Università degli Studi di Perugia

Dipartimento di Ingegneria

ingegneria informatica e robotica

Uso di Mininet distribuito per creare una rete monitorata con sFlow

Dervishaj Ergys

Luca Fagiolo

Indice

1. Mininet

Mininet è un emulatore di rete in grado di simulare una serie di host, switch, router e collegamenti su un singolo kernel Linux. Nel concreto consente di simulare una rete completa utilizzando la virtualizzazione leggera per creare una o più reti virtuali che replicano il comportamento di una rete fisica. Un host Mininet si comporta come una macchina reale, può essere acceduta tramite ssh e può eseguire programmi arbitrari (incluso tutto ciò che è installato nel sistema Linux sottostante). I programmi che si eseguono possono inviare pacchetti attraverso interfacce Ethernet simulate, con una velocità di collegamento e un ritardo specifici. I pacchetti vengono elaborati da switch Ethernet, router o middlebox simulati con una prestabilita capacità di accodamento. Gli switch sono programmabili mediante l'utilizzo di OpenFlow che consente di definire regole di inoltro dei pacchetti in modo dinamico attraverso un controllore SDN. E' inoltre possibile l'integrazione con OpenVSwitch per l'implementazione di uno switch virtuale consentendo una programmabilità avanzata e una gestione centralizzata degli switch. La rete simulata creata con Mininet risulta essere isolata rispetto alla rete LAN in cui è connesso il calcolatore. E' possibile abilitare l'utilizzo del NAT per permettere ai componenti della rete emulata di connettersi alla LAN ed eventualmente collegarsi ad internet. Un'altra possibilità è anche quella di collegare un'interfaccia reale alla rete simulata. Mininet permette di definire in modo rapido e veloce una topologia di rete. Questo può essere fatto tramite comando linux andando a richiamare una topologia predefinita o definendo una topologia custom tramite uno script in python. La creazione di una topologia, indipendentemente dal modo in cui è definita, richiede pochissimi secondi e può essere riutilizzata per replicare i risultati in altri calcolatori favorendo le possibilità di deployment.

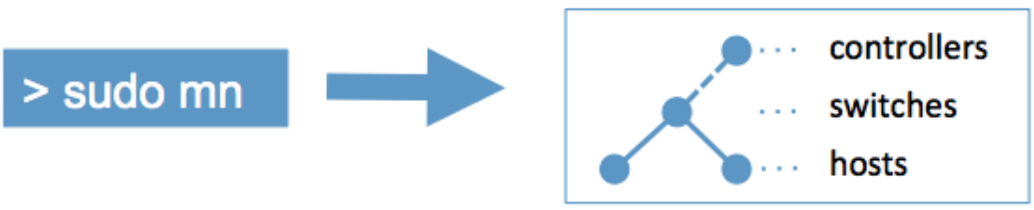

Figura 1.1

Di seguito sono elencate le principali classi e funzioni in python per la definizione di una topologia di rete:

- **Topo**: classe che fa da base per la costruzione di una topologia Mininet
- **build()**: metodo sovrascritto che va a creare la topologia
- **addSwitch()**: aggiunge uno switch alla topologia e ritorna il nome dello switch
- **addHost()**: aggiunge un host alla topologia e ritorna il nome dell'host
- **addLink()**: aggiunge un link bidirezionale alla topologia
- **Mininet**: classe di Mininet utilizzata per generare e gestire una rete
- **start()**: funzione della classe Mininet che inizializza la rete
- **pingAll()**: funzione che testa la connettivita tra i vari nodi realizzando ping reciproci
- **stop()**: funzione che termina l'esecuzione della rete

