UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA
DIMES

Corso di Laurea in

Ingegneria Informatica

Progetto Didattico di

Network Security:

Analisi di sicurezza su reti SDN: attacchi, contromisure e valutazione prestazioni

Professori

Floriano De Rango

Mattia Giovanni Spina

Studenti

Eros De Rose 224482

Antonio Marino 224598

Vincenzo Calanna 224533

INDICE

1) Introduzione

I modelli di rete SDN (Software-Defined Networking) sono stati introdotti per offrire una maggiore flessibilità e agilità alle reti informatiche. Prima dell'introduzione di SDN, le reti erano gestite da dispositivi di rete specializzati (switch, router, firewall) che erano configurati tramite protocolli di gestione della rete come SNMP (Simple Network Management Protocol).

Tuttavia, questa architettura tradizionale di rete presenta alcune limitazioni. Ad esempio, i dispositivi di rete specializzati hanno un comportamento predefinito e non possono facilmente adattarsi a nuove esigenze o cambiamenti nell'ambiente di rete. Inoltre, la gestione centralizzata della rete può diventare molto complessa e può richiedere molte risorse.

Con SDN, la gestione della rete viene decentralizzata e spostata su un controller di rete centrale. Questo controller può programmare in modo dinamico il comportamento dei dispositivi di rete, offrendo flessibilità, scalabilità e agilità, semplificando la gestione della rete e migliorando l'efficienza operativa, adattandoli alle esigenze specifiche della rete in ogni momento. Inoltre, la gestione centralizzata semplifica la configurazione, la gestione e il monitoraggio della rete. Si precisa, inoltre, che tutte le topologie di rete prese in considerazione sono sempre dotate di un singolo controller.

L’idea progettuale consiste nello studio di alcuni attacchi contro una rete SDN, nello specifico verranno studiati l’attacco ARP e la violazione del protocollo LLDP.

2) SDN

SDN (Software Defined Networking) è una architettura di rete in cui la logica di controllo della rete viene separata dalla logica di inoltro dei pacchetti. In una rete SDN, la logica di controllo è gestita da un software di controllo centrale, mentre gli switch e i router eseguono solo il forwarding dei pacchetti. Questo consente una maggiore flessibilità nella configurazione della rete, poiché le modifiche alla configurazione possono essere effettuate in modo centralizzato attraverso il software di controllo. Inoltre, l'architettura SDN consente una maggiore programmabilità, poiché i flussi di traffico possono essere gestiti in modo dinamico attraverso l'utilizzo di API.

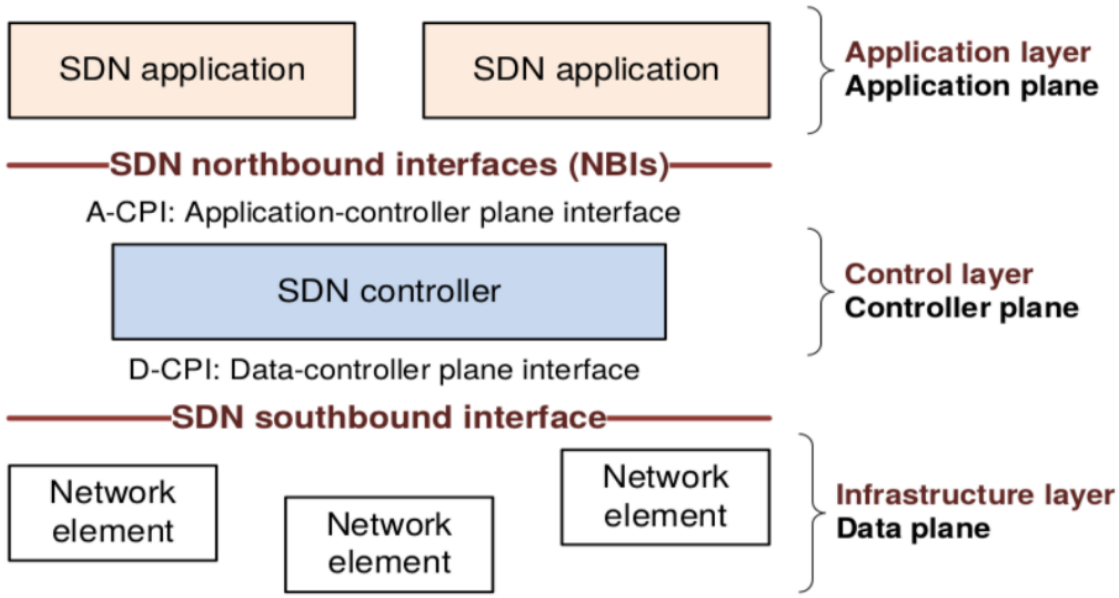

Figura 1: architettura SDN

L'Application Layer dell'SDN si occupa della gestione delle applicazioni e dei servizi di rete attraverso l'utilizzo di un software di controllo centrale. In questo livello vengono definite le regole di inoltro dei pacchetti e le politiche di sicurezza, che poi vengono implementate dai dispositivi di inoltro (switch e router) nei livelli inferiori dell'architettura SDN.
In generale, l'Application Layer dell'SDN fornisce un'interfaccia di programmazione (API) per gli sviluppatori di applicazioni, che consente loro di controllare e configurare dinamicamente la rete, quindi gestione della banda, gestione della sicurezza e gestione della virtualizzazione della rete.

Il Control Layer dell'SDN è il livello principale dell'architettura SDN che si occupa della logica di controllo della rete. In questo livello, un software di controllo centrale (chiamato anche controller SDN) gestisce la configurazione della rete e determina come i pacchetti devono essere inoltrati attraverso la rete.
Il controller SDN utilizza una vista globale della rete per prendere decisioni informate su come inoltrare i pacchetti e per gestire la configurazione dei dispositivi di inoltro (switch e router) nei livelli inferiori dell'architettura SDN. Inoltre, il controller SDN può utilizzare algoritmi di ottimizzazione per determinare il percorso ottimale per i pacchetti attraverso la rete.
Il Control Layer dell'SDN utilizza una serie di protocolli per comunicare con i dispositivi di inoltro, come OpenFlow, che è un protocollo di comunicazione standard per l'SDN. Inoltre, l'interfaccia di programmazione (API) dell'Application Layer dell'SDN è utilizzata per comunicare con le applicazioni e i servizi di rete.
L'Infrastructure Layer dell'SDN rappresenta la parte fisica della rete, che include gli switch e i router che eseguono il forwarding dei pacchetti. In questo livello, i dispositivi di inoltro sono controllati dal Control Layer dell'SDN attraverso il protocollo OpenFlow o altri protocolli di comunicazione.
I dispositivi di inoltro nell'Infrastructure Layer dell'SDN sono progettati per essere "programmabili" e per supportare la configurazione dinamica tramite il software di controllo centrale nel Control Layer. In questo modo, i dispositivi di inoltro possono essere configurati e gestiti in modo centralizzato, anziché essere configurati e gestiti singolarmente.
Inoltre, gli switch e i router nell'Infrastructure Layer dell'SDN possono includere funzionalità avanzate, come la gestione della qualità del servizio (QoS), la gestione della banda, la gestione della sicurezza e la gestione della virtualizzazione della rete.

2.1) OpenFlow

OpenFlow è un protocollo di comunicazione standard per l'architettura Software-Defined Networking (SDN). Esso consente a un controller SDN di comunicare con i dispositivi di inoltro (switch e router) nell'Infrastructure Layer dell'SDN per configurare e gestire dinamicamente la rete.
OpenFlow specifica un formato per il messaggio di controllo e un formato per la tabella di inoltro. Il controller SDN utilizza questi formati per comunicare con i dispositivi di inoltro e per configurare le regole di inoltro dei pacchetti. Inoltre, OpenFlow specifica

una serie di azioni che i dispositivi di inoltro possono eseguire sui pacchetti, come l'inoltro, la modifica delle intestazioni dei pacchetti e l'invio di pacchetti a un'applicazione specifica.

I messaggi OpenFlow scambiati tra il controller SDN e i dispositivi di inoltro possono essere di diversi tipi, a seconda della funzione o dell'operazione che devono svolgere:

- Messaggi controller-to-switch: sono inviati dal controller e non richiedono una risposta da parte dello switch. Questi messaggi vengono utilizzati principalmente per modificare la configurazione dello switch, apportare modifiche alla tabella di inoltro e inviare e inoltrare pacchetti ricevuti.
- Messaggi asincroni: sono inviati dagli switch al controller senza che il controller abbia fatto una richiesta. Questi messaggi includono i pacchetti "packet-in" per i quali non esiste una corrispondenza nella tabella di inoltro, nonché i pacchetti che informano il controller di modifiche alle porte dello switch o di rimozioni di voci dalle tabelle.
- Messaggi simmetrici: sono bilaterali e vengono utilizzati per istanziate la connessione tra controller e switch e per verificare che la connessione sia ancora attiva.

Quando uno switch OpenFlow riceve un pacchetto, esso viene elaborato dalla pipeline di inoltro. La pipeline include una serie di confronti tra i campi del pacchetto ricevuto e quelli presenti nella tabella di inoltro. Poiché può essere presente più di una tabella, queste vengono ordinate in base alla priorità del confronto con i campi del pacchetto. Se esiste una corrispondenza tra i campi del pacchetto e quelli presenti in una delle tabelle, la ricerca termina e vengono eseguite le azioni corrispondenti. Se invece non viene trovata alcuna corrispondenza, il pacchetto viene associato ad una voce di "table-miss" e si decide se scartarlo o inoltrarlo al controller per ulteriori elaborazioni.

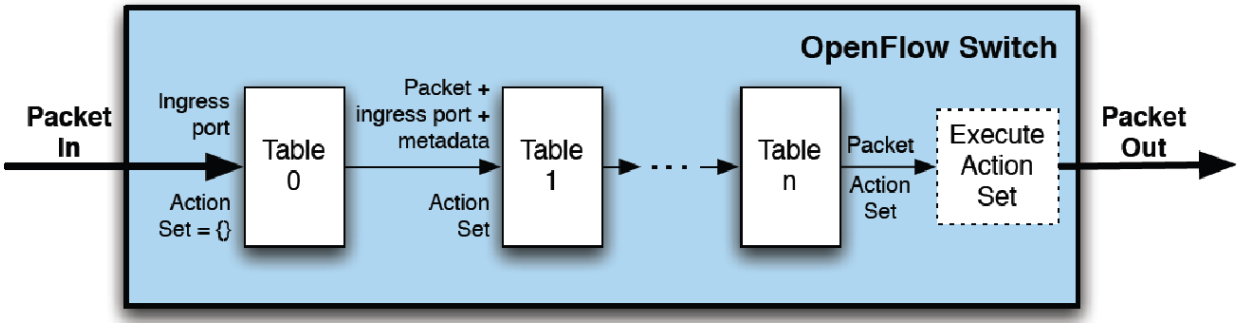

Figura 2: OpenFlow Switch

Ogni voce presente in una flow table è costituita da un insieme di campi:

- Match fields: sono i valori utilizzati per verificare la corrispondenza del pacchetto in arrivo. Possono comprendere informazioni come la porta di ingresso, l'header del pacchetto ed eventuali metadati di altre tabelle (se presente un meccanismo di pipeline).
- Counters: contatori che vengono incrementati solo nel caso in cui si verifichi una corrispondenza del pacchetto.
- Priority: valore numerico assegnato in caso di due o più corrispondenze del pacchetto.
- Instructions: azioni da eseguire nel caso in cui si verifichi una corrispondenza del pacchetto.
- Time out: tempo di permanenza della voce all'interno della tabella prima che venga rimossa.
- Cookie: valori utilizzati dal controller per ottenere informazioni statistiche.
- Flags: valori utilizzati per differenziare la gestione di una voce della tabella.

2.2) Mininet

Mininet è un ambiente di simulazione di rete software-defined networking (SDN) open-source. Permette di creare e testare topologie di reti complesse in un ambiente di simulazione virtuale su un unico computer. Utilizza il software di virtualizzazione OpenVSwitch per creare switch virtuali e il software di emulazione di nodi di rete Linux per creare host virtuali. Mininet consente agli sviluppatori di testare i propri controller SDN e le applicazioni su una vasta gamma di topologie di rete, senza dover utilizzare attrezzature di rete fisiche. Inoltre, fornisce un'interfaccia di riga di comando intuitiva per la configurazione e il controllo della topologia di rete simulata, nonché un'interfaccia Python per la programmazione automatizzata dei test.

2.3) POX

POX è una piattaforma software open-source utilizzata per la creazione e l'esecuzione di controller per le reti Software Defined (SDN). POX offre una serie di moduli precostruiti che possono essere utilizzati per implementare diverse funzionalità di controllo, come ad esempio l'apprendimento degli indirizzi MAC, il routing dinamico, la gestione del flusso, ecc. Inoltre, POX è progettato per essere estendibile, permettendo agli sviluppatori di

creare i propri moduli personalizzati per soddisfare le esigenze specifiche delle loro reti SDN.

2.3.1) Modulo forwarding.l3_learning

Il modulo l3_learning del controller POX è un modulo di apprendimento di livello 3. Il suo scopo principale è quello di imparare le route di livello 3 della rete e di installare automaticamente le corrispondenti flow table negli switch per inoltrare i pacchetti. Il modulo utilizza il protocollo ARP per imparare gli indirizzi MAC delle interfacce di rete e per scoprire la topologia della rete. Quando un pacchetto ARP è ricevuto da uno switch, il modulo l3_learning lo utilizza per imparare l'indirizzo MAC dell'host e l'interfaccia di rete su cui si trova. Inoltre, il modulo utilizza l'informazione per installare automaticamente una voce nella flow table dello switch per inoltrare i pacchetti destinati a quell'host. In questo modo, il modulo l3_learning consente di automatizzare la configurazione dei percorsi di livello 3 nella rete SDN. Nel primo attacco che andremo a vedere si cercherà di violare questo modulo tramite un attacco di tipo ARP poisoning e successivamente si vedrà una mitigazione a questo attacco.

