai nodi che operano in modo affidabile. Questo comporta un incremento del loro punteggio di fiducia, incentivando un comportamento virtuoso. La formula utilizzata per calcolare il *Reward* di un nodo S è:

$$Rs(t) = \alpha * \delta(t)$$

Dove α è un coefficiente di guadagno e δ è l'intervallo di tempo in cui l'utente si comporta correttamente.
Il valore di $\alpha(0)$, cioè alfa al tempo $t = 0$, è 0.01. In generale, la formula utilizzata per il calcolo di α è:

$$\alpha(t) = \alpha(t-1) * (1 + \beta * T_S(t-1))$$

Dove $\beta(t)$ è un coefficiente di sensibilità che controlla quanto velocemente α si adatta alle variazioni. Nel caso in questione, β = 0.01.

5.4 Threshold

Al fine di scegliere il valore ottimale per la soglia ("*Threshold*") usata per determinare se un nodo ha un comportamento malevolo, sono stati condotti una serie di esperimenti per valutarne il valore scelto rispetto alla metrica F1-Score.
L'F1-Score è definito come la media armonica tra la *precisione* (P) e il *recupero* (R).
Sia la precisione che il recupero sono numeri compresi tra 0 e 1.
Il punteggio F1 è, quindi, numero compreso tra 0 e 1 che permette di catturare l'andamento della precisione e del recupero in un unico valore.
Poiché l'obiettivo è quello di trovare il valore di soglia che può massimizzare il valore dell'F1-Score, sono stati testati diversi valori per la soglia utilizzata per identificare un attaccante, eseguendo simulazioni che consideravano due scenari, con 5 attaccanti su 10 utenti in totale e con 10 attaccanti su 10 utenti in totale.
Il miglior valore di F1-Score si ottiene quando la soglia è impostata a 0.7, raggiungendo un valore di picco per l'F1-Score pari al 88%.
I valori inferiori a 0.7, in particolare con la soglia pari a 0.5, mostrano le prestazioni peggiori. Più in particolare, valori piccoli rendono difficile soddisfare l'equazione della trustiness. Ciò significa che le sorgenti malevole non sono classificate correttamente come tali e che gli FN aumentano.
Al contrario, quando i valori della soglia sono maggiori di 0.7, in particolare con 0.9, l'equazione della trustiness è più facile da soddisfare, il che porta a un aumento del numero di FP.

Veri Positivi	Veri Negativi	Falsi Positivi	Falsi Negativi		Accuratezza	Prob. di falso allarme			Prob. di mancato allarme
8	0	0	2		0,8	0			1
N° di attaccanti	**N° di utenti**	**F1-Score**			**F1-Score**	**Precisione**			**Recupero**
5	10	0,888888889			0,888888889	1			0,8
10	10	0,888888889							

Figura 5: F1-Score con soglia=0.7

Veri Positivi	Veri Negativi	Falsi Positivi	Falsi Negativi		Accuratezza	Prob. di falso allarme			Prob. di mancato allarme
7	0	0	3		0,7	0			1
N° di attaccanti	**N° di utenti**	**F1-Score**			**F1-Score**	**Precisione**			**Recupero**
5	10	0,75			0,823529412	1			0,7
10	10	0,823529412							

Figura 6: F1-Score con soglia=0.5

Veri Positivi	Veri Negativi	Falsi Positivi	Falsi Negativi		Accuratezza	Prob. di falso allarme			Prob. di mancato allarme
9	0	0	1		0,9	0			1
N° di attaccanti	N° di utenti	F1-Score			F1-Score	Precisione			Recupero
5	10	0,833333333			0,947368421	1			0,9
10	10	0,947368421							

Figura 7: F1-Score con soglia=0.9

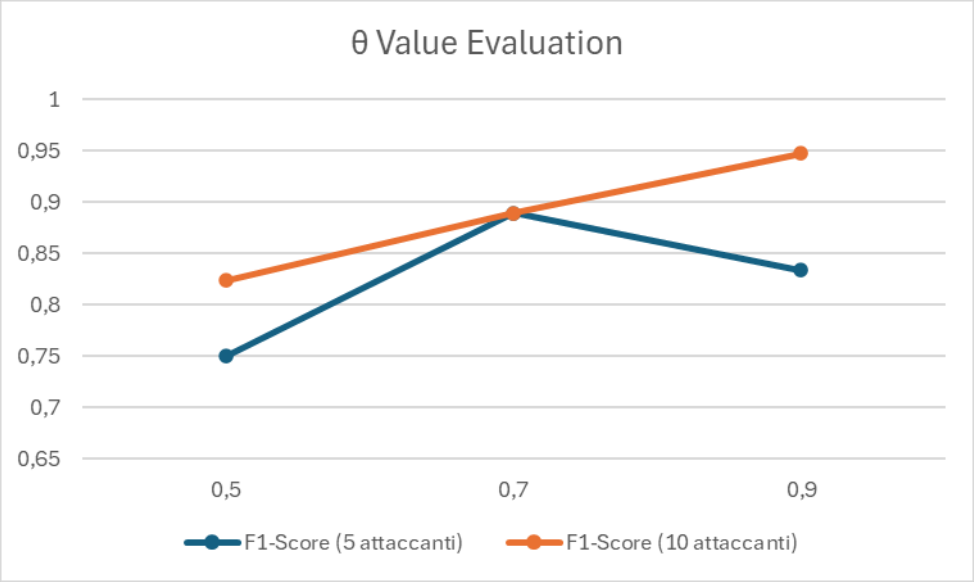


Figura 8: Confronto tra soglia e F1-Score

Tempo di detection

Per mostrare quanto il sistema impiega a rilevare un attaccante e ad eliminarlo dalla rete sono stati generati diversi scenari.