2. sFlow

sFlow è un protocollo di monitoraggio di rete progettato per offrire informazioni dettagliate sul traffico in tempo reale. L'obiettivo principale di sFlow è semplificare la gestione delle reti, consentendo agli amministratori di ottenere una visione chiara delle prestazioni senza dover analizzare ogni singolo pacchetto. Il nome deriva dall'abbreviazione di "sampled flow" e ne riassume i concetti fondamentali. Infatti il funzionamento di sFlow si basa sul concetto di flusso, ovvero l'approssimazione di una sessione TCP. Per essere più esatti un flusso è una raccolta di pacchetti con indirizzi identici (Livello 3/Livello 4) che sono strettamente raggruppati nel tempo. A differenza di altri protocolli di monitoraggio basati sui flussi, sFlow adotta un approccio di campionamento deterministico, in cui i dispositivi di rete selezionano periodicamente un sottoinsieme di pacchetti in transito attraverso di essi. Questi pacchetti campionati vengono utilizzati per estrarre informazioni dettagliate sul traffico, consentendo agli amministratori di rete di identificare tendenze, individuare problemi di performance, e ottimizzare la configurazione di rete.
Questo protocollo può essere implementato su una varietà di dispositivi di rete, tra cui switch e router, consentendo agli amministratori di ottenere una visione globale delle prestazioni di rete. La sua efficienza in ambienti congestionati e ad alta velocità lo rende particolarmente adatto per reti complesse.
sFlow offre dati ad alta granularità, consentendo di analizzare indirizzi IP, protocolli, interfacce di rete e altri attributi del traffico. Questa varietà di informazioni facilita la risoluzione dei problemi di rete, fornendo una visione completa delle dinamiche di traffico. Il campionamento dei pacchetti comporta diversi vantaggi, come l'utilizzo ridotto delle risorse di calcolo e l'elevata tempestività nell'ottenimento delle informazioni; tuttavia i metodi di campionamento potrebbero non essere in grado di rilevare flussi di dimensioni ridotte, e potrebbero mancare eventi di rete come picchi di traffico.
sFlow è spesso integrato con strumenti di gestione e monitoraggio di rete, semplificando ulteriormente l'analisi dei dati raccolti. Questa integrazione consente agli amministratori di rete di prendere decisioni informate, ottimizzare la capacità di rete e risolvere rapidamente eventuali problematiche.

Generalmente una soluzione basata sull'utilizzo di sFlow prevede due entità principali:

- dispositivi di rete (switch o router) che agiscono come agenti sFlow che monitorano il traffico ed inviano dati sFlow
- applicazioni software che ricevono e analizzano i dati sFlow.

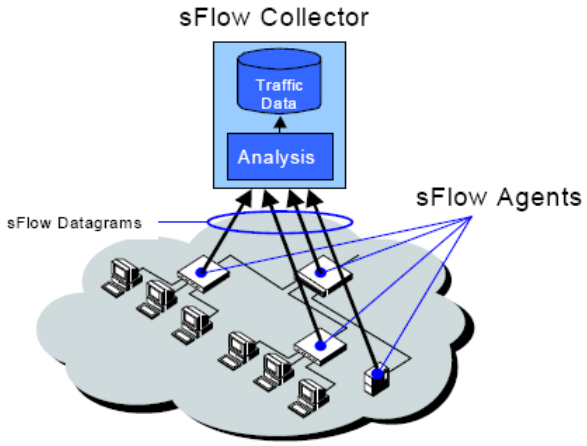

Figura 2.1: architettura generica di un contesto sFlow_Mininet

2.1 sFlow-RT

Un esempio di applicazione software che permette la visualizzazione e analisi dei dati sFlow è **sFlow-RT** andando a convertire le misurazioni grezze inviate dagli agenti e convertendole in misurazioni operative accessibili tramite RESTFlow API.

Gli ambiti di applicazione di sFlow risultano essere diversi:

- **Risoluzione dei problemi di rete**: sFlow rende visibili pattern di traffico anomali con sufficiente dettaglio per consentire una rapida identificazione, diagnosi e correzione.
- **Controllo della congestione**: il monitoraggio continuo dei flussi di traffico su tutte le porte può essere utilizzato per evidenziare istantaneamente collegamenti congestionati, identificare la fonte del traffico e limitarne la velocità o priorità in base alla tipologia.
- **Sicurezza**: sFlow rende visibili pattern di traffico anomali con sufficiente dettaglio per consentire una rapida identificazione, diagnosi e correzione.
- **Profilazione delle rotte**: poiché sFlow contiene informazioni di inoltro, può essere utilizzato per profilare le rotte più attive e i flussi specifici trasportati da queste rotte. Queste informazioni vengono poi utilizzate per ottimizzare l'instradamento, migliorando la connettività e le prestazioni.