2.3.2) Modulo openflow.discovery

Il modulo discovery del controller POX è un modulo Python che consente ai dispositivi di rete di scoprire la topologia di rete in cui si trovano. Il modulo discovery utilizza il protocollo OpenFlow per interrogare gli switch della rete e raccogliere informazioni sulla topologia della rete, come ad esempio le connessioni tra gli switch e gli indirizzi MAC degli host presenti nella rete. Il modulo discovery fornisce un'implementazione di base del protocollo Topology Discovery di OpenFlow, che può essere utilizzato come base per lo sviluppo di applicazioni di rete più avanzate. Ad esempio, è possibile utilizzare il modulo discovery per sviluppare applicazioni di rete che ottimizzano il routing dei pacchetti, minimizzano il congestionamento della rete o rilevano le interruzioni di rete. In sintesi, il modulo discovery del controller POX è uno strumento potente per comprendere la topologia della rete e sviluppare applicazioni di rete intelligenti e sofisticate. Nel secondo attacco che andremo a vedere si cercherà di violare questo modulo tramite un attacco di tipo LLDP injection e successivamente si vedranno alcune mitigazioni a questo attacco.

3) Address Resolution Protocol

Per stabilire una connessione tra due host, A e B, in una rete classica, l'host A deve prima ottenere l'indirizzo MAC associato all'indirizzo IP dell'host B tramite il protocollo ARP. Questo avviene mediante una richiesta broadcast (ARP request) inviata da A a tutti i dispositivi della rete, chiedendo l'indirizzo MAC dell'IP dell'host B. Tutti gli host riceveranno la richiesta, ma solo B fornirà una risposta tramite ARP reply.

3.1) Attacco ARP poisoning

L'ARP poisoning è un attacco in cui un attaccante che controlla un host della rete invia pacchetti di ARP reply falsi per manipolare la tabella ARP di un altro host. In questo modo, tutti i pacchetti destinati all'host B verranno inavvertitamente inviati all'host controllato dall'attaccante. L'attacco può avvenire anche senza una richiesta ARP precedente poiché il protocollo ARP non controlla se un pacchetto di risposta sia correlato a una specifica richiesta.

Per il progetto, è stata presa in considerazione la tabella ARP costruita all'interno del modulo del controller l3_learning.

L'attacco utilizzato è stato implementato sfruttando la libreria Scapy, inondando un host della rete forgiando pacchetti di tipo *ARP reply* con i seguenti campi:

- IP sorgente corrispondente a quello dell'host controllato dall'attaccante,
- MAC sorgente fake, ovvero non appartenente ad alcun host all'interno della rete,
- IP destinazione corrispondente ad un host della rete,
- MAC destinazione corrispondente all'host con l'IP destinazione precedente.

In maniera equivalente, come MAC sorgente si potrebbe inserire anche quello dell'host attaccante.

3.2) Esempio di attacco

Supponiamo che, all'interno di una rete costruita come nella figura seguente, l'host denominato h2 voglia effettuare il poisoning della tabella ARP degli host denominati h1 e h3.

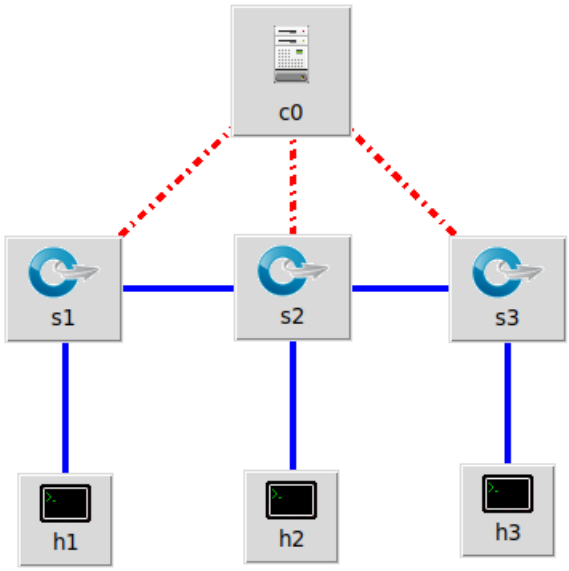

Figura 3: rete SDN su mininet

Nella figura seguente si riporta la tabella ARP dell'host h1 in due momenti differenti: la prima che contiene le corrispondenze IP-MAC corrette mentre la seconda a seguito di un attacco. Si nota, quindi, come nel secondo caso il MAC associato all'host h2 corrispondente all'IP 10.0.0.2 non sia quello reale, ma un indirizzo fake scelto dall'attaccante.

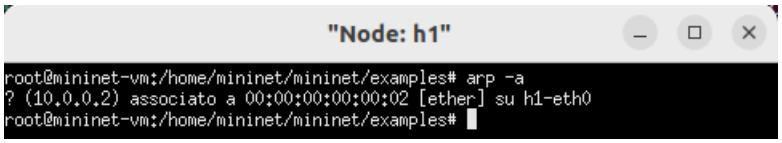

Figura 4: tabella ARP nodo 1 senza attacco

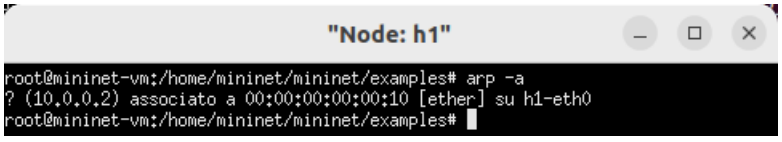

Figura 5: tabella ARP nodo 1 dopo l'attacco

3.3) Mitigazione

Il metodo checkArp è stato implementato per controllare i pacchetti ARP reply ricevuti dal controller. Viene chiamato all'interno del metodo _handle_openflow_PacketIn ogni volta che il controller riceve un pacchetto di questo tipo. Il suo scopo è verificare se i dati contenuti nel pacchetto indicano un attacco di tipo ARP poisoning. L'algoritmo utilizzato per la rilevazione dell'attacco è stato creato utilizzando una tabella ARP statica all'interno della classe Python. Questa tabella contiene le corrispondenze tra gli indirizzi IP e MAC degli host presenti nella rete e non viene modificata una volta avviato il controller. Lo pseudocodice dell'algoritmo è mostrato nella figura 6.

1. If source MAC of Ethernet not like Source MAC of ARP
2. Spoofed (Drop)
3. Else
4. If source MAC-IP addresses mapping in ARP header not found in Main table.
5. Spoofed (Drop)
6. Else
7. If Destination IP of ARP not found in Main Table
8. Spoofed (Drop)
9. Else
10. If Destination MAC is Broadcast
11. If ARP Reply and Destination MAC is Broadcast and Source IP not like Destination IP
12. Spoofed (Drop)
13. Else
14. Broadcast
15. Else
16. Forward to designated host

Figura 6: pseudocodice algoritmo verifica pacchetto ARP malevolo

Nel caso in cui il metodo usato per la detection ritorni esito positivo, il metodo *_handle_openflow_PacketIn* viene immediatamente terminato, in modo da non modificare le tabelle ARP contenente i valori corretti definiti all'avvio.

Di seguito si mostra come, a seguito di un attacco il controller sia stato in grado di effettuare la detection appena descritta e di ignorare i pacchetti malevoli, senza inoltrare verso gli switch un pacchetto di tipo *Packet-out*.

```
WARNING:forwarding.l3_mitigationARPRegolaFlusso:00:00:00:00:00:02->00:00:00:00:0
0:01 ignorato
```

Figura 7: rilevamento pacchetto ARP malevolo da parte del controller

Nella figura seguente è riportata la regola di flusso installata sullo switch s2 a seguito dell'attacco ARP poisoning effettuato dall'host h2. I pacchetti arp provenienti dall'host h2 vengono droppati per un tempo di 10 secondi.

```
mininet> sh ovs-ofctl dump-flows s2
 cookie=0x0, duration=8.888s, table=0, n_packets=4, n_bytes=168, idle_timeout=10
, hard_timeout=10, arp,dl_src=00:00:00:00:00:02,arp_spa=10.0.0.2 actions=drop
```

Figura 8: installazione regola di flusso su switch

In particolare, nell'esempio riportato, l'attacco ricade nel caso in cui l'indirizzo MAC sorgente del livello Ethernet sia differente rispetto a quello a livello ARP. Nello script di attacco, infatti, si è intervenuti nei campi contenuti all'interno del frame ARP, riportato in figura.

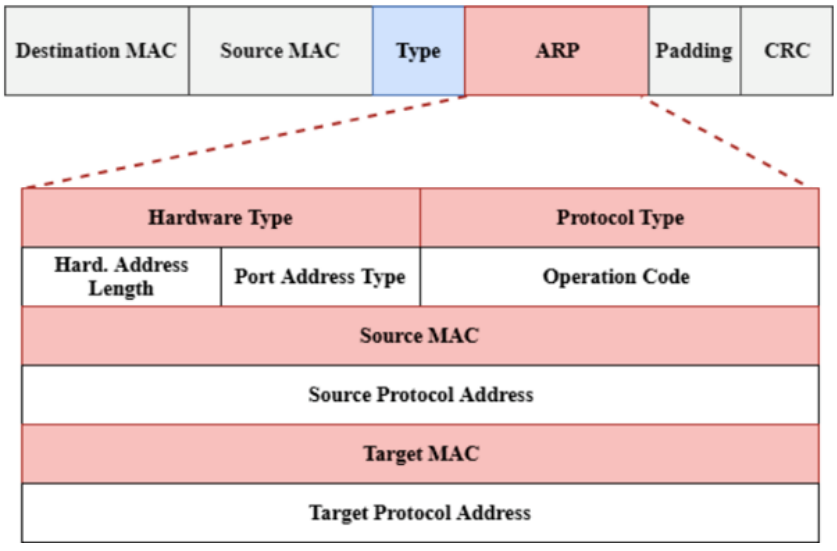

Figura 9: frame ARP

3.4) Valutazione prestazioni

Attraverso l'utilizzo di Tyshark è stato possibile determinare il numero di pacchetti ARP totali scambiati all'interno della rete nelle diverse situazioni indicate nel grafico. In particolare, si può osservare nella situazione sotto attacco una significativa riduzione di pacchetti ARP rilevati quando la mitigazione è attiva, poiché i pacchetti vengono bloccati direttamente dallo switch dopo aver installato la regola di flusso. Mentre nella situazione senza mitigazione i pacchetti di tipo ARP malevoli vengono processati senza fare distinzione tra pacchetti leciti e illeciti.

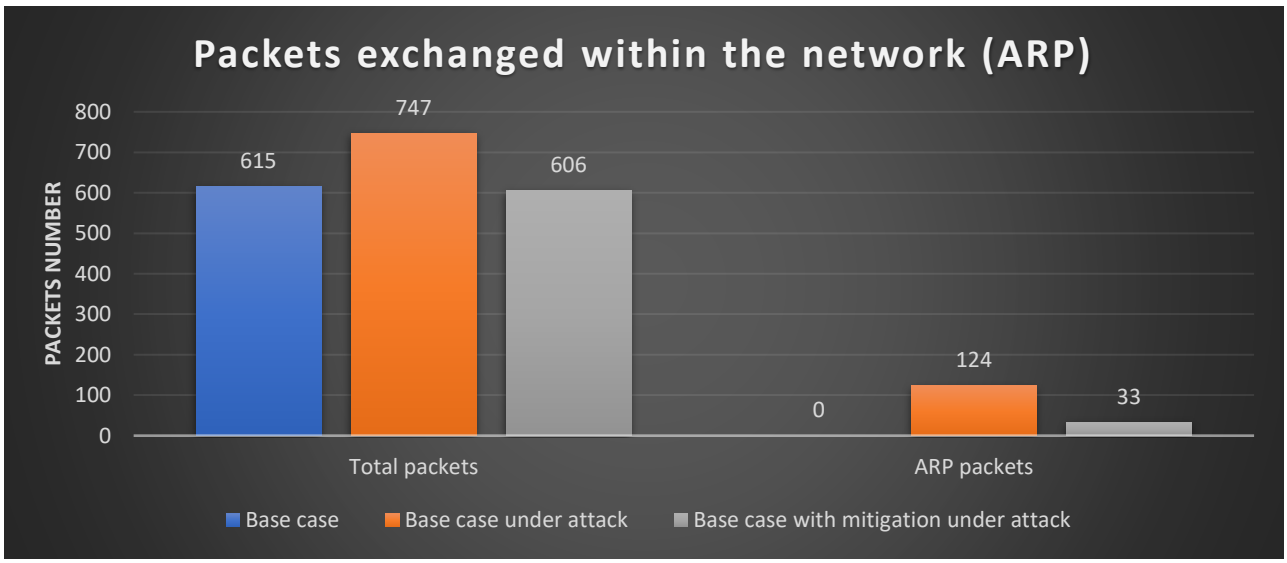

Figura 10: pacchetti scambiati all'interno della rete

Attraverso lo strumento Wireshark, è stato possibile rilevare la larghezza di banda utilizzata nelle varie situazioni, considerando lo stesso tempo per ciascun caso evidenziato. In particolare, nel caso base e nella situazione sotto attacco con la mitigazione attiva, la larghezza di banda utilizzata è simile, poiché vengono processati solo i pacchetti leciti. Mentre nel caso sotto attacco è presente un incremento di pacchetti dovuto all'attacco stesso e tutti i pacchetti (leciti o illeciti) vengono processati.

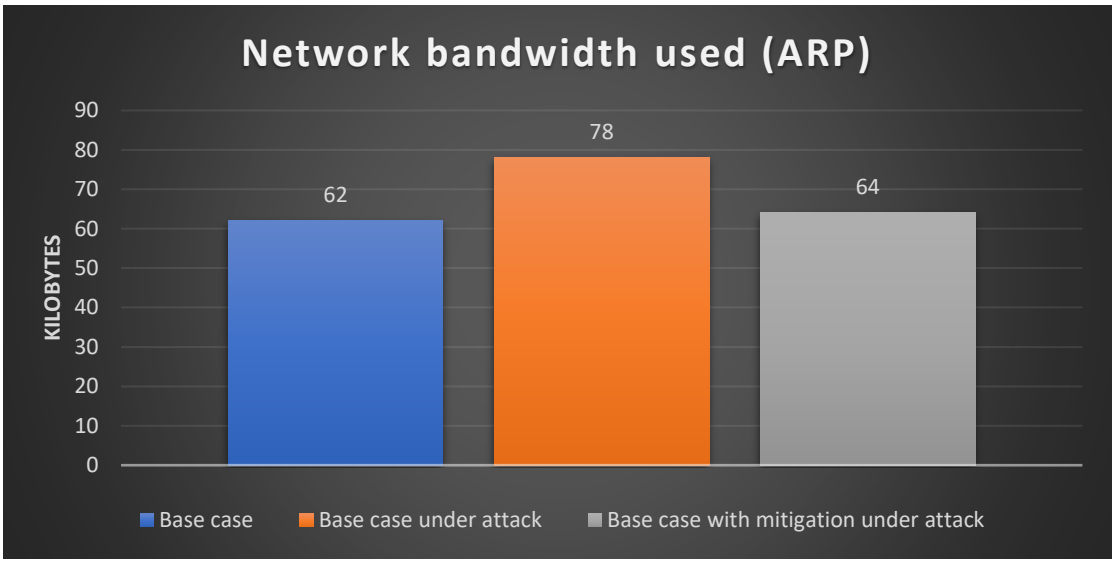

Figura 11: banda utilizzata

Attraverso l'utilizzo di uno script realizzato appositamente, è stato possibile salvare i dati relativi all'utilizzo di CPU e RAM da parte del controller, nel tempo. Dal grafico possiamo osservare che vi è un utilizzo leggermente maggiore della CPU nel caso di attacco senza la mitigazione attiva, rispetto al caso base e al caso di attacco con mitigazione. Questo perché il carico di lavoro è maggiore in quanto tutti i pacchetti passano dal controller e non vengono immediatamente bloccati dallo switch.