1) Il primo scenario è quello in cui il 60% dei conflitti totali è composto da quelli più gravi, *Shadowing* e *Correlation*.
2) Il secondo scenario è quello in cui il 60% dei conflitti totali è composto da quelli meno gravi, *Redundancy* e *Overlap*.
3) Nel terzo scenario, c'è una distribuzione randomica dei conflitti, in cui tutti i conflitti sono equiprobabili.
4) Nel quarto scenario, ogni conflitto si verifica il 20% delle volte.

I test sono stati effettuati considerando un valore λ , che permette di pesare la considerazione che si ha della storia presente e passata. La formula della trustiness utilizzata è la seguente:

$$T_S\,(t) = \lambda * T_S\,(t-1) + (1-\lambda) * (Rs(t) - Ps(t))$$

In uno scenario con un valore di λ pari a 0.5, si tiene in considerazione la storia recente tanto quanto quella passata. Quando λ è uguale 0.3, si pesa di più la storia recente; l'opposto avviene quando λ è pari a 0.7.
Nelle tre figure che seguono sono mostrati i risultati dei test con i seguenti valori di lambda. Quello che si deduce è che con valori di λ più piccoli, il tempo di rilevamento di un attaccante è più basso. Questo avviene perché il sistema "dimentica" più rapidamente le informazioni passate e reagisce più velocemente ai cambiamenti. Di conseguenza, se c'è un attaccante, il sistema ne rileverà il comportamento più velocemente, poiché la penalità avrà un impatto più immediato sulla trustiness.

	time [ms]
20% for each conflict	50365
60% Shadowing&Correlation	80108
60% Redudancy&Overlap	94122
Equiprobable conflicts	78747

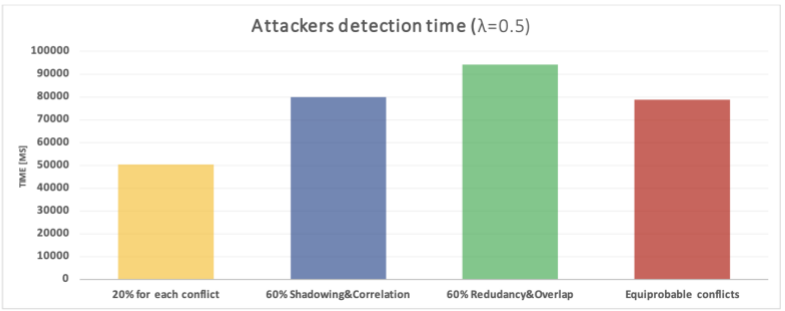


Figura 9: detection time con $\lambda = 0.5$

	time [ms]
20% for each conflict	44080
60% Shadowing&Correlation	70783
60% Redudancy&Overlap	85503
Equiprobable conflicts	59144

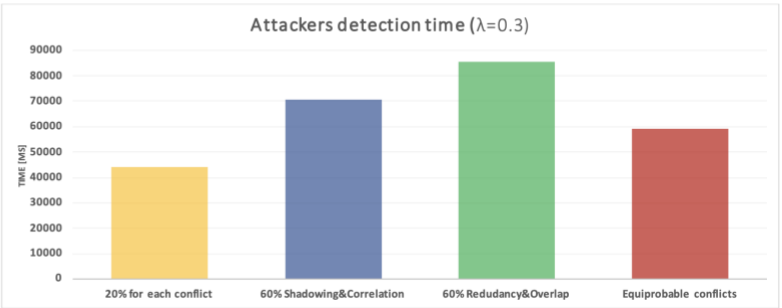


Figura 10: detection time con $\lambda = 0.3$

	time [ms]
20% for each conflict	54576
60% Shadowing&Correlation	110384
60% Redudancy&Overlap	120912
Equiprobable conflicts	119812

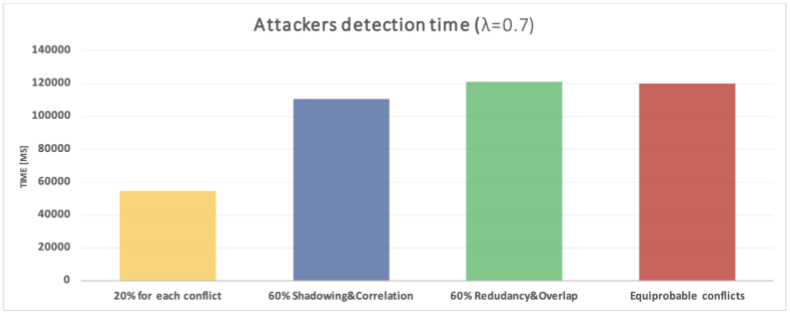


Figura 11: detection time con $\lambda = 0.3$

6. Conclusioni

Il progetto ha esplorato in profondità l'integrazione e l'ottimizzazione delle regole di flusso in un ambiente SDN, con particolare attenzione alla *detection* dei conflitti tramite l'implementazione di un'applicazione sviluppata su ONOS.
L'applicazione "**Conflict Detector"** ha dimostrato di essere capace di identificare in modo efficiente gli utenti che generano conflitti di regole che, se non trattati, potrebbero portare a vulnerabilità significative e a una riduzione delle prestazioni della rete.
Scegliendo un valore di soglia di trustiness accurato, il sistema è in grado di rilevare un attaccante presente nella rete con un alto livello di precisione, con un alto valore di F1-Score.
I risultati hanno evidenziato che i conflitti di maggiore gravità vengono risolti più rapidamente rispetto a quelli di minore entità, in modo tale che un attaccante malevolo possa arrecare meno danni al sistema.
I risultati ottenuti indicano che l'approccio proposto può essere esteso e applicato a scenari più complessi, come reti multi-dominio o ambienti cloud.