3. Esperimento

3.1 Preparazione del sistema

Per l'esecuzione degli esperimenti sono state utilizzate due macchine virtuali, di seguito viene descritto il processo di inizializzazione per ognuna di esse.

La prima macchina virtuale è "VM_Mininet" e ha lo scopo di contenere la simulazione della rete di host. Per questa VM sono state abilitate tre schede di rete, una connessa alla scheda del PC host tramite NAT, mentre le altre due sono connesse alla rete interna dove sarà poi collegata anche la seconda VM.

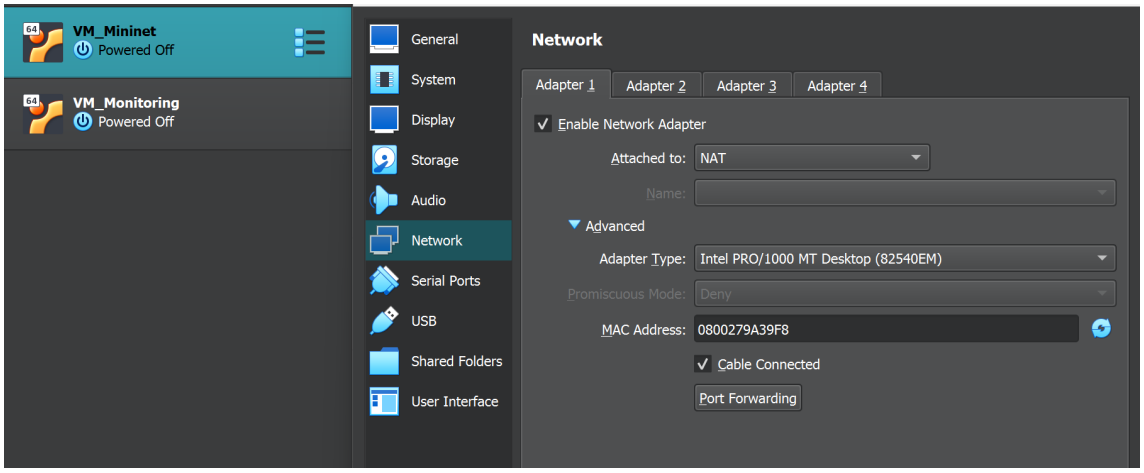

Figura 3.1: Scheda di rete con NAT della VM_Mininet (enp0s3)

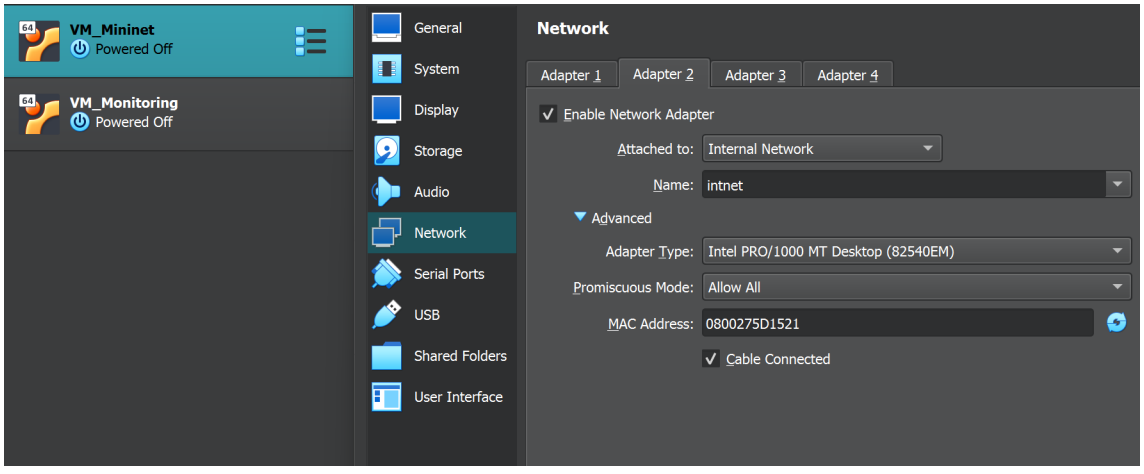

Figura 3.2: Scheda di rete interna della VM_Mininet (enp0s8)