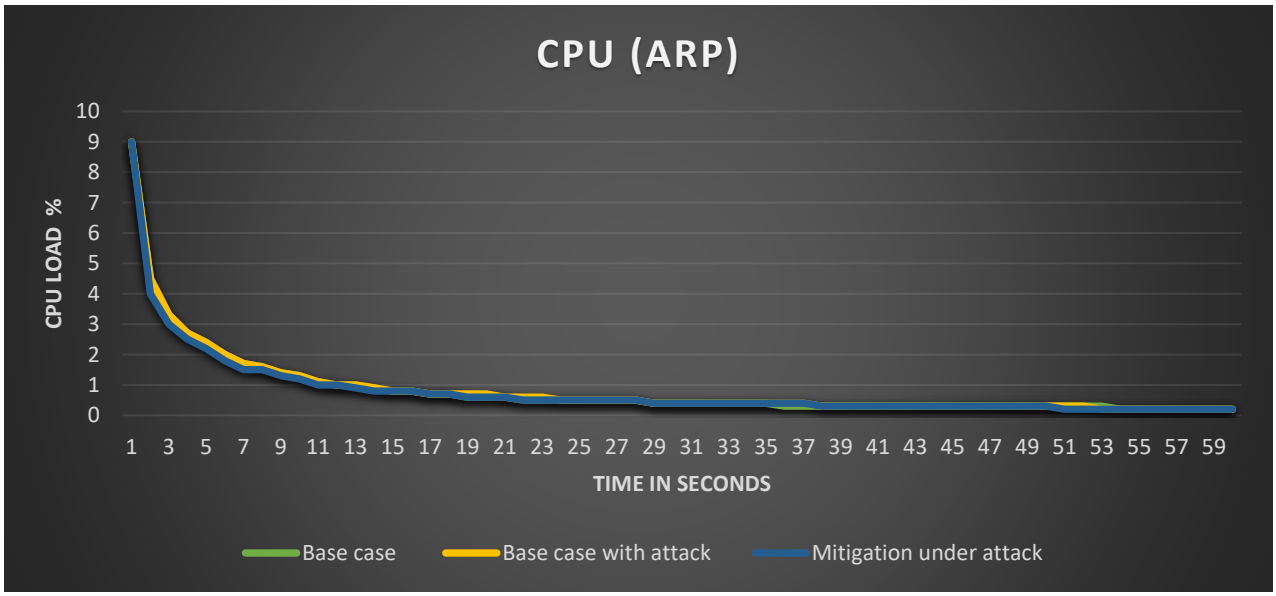

Figura 12: utilizzo CPU

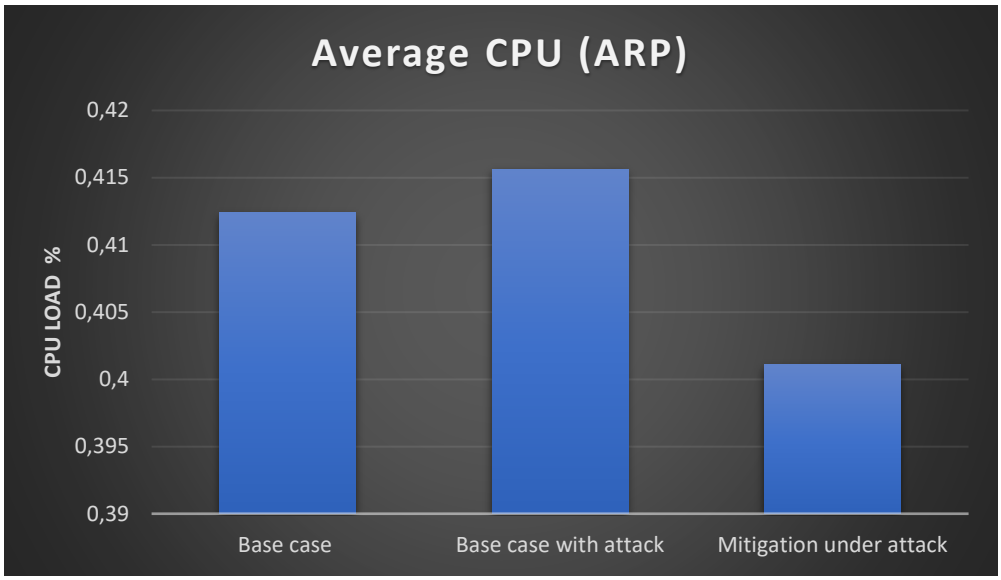

Figura 13: utilizzo CPU medio

Mentre per quanto riguarda la RAM utilizzata, il valore nel tempo è identico per tutti e tre i casi poiché l'attacco e la mitigazione non risultano dal punto di vista computazionale più pesanti.

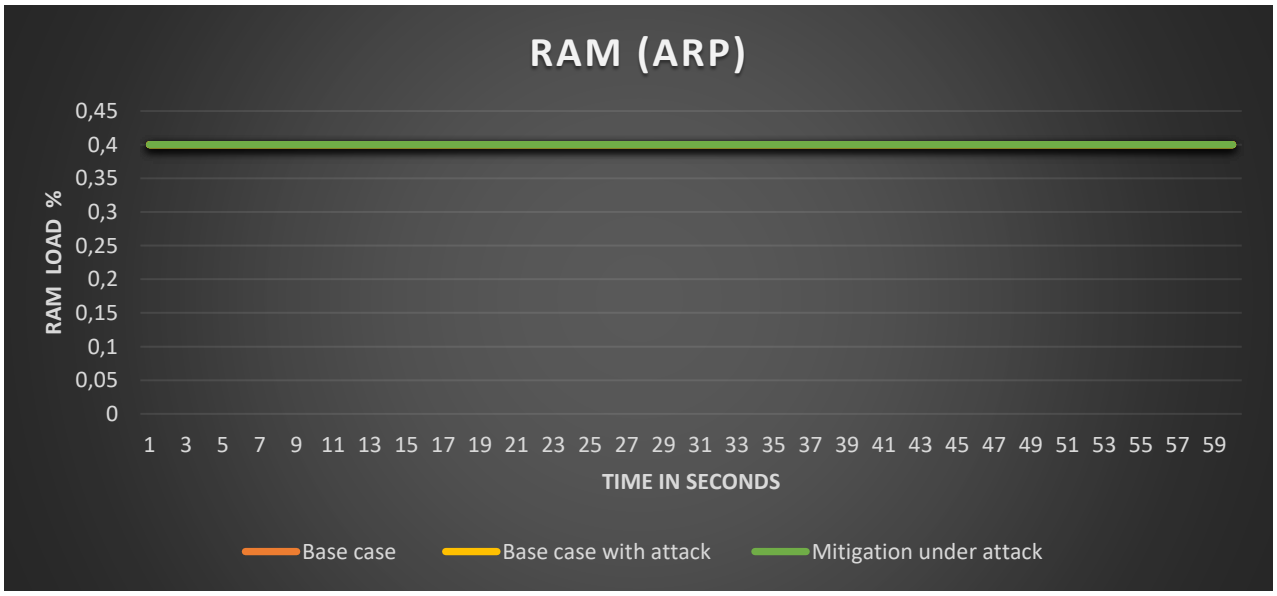

Figura 14: utilizzo memoria

Attraverso l'utilizzo di codice aggiuntivo nel controller è stato possibile calcolare i tempi di latenza dei pacchetti accettati o rifiutati. In particolare, come è evidenziato dal grafico, in caso di rete senza mitigazione ogni pacchetto ARP passa attraverso tutti i controlli di verifica e viene accettato in qualsiasi caso. Mentre quando è attiva la mitigazione, il pacchetto malevolo viene identificato dal metodo checkARP, senza passare

attraverso tutti i controlli successivi di verifica, e droppato. Inoltro nel grafico è possibile osservare la latenza media dell'installazione della regola di flusso sullo switch.

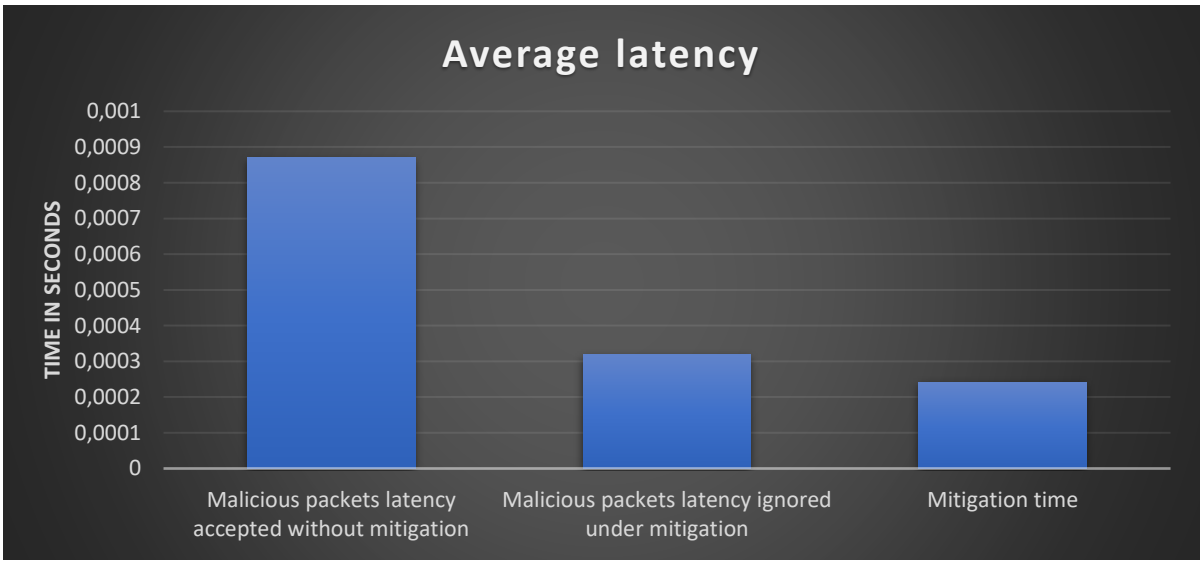

Figura 15: latenza media

Inoltre i dati riguardanti i pacchetti scambiati nel tempo sono stati raccolti attraverso l'utilizzo di Wireshark. In una prima fase c'è quella di discovery della rete, avviata dal controller, nel quale vengono identificati switch e host della rete attraverso invio di pacchetti. Dopodichè la rete, nella situazione normale scambia regolarmente pacchetti.

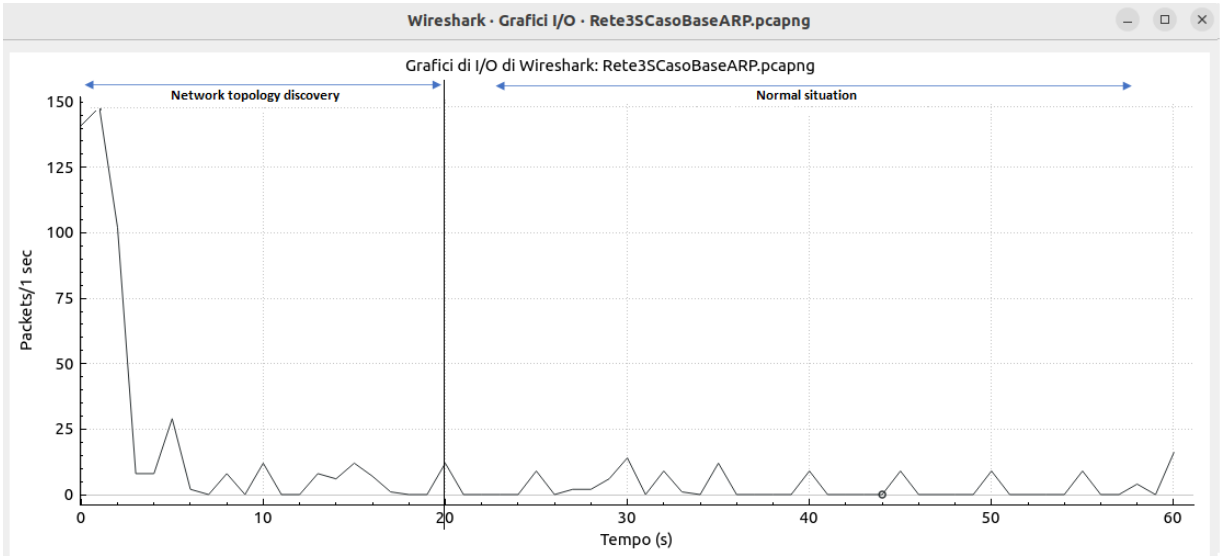

Figura 16: scambio pacchetti nel tempo caso base

Invece, nel caso di attacco senza mitigazione attiva, dopo la fase di discovery della rete viene effettuato l'attacco da un host che attraverso dei pacchetti in broadcast identifica

il MAC della vittima e successivamente vi è il tentativo di avvelenamento (poisoning) della tabella ARP dell'host vittima. Dopodichè il "poisoning" viene riprovato ogni cinque secondi in modo tale che la tabella ARP risulti sempre avvelenata e non si sistemi con l'arrivo di pacchetti validi.

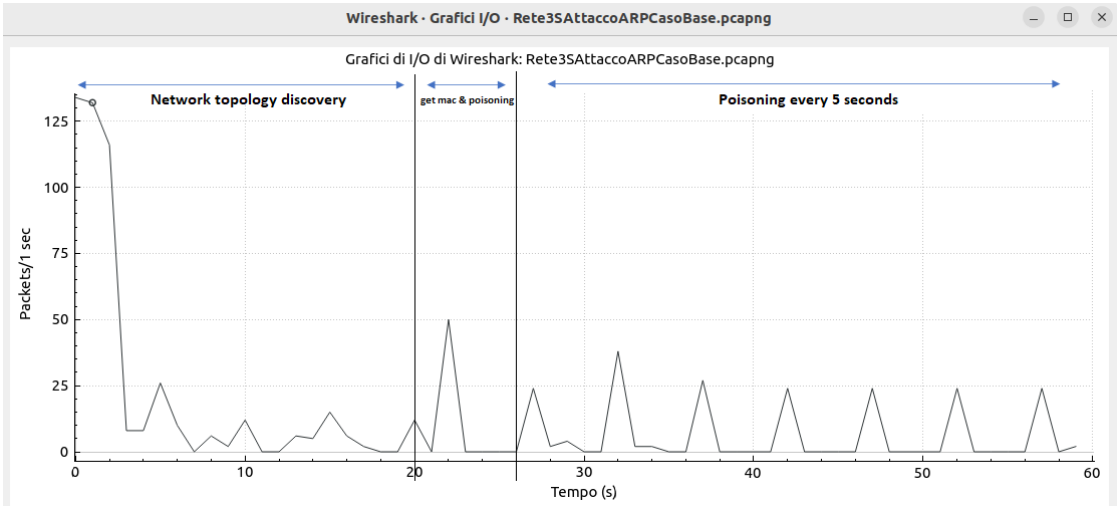

Figura 17: scambio pacchetti nel tempo sotto attacco

Nel caso di attacco con mitigazione attiva, le prime due fasi sono identiche a quelle evidenziate in precedenza, mentre nella terza fase viene identificato l'attacco da parte dell'host e viene installata la regola di flusso sullo switch collegato all'host attaccante. Quest'ultimo viene bloccato per un periodo prefissato di tempo e la rete si stabilizza nel suo caso normale.

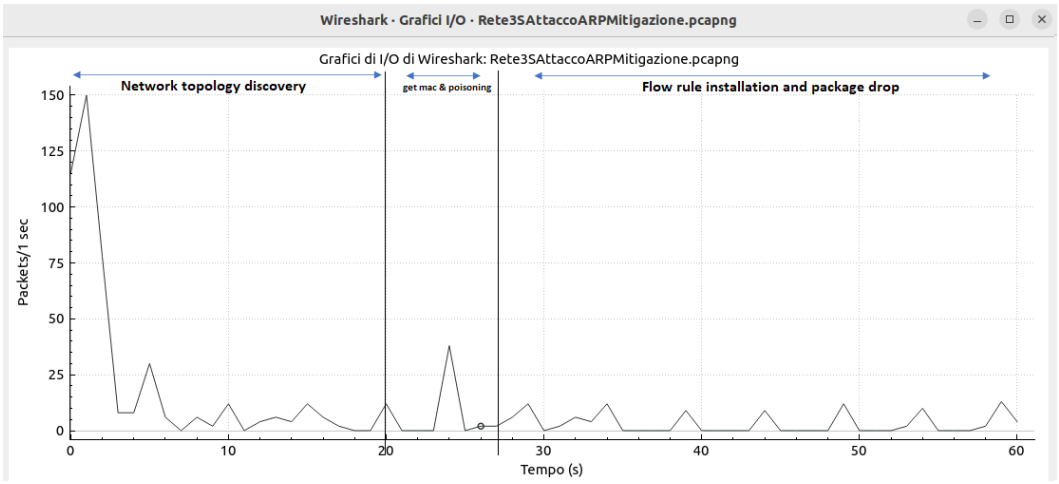

Figura 18: scambio pacchetti nel tempo sotto attacco con mitigazione

4) Link Layer Discovery Protocol

LLDP (Link Layer Discovery Protocol) è un protocollo di rete utilizzato per scoprire automaticamente le informazioni sui dispositivi connessi in una rete LAN (Local Area Network). LLDP consente ai dispositivi di scoprire informazioni sui propri vicini, come indirizzi MAC e indirizzi IP, tipi di dispositivi e di connessione, nomi di dispositivi e altre informazioni. Il protocollo è stato progettato per essere indipendente dalla piattaforma e dal sistema operativo, in modo che i dispositivi di diversi produttori possano comunicare tra loro. LLDP viene utilizzato principalmente per la gestione della rete, come la risoluzione dei problemi e la configurazione automatica dei dispositivi. Un frame LLDP, come mostrato nella figura sottostante, è un pacchetto di dati utilizzato per trasmettere informazioni LLDP su una LAN. È composto da una serie di campi, tra cui:

- L'indirizzo MAC di destinazione, utilizzato per identificare il dispositivo di rete a cui il pacchetto deve essere inviato,
- L'indirizzo MAC di origine, utilizzato per identificare il dispositivo di rete che invia il pacchetto
- Il tipo di pacchetto, utilizzato per identificare che si tratta di un pacchetto LLDP
- La sezione TLV (Type-Length-Value), che contiene le informazioni specifiche sui dispositivi di rete e sulla loro configurazione.

Il pacchetto LLDP viene inviato in modo periodico da ogni dispositivo di rete sulla LAN, in modo che gli altri dispositivi possano raccogliere informazioni sui loro vicini di rete.

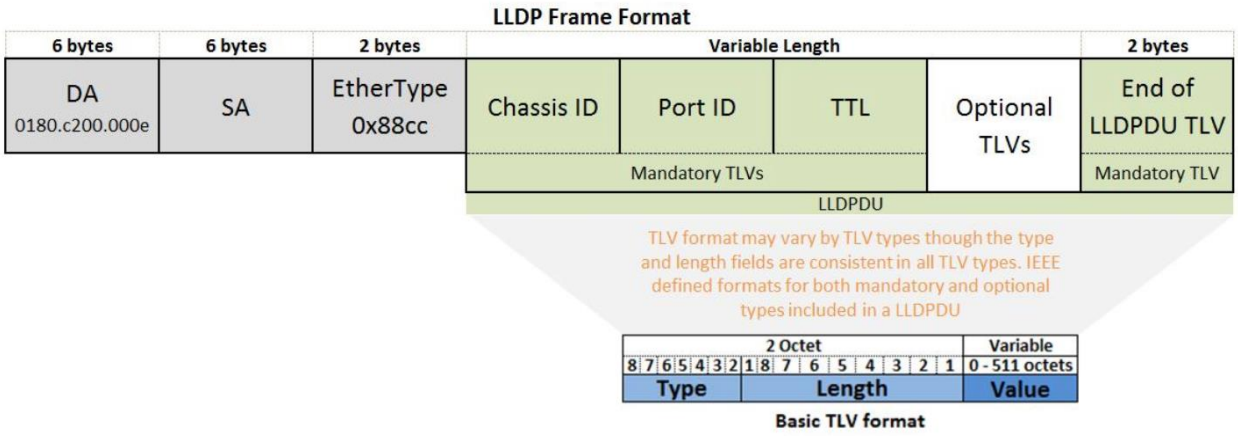

Figura 19: frame LLDP

I campi della sezione TLV (Type-Length-Value) in un frame LLDP sono costituiti da tre parti:

1) Type: indica il tipo di informazione contenuta nel TLV, come ad esempio l'indirizzo MAC, l'indirizzo IP, il nome del dispositivo, ecc.
2) Length: indica la lunghezza in byte del campo Value.
3) Value: contiene il valore effettivo dell'informazione descritta dal campo Type.

Nello scenario della figura sottostante, il controller invia dei pacchetti di tipo Packet_Out allo switch s1 sulla porta con ID 1. Lo switch s1 riceverà messaggi Packet_Out, decomprimerà il pacchetto LLDP dal messaggio Packet_Out e inoltrerà solo il pacchetto LLDP corrispondente (basato sull'indirizzo MAC della porta in LLDPDU). Quando lo switch s2 riceve un pacchetto LLDP, lo analizzerà, scriverà il suo Chassis-ID switch e aggiungerà port ID (ovvero, la porta attraverso cui lo switch ha ricevuto il pacchetto LLDP). Quindi, lo switch s2 incapsulerà il pacchetto LLDP in un messaggio Packet_In e lo invierà al controller. Il controller a sua volta analizzerà il messaggio Packet_In e scoprirà i nuovi collegamenti rappresentati dalla mappatura tra s1 e s2. Tuttavia, lo stesso metodo viene ripetuto per scoprire i restanti collegamenti nella rete.

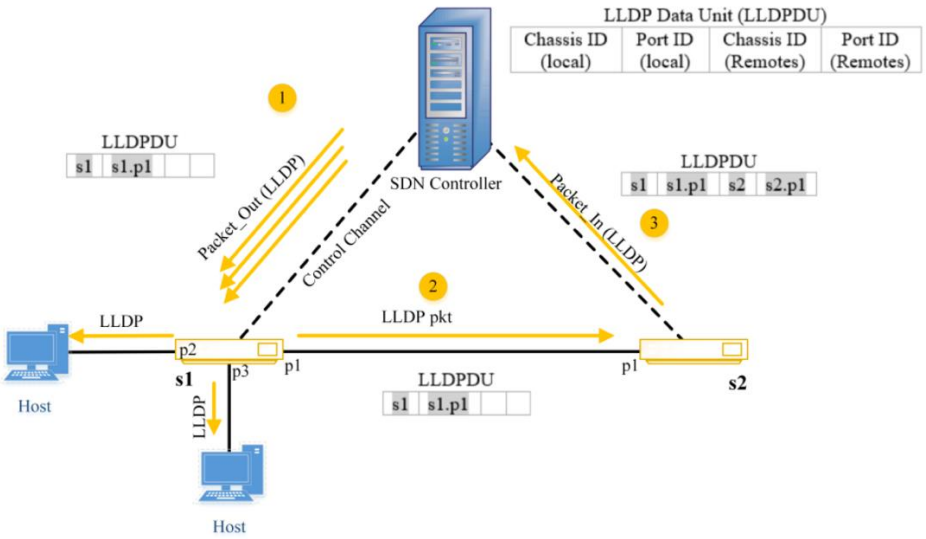

Figura 20: esempio scambio pacchetto LLDP

4.1) Attacco LLDP injection

In questo scenario d'attacco l'obiettivo è quello di violare il protocollo LLDP, generando false informazioni sulla topologia di rete che verranno memorizzate ed elaborate dal controller di rete.

L'attacco consiste nella generazione di un link fasullo in seguito ad uno sniffing di un pacchetto LLDP. Un host A collegato a uno switch S1 si mette in ascolto su un'interfaccia per ottenere un pacchetto LLDP. Una volta ottenuto il pacchetto LLDP sostituisce il dpid (corrispondente al MAC dello switch S1) di destinazione del pacchetto con il dpid di uno switch S2 a cui lo switch S1 non è collegato e invia questo nuovo pacchetto fabbricato sulla rete in modo tale che il controller veda un link fasullo tra i due switch che in realtà non sono collegati realmente.

4.2) Esempio d'attacco

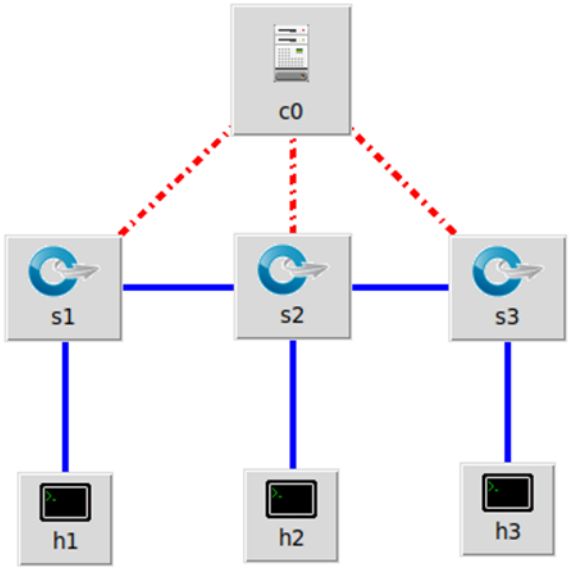

Figura 21: rete SDN costruita su mininet

Nello scenario d'attacco l'attaccante, rappresentato dall'host h1, si mette in ascolto su un'interfaccia e cattura un pacchetto LLDP con il dpid dello swtich s1. Il pacchetto LLDP viene modificato sostituendo il dpid dello switch s1 con il dpid dello switch s3. Il pacchetto viene inviato sulla rete ingannando così il controller che vedrà un link, in realtà inesistente, tra lo switch s1 e lo switch s3.

Nella figura seguente si riportano i link visti dal controller in due momenti differenti: la prima figura contiene solo i link reali tra switch s1-s2 e s2-s3, la seconda contiene i link reali più il link fasullo tra s1-s3 in seguito all'attacco.

```
mininet@mininet-vm:~/pox$ python3 pox.py --verbose openflow.discovery
POX 0.8.0 (halosaur) / Copyright 2011-2022 James McCauley, et al.
DEBUG:core:POX 0.8.0 (halosaur) going up...
DEBUG:core:Running on CPython (3.10.6/Nov 14 2022 16:10:14)
DEBUG:core:Platform is Linux-5.15.0-58-generic-x86_64-with-glibc2.35
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:core:POX 0.8.0 (halosaur) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:openflow.discovery:Installing flow for 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
DEBUG:openflow.discovery:Installing flow for 00-00-00-00-00-03
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
DEBUG:openflow.discovery:Installing flow for 00-00-00-00-00-02
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.2 -> 00-00-00-00-00-01.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.3 -> 00-00-00-00-00-03.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.2 -> 00-00-00-00-00-02.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.2 -> 00-00-00-00-00-02.2
```

Figura 22: link tra switch visti dal controller in situazione senza attacco

```
mininet@mininet-vm:~/pox$ python3 pox.py --verbose openflow.discovery
POX 0.8.0 (halosaur) / Copyright 2011-2022 James McCauley, et al.
DEBUG:core:POX 0.8.0 (halosaur) going up...
DEBUG:core:Running on CPython (3.10.6/Nov 14 2022 16:10:14)
DEBUG:core:Platform is Linux-5.15.0-58-generic-x86_64-with-glibc2.35
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:core:POX 0.8.0 (halosaur) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:openflow.discovery:Installing flow for 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
DEBUG:openflow.discovery:Installing flow for 00-00-00-00-00-03
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
DEBUG:openflow.discovery:Installing flow for 00-00-00-00-00-02
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.2 -> 00-00-00-00-00-01.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.3 -> 00-00-00-00-00-03.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.2 -> 00-00-00-00-00-02.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.2 -> 00-00-00-00-00-02.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.1 -> 00-00-00-00-00-01.1
```

Figura 23: link tra switch visti dal controller in situazione sotto attacco

4.3) Mitigazioni

Per contrastare l'attacco di tipo LLDP injection è stato modificato il modulo openflow.discovery inserendo del codice aggiuntivo per implementare una serie di mitigazioni. La prima si basa su HMAC, la seconda si basa su firma digitale con RSA e la terza, invece, utilizza sempre la firma digitale ma con crittografia a curve ellittiche.