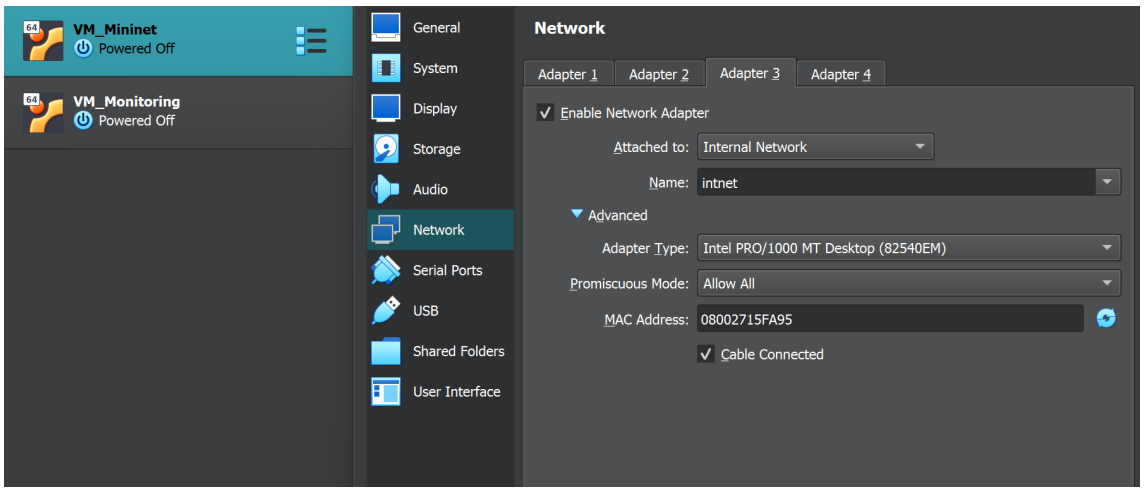

Figura 3.3: Scheda di rete interna della VM_Mininet (enp0s9)

3.2 Creazione della rete con Mininet

Nell'ambiente di Ubuntu 22.04 è stato installato Mininet tramite il comando *sudo apt-get install mininet*. Successivamente è stato creato il seguente file Python in grado di generare una topologia personalizzata con lo scopo di testare le caratteristiche di sFlow. La topologia della rete creata con Mininet è costituita da due reti private 10.0.0.0/24 e 20.0.0.0/24

```
1  from mininet.topo import Topo
2  from mininet.net import Mininet
3  from mininet.nodelib import NAT
4  from mininet.node import Node
5  from mininet.log import setLogLevel
6  from mininet.cli import CLI
7  from mininet.link import Intf
8
9  class LinuxRouter(Node):
10     def config(self, **params):
11         super(LinuxRouter, self).config(**params)
12         self.cmd('sysctl net.ipv4.ip_forward=1')
13
14     def terminate(self):
15         self.cmd('sysctl net.ipv4.ip_forward=0')
16         super(LinuxRouter, self).terminate()
17
18
19 class InternetTopo(Topo):
20     def build(self, **_kwargs):
21         r0 = self.addNode('r0', cls=LinuxRouter)
```

```
22
23          switch1 = self.addSwitch('s1')
24          h1 = self.addHost('h1', ip='10.0.0.10/24', defaultRoute='via
                10.0.0.1')
25          h2 = self.addHost('h2', ip='10.0.0.20/24', defaultRoute='via
                10.0.0.1')
26          self.addLink(h1, switch1)
27          self.addLink(h2, switch1)
28
29          switch2 = self.addSwitch('s2')
30          h3 = self.addHost('h3', ip='20.0.0.30/24', defaultRoute='via
                20.0.0.1')
31          h4 = self.addHost('h4', ip='20.0.0.40/24', defaultRoute='via
                20.0.0.1')
32          self.addLink(h3, switch2)
33          self.addLink(h4, switch2)
34
35          self.addLink(r0, switch1, intfName1='r0-eth1', params1={'ip':
                '10.0.0.1/24'})
36          self.addLink(r0, switch2, intfName1='r0-eth2', params1={'ip':
                '20.0.0.1/24'})
37
38
39  def run():
40      topo = InternetTopo()
41      net = Mininet(topo=topo, waitConnected=True)
42      net.start()
43      routerNode = net.getNodeByName("r0")
44      _intf0 = Intf('enp0s3', node=routerNode)
45      routerNode.cmd('dhclient enp0s3')
46      routerNode.cmd('ifconfig r0-eth1 10.0.0.1 netmask 255.255.255.0')
47      routerNode.cmd('ifconfig r0-eth2 20.0.0.1 netmask 255.255.255.0')
48      routerNode.cmd('iptables -t nat -A POSTROUTING -o enp0s3 -j
            MASQUERADE')
49      CLI(net)
50      net.stop()
51
52
53  if __name__ == '__main__':
54      setLogLevel('info')
55      run()
```