4.3.1) HMAC

La mitigazione della LLDP injection può essere realizzata attraverso l'utilizzo dell'HMAC (Hashed Message Authentication Code), una tecnica di autenticazione basata su hash che consente di verificare l'integrità e l'autenticità dei messaggi scambiati.
Si è pensato di utilizzare HMAC con una funzione hash SHA-256.

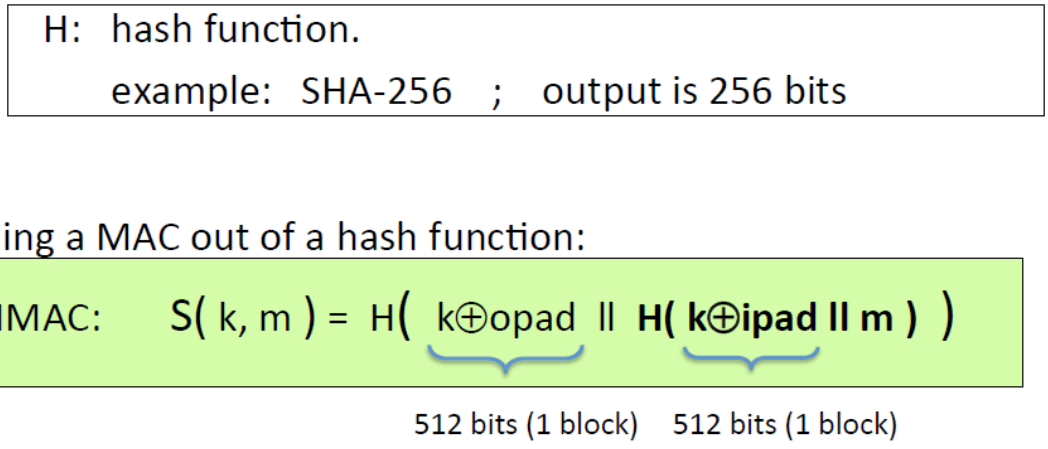

Figura 24: algoritmo HMAC

```
hmacKey = secrets.token_hex(16)

def hmacEncryption(port_id, chassis_id, ttl, key):
    m = port_id + chassis_id + str(ttl).encode()
    h = hmac.new(key.encode(), m, hashlib.sha256)
    return h.hexdigest()
```

Figura 25: codice python HMAC

La chiave K, di dimensione pari a 128 bit, generata in modo pseudocasuale, viene aggiornata ogni volta che il Controller inizia un nuovo ciclo di discovery sulla rete. Il messaggio M è formato dalla concatenazione di chassis_id, dpid e ttl.

```
mininet@mininet-vm:~/pox$ python3 pox.py --verbose openflow.discovery_mitigationHMAC
POX 0.8.0 (halosaur) / Copyright 2011-2022 James McCauley, et al.
DEBUG:core:POX 0.8.0 (halosaur) going up...
DEBUG:core:Running on CPython (3.10.6/Nov 14 2022 16:10:14)
DEBUG:core:Platform is Linux-5.15.0-58-generic-x86_64-with-glibc2.35
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:core:POX 0.8.0 (halosaur) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:openflow.discovery_mitigationHMAC:Installing flow for 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
DEBUG:openflow.discovery_mitigationHMAC:Installing flow for 00-00-00-00-00-03
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
DEBUG:openflow.discovery_mitigationHMAC:Installing flow for 00-00-00-00-00-02
INFO:openflow.discovery_mitigationHMAC:link detected: 00-00-00-00-00-02.2 -> 00-00-00-00-00-01.2
INFO:openflow.discovery_mitigationHMAC:link detected: 00-00-00-00-00-02.3 -> 00-00-00-00-00-03.2
INFO:openflow.discovery_mitigationHMAC:link detected: 00-00-00-00-00-03.2 -> 00-00-00-00-00-02.3
INFO:openflow.discovery_mitigationHMAC:link detected: 00-00-00-00-00-01.2 -> 00-00-00-00-00-02.2
ERROR:openflow.discovery_mitigationHMAC:L'HMAC presente nel pacchetto non corrisponde a quello ge
nerato in base ai campi di riferimento.
```

Figura 26: rilevamento pacchetto LLDP malevolo

L'utilizzo dell'HMAC consente di mitigare la minaccia della LLDP injection, in quanto i messaggi LLDP contraffatti non saranno in grado di superare l'autenticazione basata su HMAC e saranno rifiutati dalla rete, se l'attaccante provasse ad effettuare l'attacco mostrato prima non avrebbe alcun effetto, in quanto non sarebbe verificata l'integrità del frame LLDP.
La complessità derivata dall'utilizzo di HMAC non è molto elevata in quanto non vi è una gestione delle chiavi tra i diversi host, ma se ne occupa solo il controller.

4.3.2) Firma digitale basata su numeri primi (RSA)

Un secondo modo di mitigare LLDP injection può essere realizzato mediante l'utilizzo di una firma digitale, una tecnica di autenticazione basata su crittografia a chiave pubblica. In pratica, il mittente di un messaggio LLDP utilizza la propria chiave privata per creare una firma digitale del messaggio, che viene poi inviata insieme al messaggio stesso. Il destinatario riceve il messaggio e verifica la firma digitale utilizzando la chiave pubblica del mittente. Se la verifica della firma digitale ha successo, il destinatario può essere certo che il messaggio proviene dal mittente legittimo e che non è stato modificato durante la trasmissione.
L'utilizzo di una firma digitale consente quindi di mitigare la minaccia della LLDP injection, in quanto i messaggi LLDP contraffatti non saranno in grado di generare una firma digitale valida utilizzando la chiave privata del mittente legittimo, e saranno rifiutati dalla rete.
La generazione e la verifica della firma digitale richiedono una computazione crittografica che può comportare un aumento del carico di elaborazione nei dispositivi di rete.

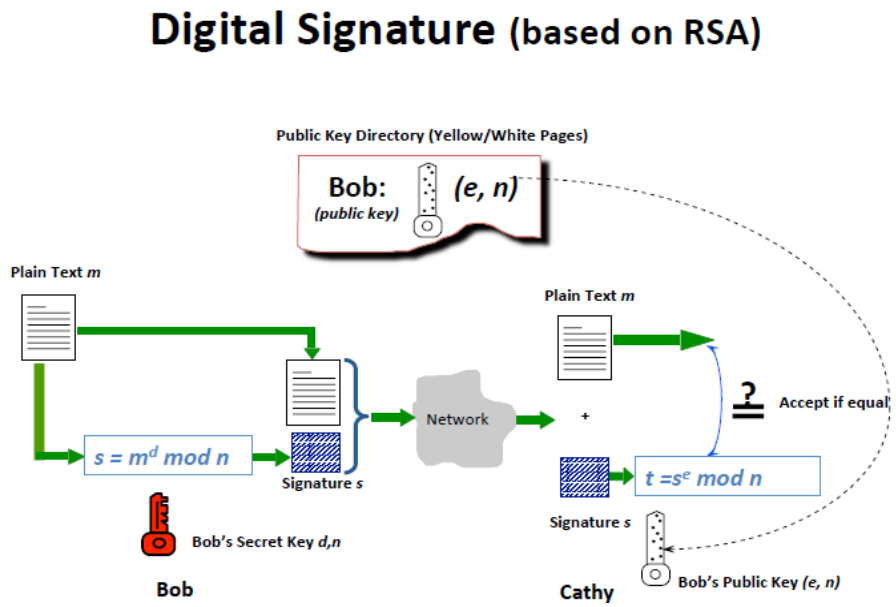

Figura 27: funzionamento firma digitale

```python
private_key=rsa.generate_private_key(public_exponent=65537, key_size=2048)

public_key=private_key.public_key()

data=port_id.id + chassis_id.id + str(ttl.ttl).encode()
firma = private_key.sign(data,padding.PSS(mgf=padding.MGF1(hashes.SHA256()),salt_length=padding.PSS.MAX_LENGTH),hashes.SHA256())
firma_str = base64.b64encode(firma).decode()
sysdesc.payload = ('dpid:' + hex(int(dpid))[2:] + ';' + firma_str).encode()     #[2:] per togliere 0x davanti il dpid

data=lldph.tlvs[1].id + lldph.tlvs[0].id + str(lldph.tlvs[2].ttl).encode()
signature = base64.b64decode(smac[1].encode())
try:
    public_key.verify(signature,data,padding.PSS(mgf=padding.MGF1(hashes.SHA256()),salt_length=padding.PSS.MAX_LENGTH),hashes.SHA256())
```

Figura 28: codice python generazione chiavi, firma dei dati e verifica della firma

Nel nostro caso è stata usata la firma digitale basata sull'algoritmo RSA con chiave a 2048 bit. La figura 29 rappresenta l'evoluzione della rete per quel che riguarda il protocollo LLDP. Si può notare che se si prova a generare un link fasullo tra due switch, attraverso un LLDP injection, questo viene rilevavo dalla mitigazione in atto e risulterà "firma non valida".

```
mininet@mininet-vm:~/pox$ python3 pox.py --verbose openflow.discovery_mitigationFirmaDigitaleRSA2048
POX 0.8.0 (halosaur) / Copyright 2011-2022 James McCauley, et al.
DEBUG:core:POX 0.8.0 (halosaur) going up...
DEBUG:core:Running on CPython (3.10.6/Nov 14 2022 16:10:14)
DEBUG:core:Platform is Linux-5.19.0-32-generic-x86_64-with-glibc2.35
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:core:POX 0.8.0 (halosaur) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:openflow.discovery_mitigationFirmaDigitaleRSA2048:Installing flow for 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
DEBUG:openflow.discovery_mitigationFirmaDigitaleRSA2048:Installing flow for 00-00-00-00-00-02
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
DEBUG:openflow.discovery_mitigationFirmaDigitaleRSA2048:Installing flow for 00-00-00-00-00-03
Signature is valid
INFO:openflow.discovery_mitigationFirmaDigitaleRSA2048:link detected: 00-00-00-00-00-02.2 -> 00-00-00-00-00-01.2
Signature is valid
INFO:openflow.discovery_mitigationFirmaDigitaleRSA2048:link detected: 00-00-00-00-00-02.3 -> 00-00-00-00-00-03.2
Signature is valid
INFO:openflow.discovery_mitigationFirmaDigitaleRSA2048:link detected: 00-00-00-00-00-03.2 -> 00-00-00-00-00-02.3
Signature is valid
Signature is valid
Signature is valid
Signature is valid
INFO:openflow.discovery_mitigationFirmaDigitaleRSA2048:link detected: 00-00-00-00-00-01.2 -> 00-00-00-00-00-02.2
Signature is valid
Signature is valid
Signature is valid
Signature is valid
Signature is valid
Signature is valid
Signature is valid
Signature is valid
Signature is invalid
Signature is valid
Signature is valid
Signature is valid
```

Figura 29: rilevamento firma non valida

4.3.3) Firma digitale basata su crittografia a curve ellittiche

La mitigazione della LLDP (Link Layer Discovery Protocol) injection può essere realizzata anche mediante l'utilizzo di una firma digitale basata sulle curve ellittiche crittografiche (ECC), una tecnologia di crittografia a chiave pubblica che utilizza curve ellittiche al posto dei numeri primi utilizzati in altre tecniche di crittografia.
In pratica, l'ECC permette di utilizzare chiavi più corte rispetto ad altre tecniche di crittografia, ma con la stessa forza crittografica.
L'utilizzo di una firma digitale basata sulle curve ellittiche crittografiche consente quindi di mitigare la minaccia della LLDP injection, in quanto i messaggi LLDP contraffatti non saranno in grado di generare una firma digitale valida utilizzando la chiave privata del mittente legittimo, e saranno rifiutati dalla rete.
Inoltre, l'utilizzo delle curve ellittiche crittografiche per la generazione di firme digitali è anche più sicuro rispetto ad altre tecniche di crittografia a chiave pubblica, in quanto è più resistente ad alcune tecniche di attacco crittografico come l'attacco a forza bruta e l'attacco di ricerca della chiave.
Sono state utilizzate due tipi di curve: le curve random e le curve di Koblitz che sono due tipi di curve ellittiche che possono essere utilizzate per la crittografia a chiave pubblica. Le curve di Koblitz sono caratterizzate dalla loro struttura particolare, mentre le curve random sono generate casualmente.
In generale, le curve di Koblitz sono considerate meno efficienti delle curve random in termini di prestazioni, poiché richiedono un maggior numero di operazioni per le operazioni aritmetiche di base (ad esempio la somma di due punti sulla curva) rispetto alle curve random. Tuttavia, la scelta tra l'utilizzo di curve di Koblitz o curve random dipende dalle esigenze specifiche dell'applicazione. Se la velocità è una priorità assoluta, le curve random sono generalmente la scelta migliore mentre se la memoria e le risorse computazionali sono limitate, le curve di Koblitz possono essere una scelta più vantaggiosa.

```
private_key = ec.generate_private_key(ec.SECP256R1(), default_backend())

public_key = private_key.public_key()

data=port_id.id + chassis_id.id + str(ttl.ttl).encode()
firma = private_key.sign(data, ec.ECDSA(hashes.SHA256()))
firma_str = base64.b64encode(firma).decode()
sysdesc.payload = ('dpid:' + hex(int(dpid))[2:] + ';' + firma_str).encode()     #[2:] per togliere 0x davanti il dpid

data=lldph.tlvs[1].id + lldph.tlvs[0].id + str(lldph.tlvs[2].ttl).encode()
signature = base64.b64decode(smac[1].encode())
try:
    public_key.verify(signature, data, ec.ECDSA(hashes.SHA256()))
```

Figura 30: codice python generazione chiavi, firma dei dati e verifica della firma con curve ellittiche

La figura 31 rappresenta l'evoluzione della rete per quel che riguarda il protocollo LLDP. Si può notare che se si prova a generare un link fasullo tra due switch, attraverso un LLDP injection, questo viene rilevato dalla mitigazione in atto e risulterà "firma non valida" come visto anche precedentemente.

```
mininet@mininet-vm:~/pox$ python3 pox.py --verbose openflow.discovery_mitigationCER256
POX 0.8.0 (halosaur) / Copyright 2011-2022 James McCauley, et al.
DEBUG:core:POX 0.8.0 (halosaur) going up...
DEBUG:core:Running on CPython (3.10.6/Nov 14 2022 16:10:14)
DEBUG:core:Platform is Linux-5.19.0-32-generic-x86_64-with-glibc2.35
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:core:POX 0.8.0 (halosaur) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:openflow.discovery_mitigationCER256:Installing flow for 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
DEBUG:openflow.discovery_mitigationCER256:Installing flow for 00-00-00-00-00-03
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
DEBUG:openflow.discovery_mitigationCER256:Installing flow for 00-00-00-00-00-02
Signature is valid
INFO:openflow.discovery_mitigationCER256:link detected: 00-00-00-00-00-02.2 -> 00-00-00-00-00-01.2
Signature is valid
INFO:openflow.discovery_mitigationCER256:link detected: 00-00-00-00-00-02.3 -> 00-00-00-00-00-03.2
Signature is valid
INFO:openflow.discovery_mitigationCER256:link detected: 00-00-00-00-00-03.2 -> 00-00-00-00-00-02.3
Signature is valid
Signature is valid
Signature is valid
Signature is valid
INFO:openflow.discovery_mitigationCER256:link detected: 00-00-00-00-00-01.2 -> 00-00-00-00-00-02.2
Signature is valid
Signature is valid
Signature is valid
Signature is valid
Signature is valid
Signature is valid
Signature is valid
Signature is valid
Signature is invalid
Signature is valid
```

Figura 31: rilevamento firma non valida

Per semplicità, viene riportato solamente il funzionamento di una delle quattro curve ellittiche studiate poiché è pressoché identico per ognuna e la loro differenza è stata osservata in termini di prestazioni.

4.4) Valutazione prestazioni

In generale, la scelta tra l'utilizzo dell'HMAC, della firma digitale basata su numeri primi della firma digitale basata su crittografia a curve ellittiche dipende dalle specifiche esigenze di sicurezza e dalle restrizioni operative della rete in questione. Se la sicurezza è la massima priorità e non ci sono vincoli operativi, la firma digitale potrebbe essere la scelta migliore. Se invece si richiede un approccio più efficiente e semplice, l'HMAC può essere una buona alternativa. La crittografia a curve ellittiche potrebbe essere una scelta ideale in ambienti con risorse limitate, come dispositivi IoT o reti a basso consumo energetico.

Attraverso l'utilizzo di Wireshark, abbiamo raccolto i dati relativi alla larghezza di banda utilizzata per ciascun tipo di mitigazione, in un determinato periodo di tempo uguale per tutti. Come si può notare, la mitigazione che fa un maggiore uso di banda risulta essere la firma digitale, poiché la chiave utilizzata dall'algoritmo è di 2048 bit. Per la firma digitale basata su curve ellittiche il consumo di banda è inferiore poiché le chiavi utilizzate hanno una lunghezza più piccola. In particolare, sulla firma digitale basata sulla crittografia a curve ellittiche si può notare che un incremento della lunghezza della chiave porta ad un aumento di consumo di banda. La mitigazione con l'utilizzo di HMAC risulta essere la meno onerosa rispetto a tutte le altre.

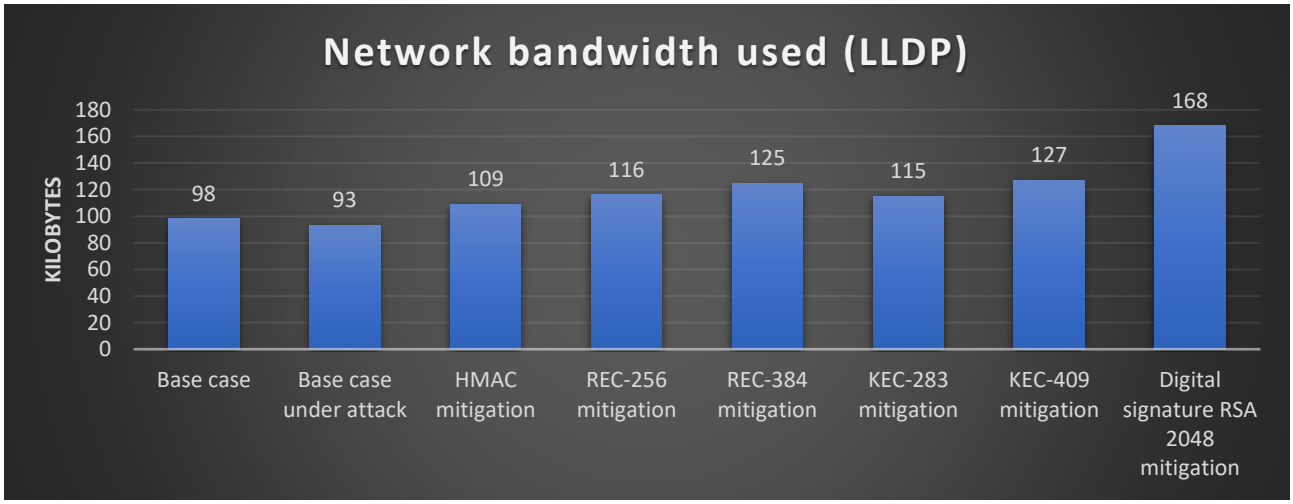

Figura 32: banda utilizzata

Per quanto riguarda il tempo di generazione e firma di un frame LLDP, sono stati raccolti una serie di dati sui tempi e fatta una media. Si può osservare dal grafico in figura 33 che a causa della generazione di un messaggio per la verifica dell'integrità dei dati il tempo

aumenta rispetto al caso base dove non è presente nessuna verifica di integrità dei dati. In particolare, si può osservare come il tempo aumenti da sinistra verso destra passando dal caso base all'HMAC e poi alle curve ellittiche random fino alla chiave con 384 bit. Anche per le curve ellittiche di Koblitz (che risultano essere più lente delle curve random) ogni volta che si aumenta la lunghezza della chiave il tempo di generazione e firma del frame aumenta. Infine, si può notare come la firma digitale permetta di firmare velocemente i documenti, molto più velocemente di quasi tutte le curve ellittiche osservate (a parte la curva ellittica random a 256 bit). I risultati del nostro test sono in accordo con risultati sperimentali trovati in rete e riportati in figura 34. In quest'ultima figura si può vedere un confronto sulla velocità di firma in base alla lunghezza delle chiavi dei due tipi di crittografia: a curva ellittica (ECC) e basata su numeri primi (RSA). Si può notare come con chiavi piccole la firma di dati sia più veloce con RSA rispetto ad ECC mentre al crescere della chiave RSA diventa più lenta di ECC.

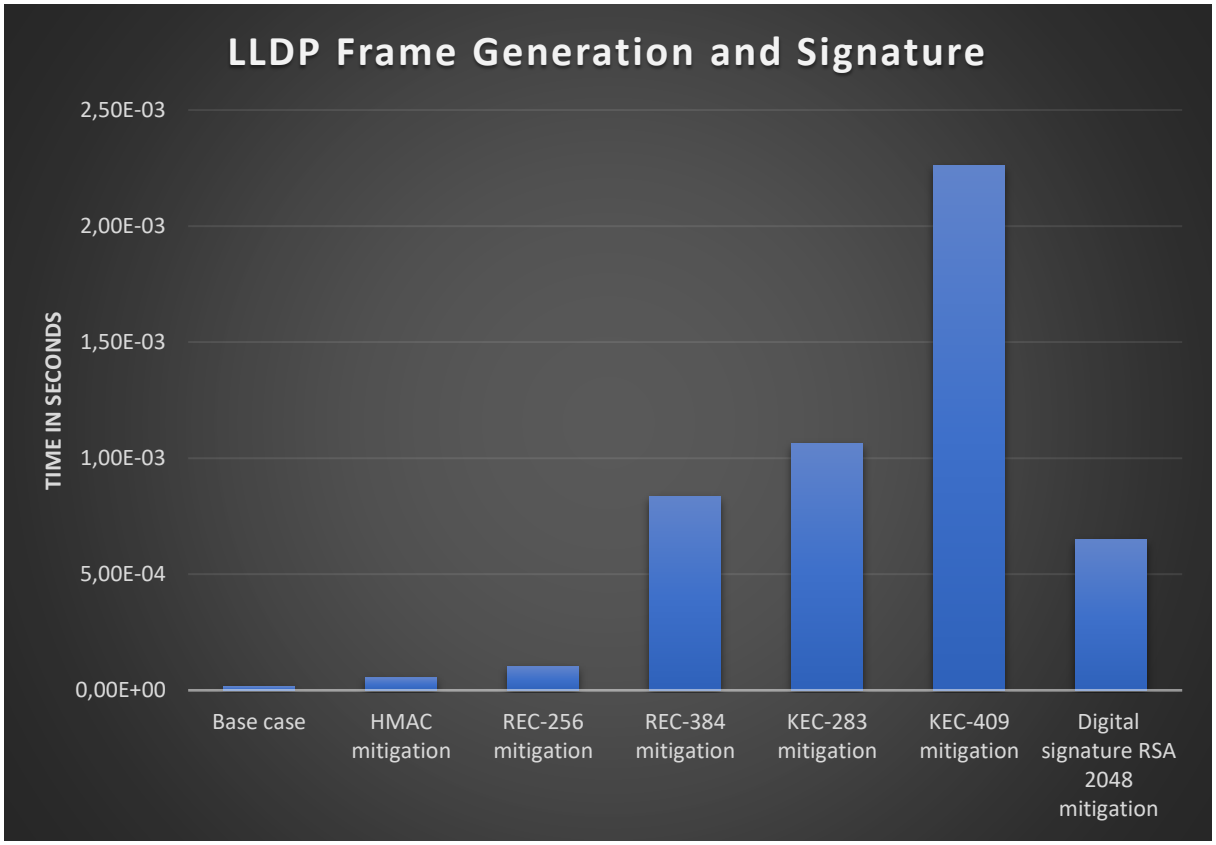

Figura 33: tempo di generazione frame LLDP

5-3: Signature generation performance

Signing	Key Length		Time (s)	
	ECC	RSA	ECC	RSA
	163	1024	0.15	0.01
	233	2240	0.34	0.15
	283	3072	0.59	0.21
	409	7680	1.18	1.53
	571	15360	3.07	9.20

Figura 34: confronto velocità firma con ECC ed RSA

5-4: Signature verification performance

Verification	Key Length		Time (s)	
	ECC	RSA	ECC	RSA
	163	1024	0.23	0.01
	233	2240	0.51	0.01
	283	3072	0.86	0.01
	409	7680	1.80	0.01
	571	15360	4.53	0.03

Figura 35: confronto verifica validità firma con ECC ed RSA

Anche qui, per quanto riguarda il tempo di processamento e verifica di integrità di un frame LLDP sono stati raccolti una serie di dati sui tempi e fatta una media. Si può osservare dal grafico in figura 36 che l'andamento è pressoché identico a quello della generazione e firma del frame. La verifica rimane più veloce di tutte sulla mitigazione HMAC ed anche qui la verifica della validità della firma digitale basata su RSA si dimostra più veloce di quella basata su curve ellittiche come riportato anche in figura 35 con un piccolo confronto. In questo caso, al contrario della generazione e firma dei dati, l'aumento della lunghezza della chiave non influenza RSA che rimane più veloce di ECC sulla verifica della validità della firma.

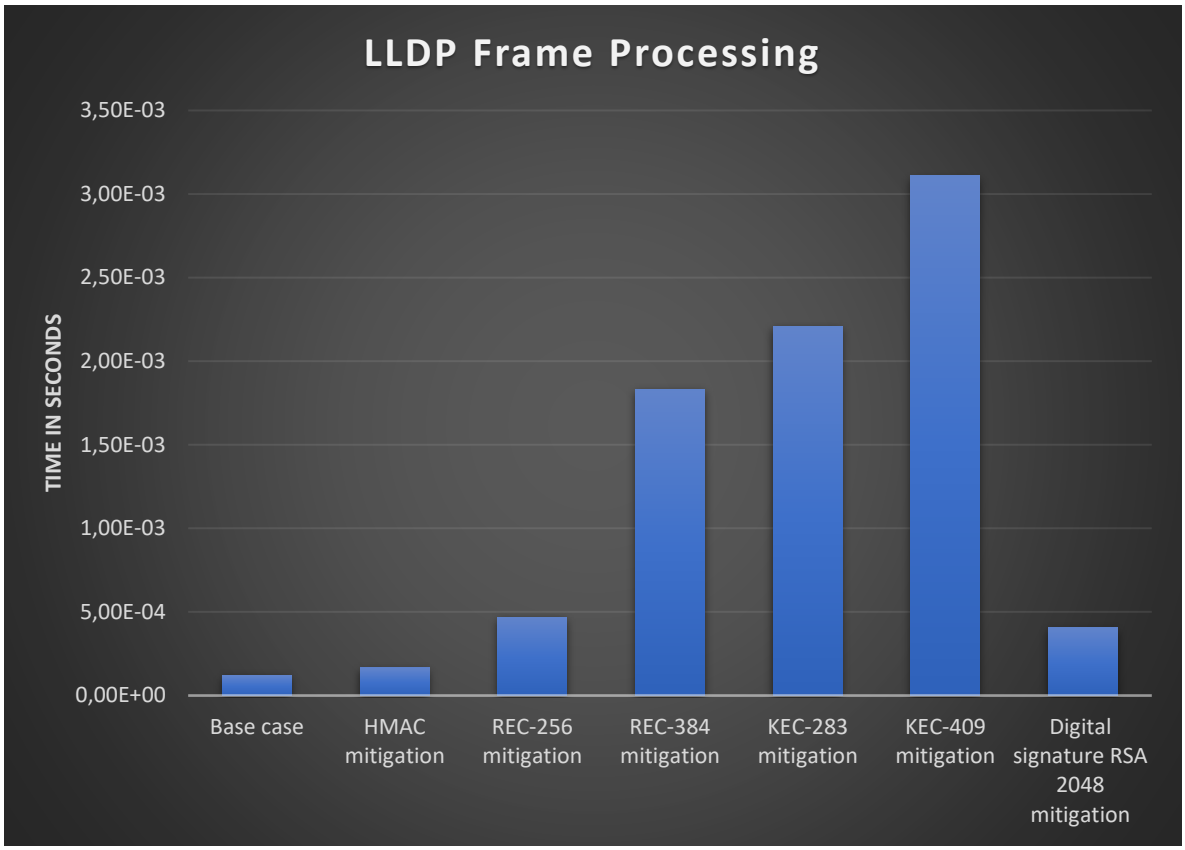

Figura 36: tempo di processamento frame LLDP

Infine, è riportato un grafico che somma i tempi medi di generazione e processamento dei frame LLDP che ovviamente è speculare ai primi due.

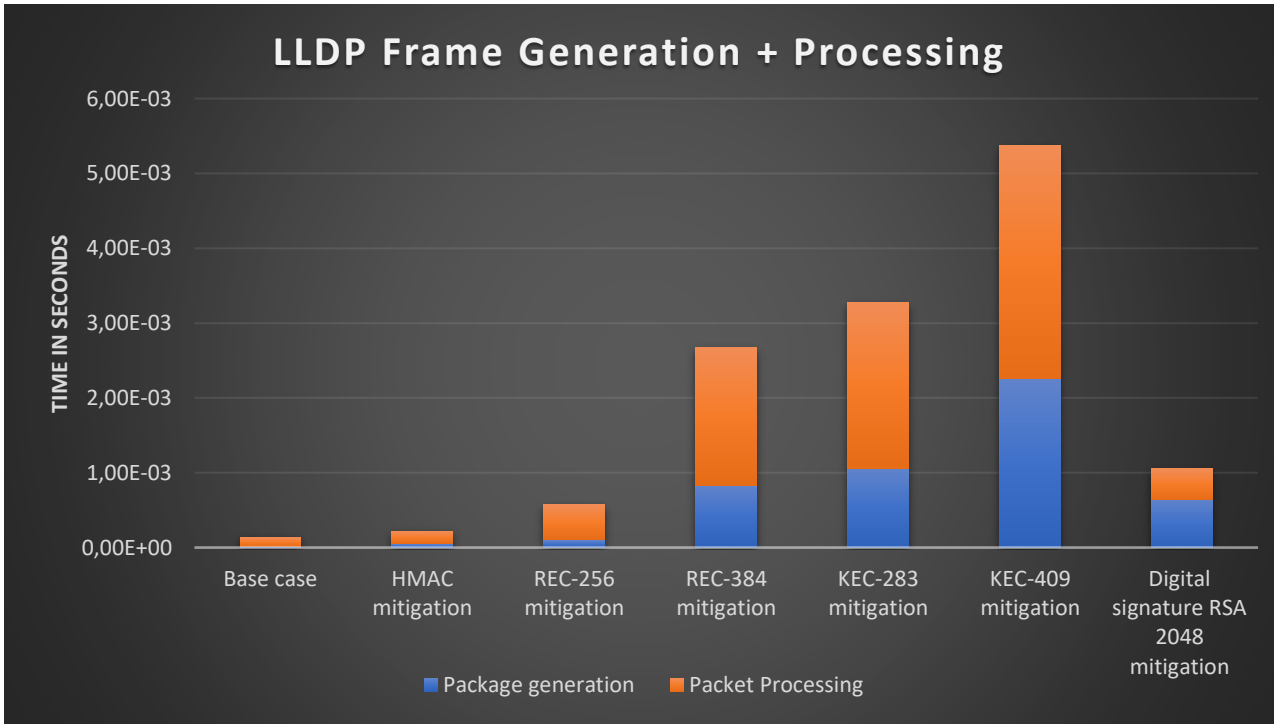

Figura 37: tempo di generazione più processamento frame LLDP

I dati relativi all'utilizzo della CPU, anche per lo studio del protocollo LLDP, sono stati raccolti tramite uno script sviluppato appositamente.
L'utilizzo della CPU con in esecuzione il modulo con la mitigazione dell'HMAC risulta essere quasi identico rispetto al caso base senza attacco e senza mitigazione attiva. Questo perché l'HMAC non aggiunge complessità in termini di carico di lavoro, essendo efficiente e semplice da implementare.

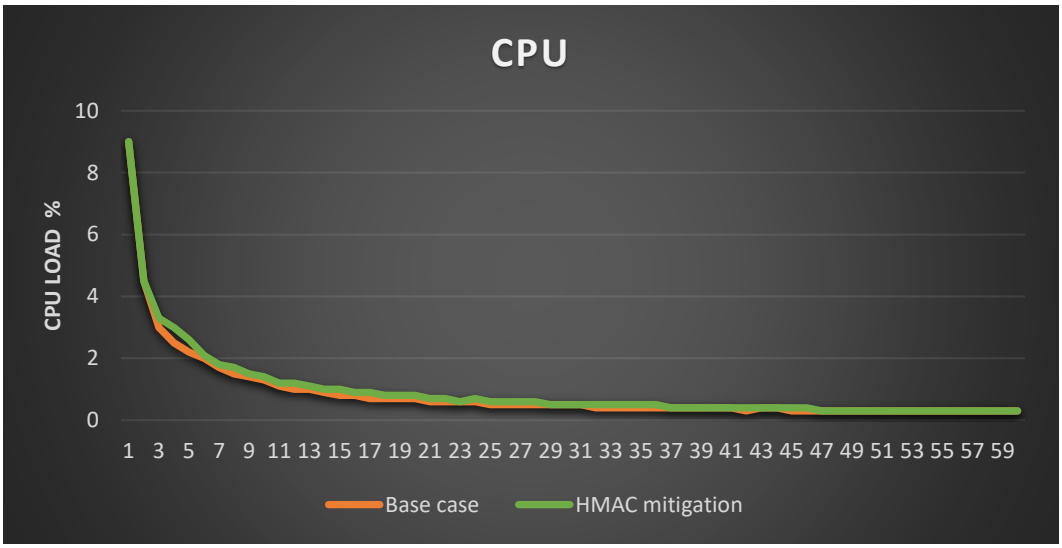

Figura 38: utilizzo CPU

L'utilizzo della CPU con in esecuzione il modulo con la mitigazione con firma digitale basato su curve ellittiche (in questo caso quella random a 256) aumenta specialmente nella fase iniziale per poi andare ad assottigliarsi col tempo.

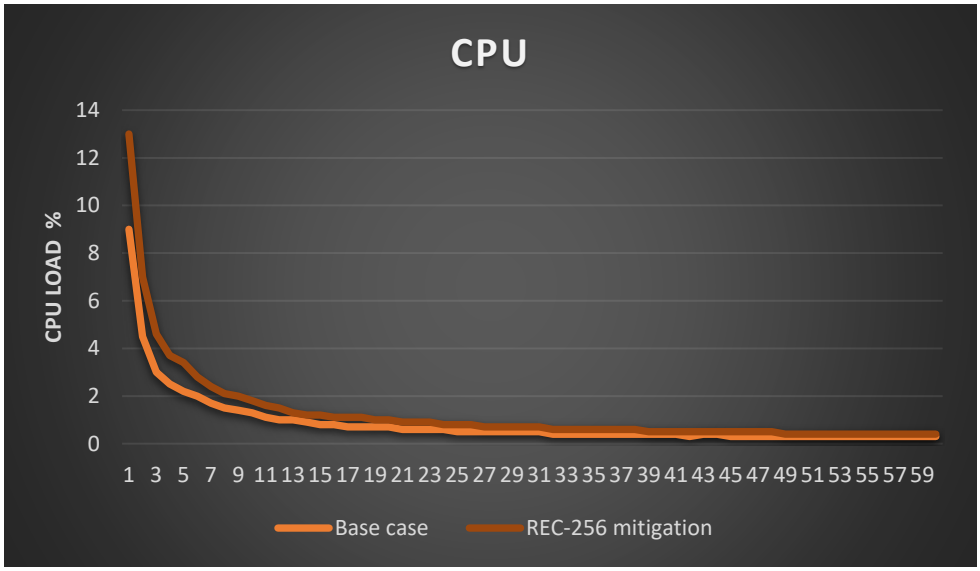

Figura 39: utilizzo CPU

Cambiando il tipo di curva ellittica e passando a una curva di Koblitz con con chiave a 283 bit il comportamento del carico sulla CPU è analogo alla curva random descritta precedentemente.

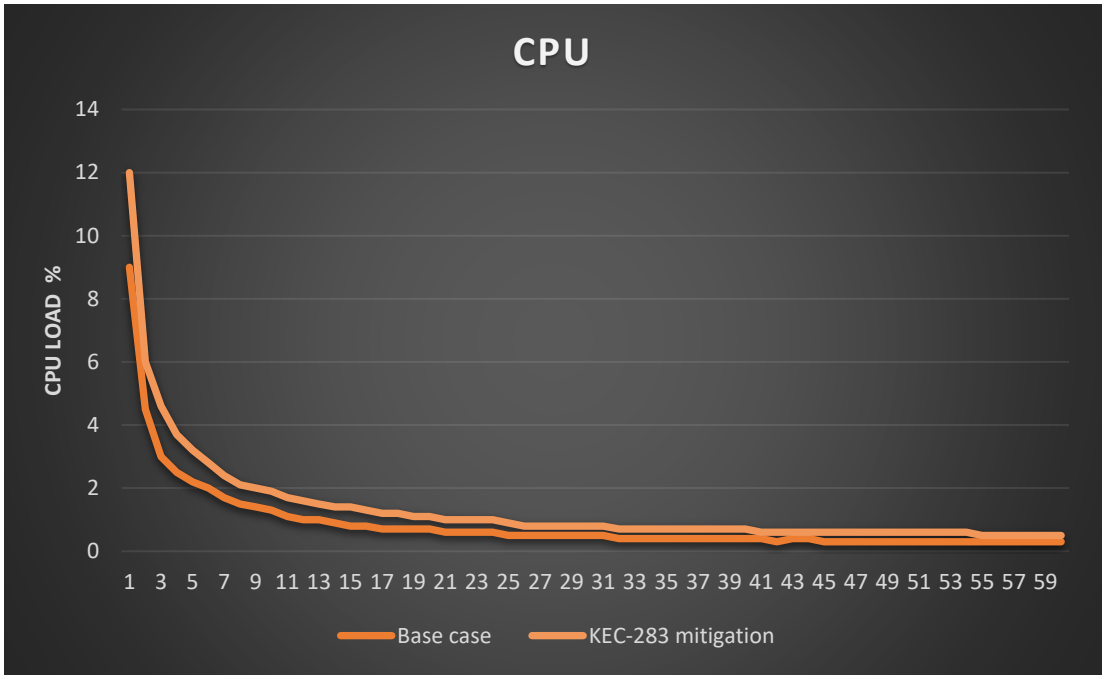

Figura 40: utilizzo CPU

L'utilizzo della CPU con in esecuzione il modulo con mitigazione della firma digitale basata su numeri primi (RSA 2048 bit) risulta essere invece molto più oneroso in termini di carico sul processore rispetto al case base (senza attacco e senza mitigazione). Questo perché la firma digitale per garantire maggiore sicurezza utilizza più risorse.

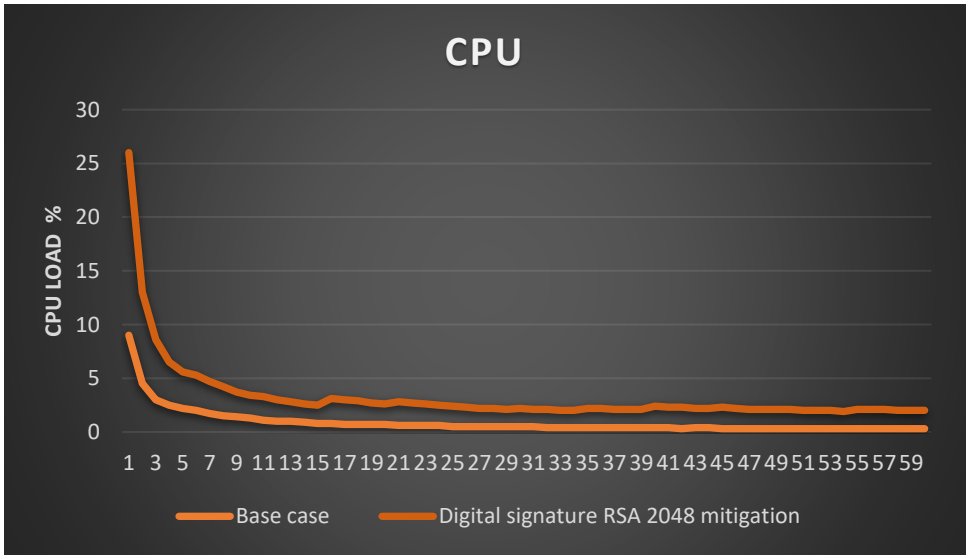

Figura 41: utilizzo CPU

Nel grafico in figura 42 è stato evidenziato il carico sul processore per quanto riguarda il controller analizzando solo i moduli con le mitigazioni con curve ellittiche random. A livello di carico sulla CPU la differenza è così piccola da non notarsi. In generale, l'inserimento della mitigazione non porta ad un carico eccessivo per il processore.

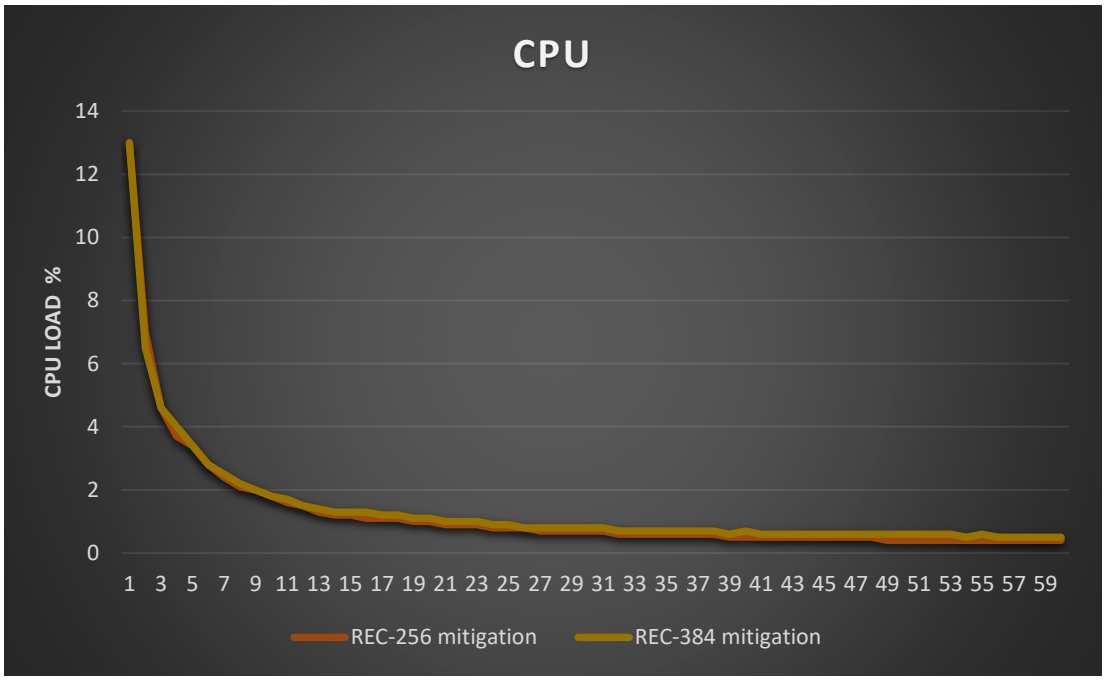

Figura 42: utilizzo CPU

Il discorso appena fatto per le curve random vale anche per le curve di Koblitz. Ciò si può notare dal grafico in figura 43.

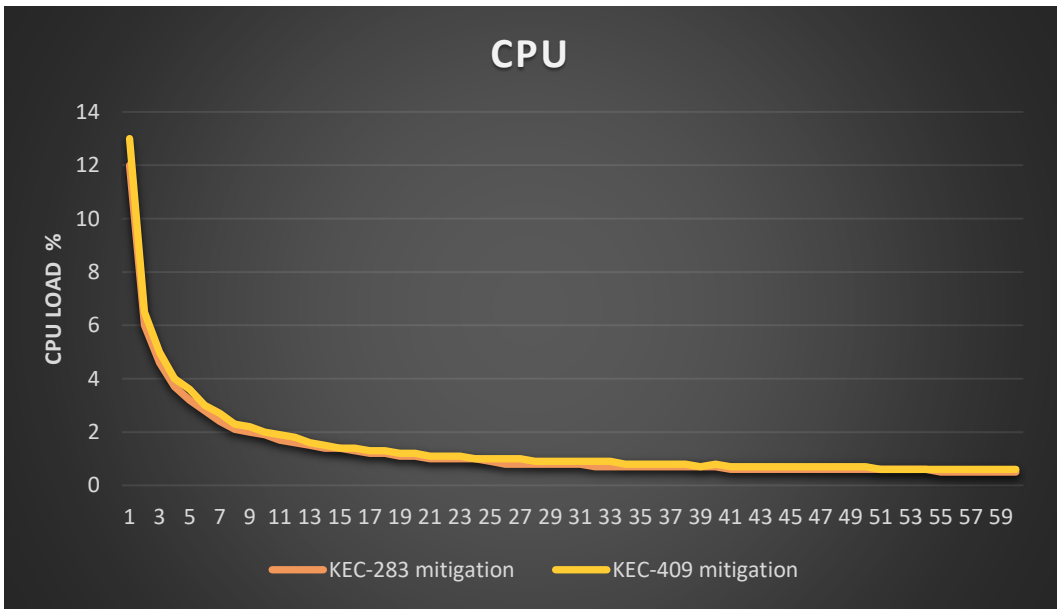

Figura 43: utilizzo CPU

Nel grafico in figura 44, invece, sono stati confrontate le seguenti quattro mitigazioni (HMAC, firma digitale basata sui numeri primi (RSA 2048 bit), firma digitale basata su crittografia a curve ellittiche random con chiave a 256 bit e firma digitale basata su curve ellittiche di Koblitz con chiave a 283). Come già descritto in precedenza, si mostra la differenza di carico sul processore osservando che la firma digitale basata su numeri primi supera di gran lunga le altre tre.

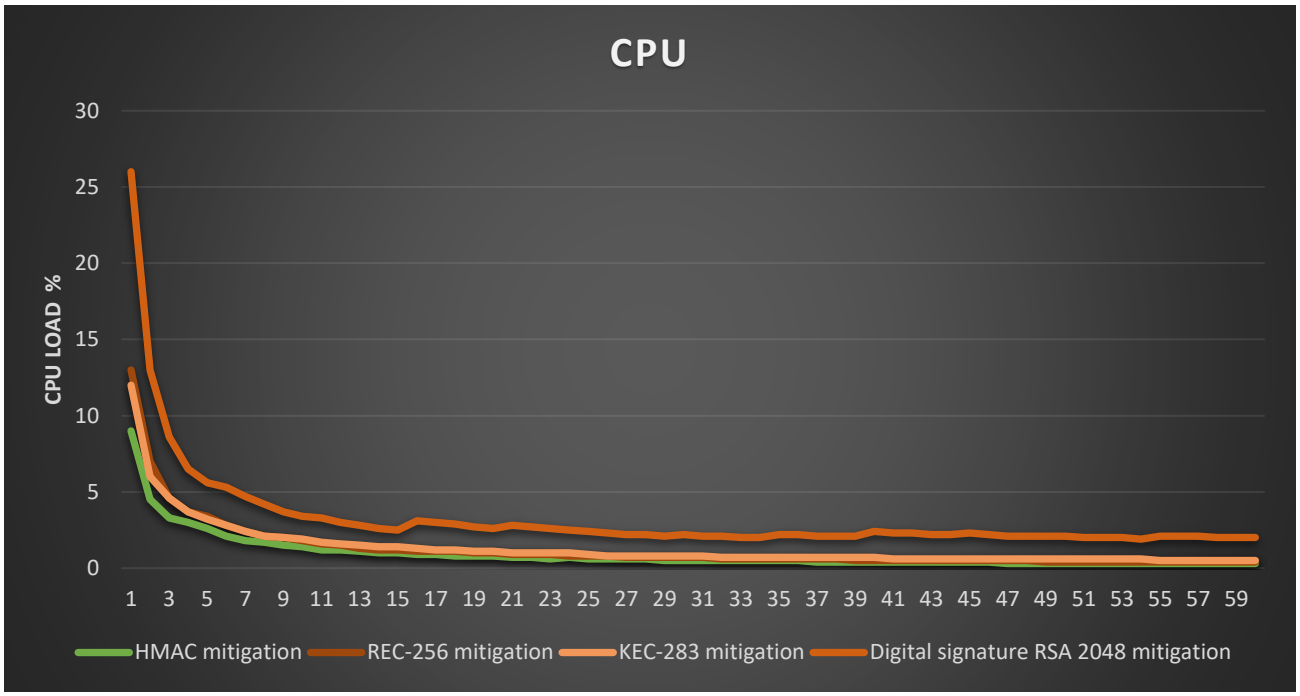

Figura 44: utilizzo CPU

Nel grafico in figura 45 sono stati confrontati i dati raccolti sull'uso della memoria (RAM) da parte del controller. In questo caso non c'è differenza tra la firma digitale e le curve ellittiche, che utilizzano un carico di memoria di poco più elevato rispetto al caso base e all'HMAC. Questo perché nell'HMAC viene utilizzata una chiave a 128 bit, mentre negli altri casi la chiave è 256 o 384 per la firma digitale basata sulle curve ellittiche random, 283 o 409 per la firma digitale basata su curve ellittiche di Koblitz e 2048 per la firma digitale basata su RSA.

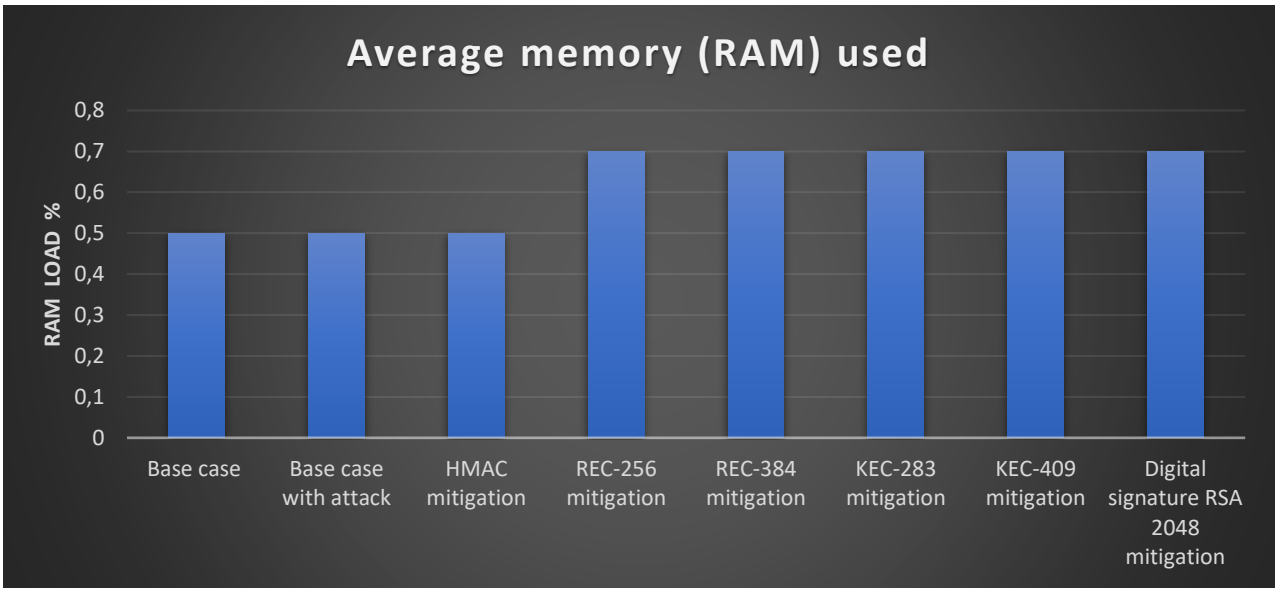

Figura 45: memoria RAM media utilizzata

Attraverso l'utilizzo di Wireshark è stato possibile analizzare la rete nelle diverse situazioni: caso base, caso base sotto attacco e attacco con le mitigazioni attive. Abbiamo osservato che l'andamento dello scambio di pacchetti nella rete è pressoché identico in ogni situazione poiché l'attacco si basa sull'invio di un solo pacchetto, che non influenza in alcun modo la rete. Per questo motivo, anche le mitigazioni inserite non influenzano la rete dal punto di vista di pacchetti scambiati poiché c'è un solo pacchetto che viene rifiutato e quindi di seguito viene riportato un solo grafico per tutti i casi.

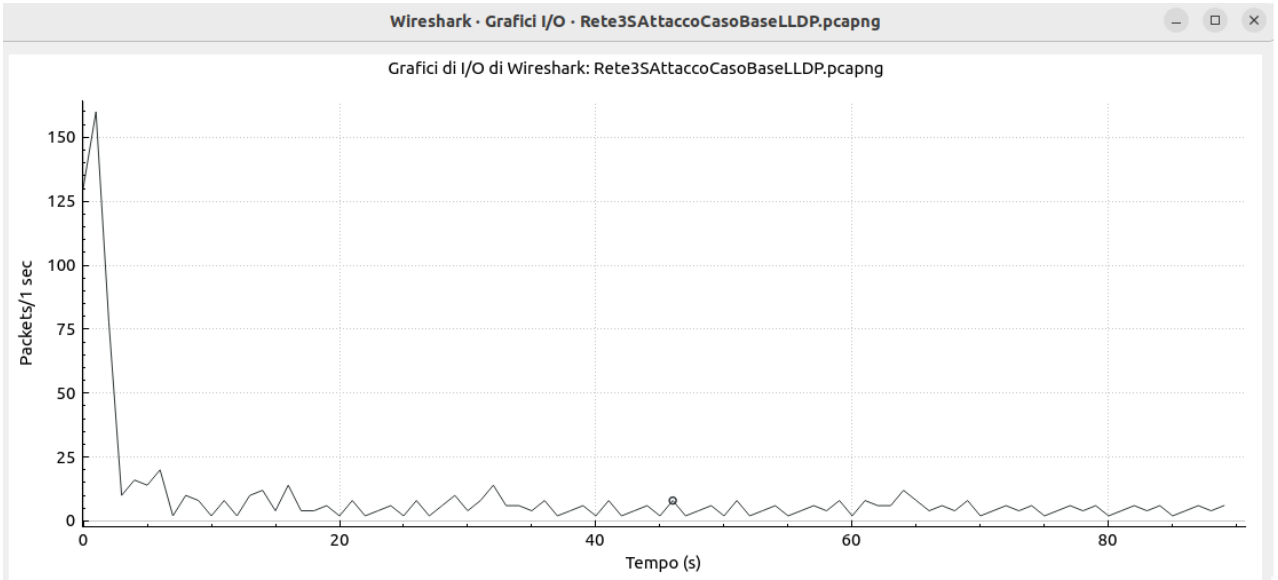

Figura 46: scambio pacchetti nel tempo

5) Conclusioni

Il progetto sviluppato ci ha permesso di ottenere delle competenze in ambito di sicurezza sulle reti, in particolare di entrare in contatto con le reti SDN che fino ad ora non avevamo mai avuto modo di vedere in nessun ambito didattico.

Tramite questo lavoro abbiamo imparato a configurare un'ambiente che consenta la simulazione di reti SDN su macchina virtuale toccando con mano vari problemi dovuti all'installazione e al settaggio dello stesso. Nello specifico, abbiamo assunto competenze su Mininet con controller Pox per l'implementazione delle reti.

Attraverso articoli accademici abbiamo effettuato una serie di analisi su problematiche di sicurezza a cui sono soggette le reti SDN e in seguito abbiamo deciso di trattare alcuni attacchi e sviluppare delle mitigazioni contro gli stessi riuscendo ad ottenere dei risultati per noi soddisfacenti.

Questa esperienza ha aggiunto del bagaglio culturale alla nostra preparazione in ambito di sicurezza informatica. Il progetto fin qui sviluppato sotto forma di macchina virtuale può essere la base per estendere la ricerca di problematiche su reti SDN e vedere ulteriori attacchi e mitigazioni.