Alla riga 23 viene creato un Router r0 definito precedentemente con la classe

LinuxRouter. Dalla riga 25 alla 29 e dalla 31 alla 35 vengono creati rispettivamente i due Switch s1 ed s2 oltre ad essere collegati ai vari host. Nelle righe 37 e 38 vengono creati i collegamenti tra gli Switch ed il Router. All'interno del metodo run() viene creata l'istanza della classe Mininet alla quale è passata la topologia personalizzata definita all'interno della funzione build(). Alla riga 46 viene fatta acquisire al router r0 l'interfaccia hardware enp0s3, in seguito, nella riga 47 si invia il comando "dhclient enp0s3" per richiedere una configurazione DHCP. Nelle righe 48 e 49 vengono configurate le interfacce r0-eth1 e r0-eth2 collegate rispettivamente agli Switch s1 ed s2 definendo l'indirizzo IP e Netmask coerenti con le due reti. Infine, alla riga 50 viene aggiunta la regola di Natting all'interfaccia enp0s3.

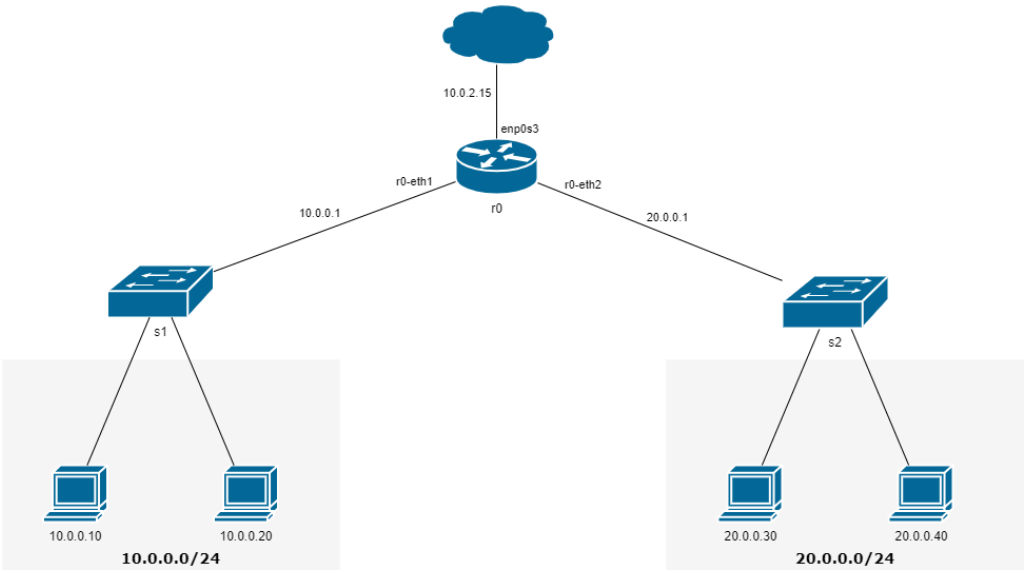

Figura 3.4: Topologia della rete Mininet

3.3 Installazione e panoramica dell'applicativo sFlow-RT

Per quanto riguarda la seconda macchina virtuale "VM_Monitoring" è stata configurata una scheda di rete connessa su rete interna "Internal_Monitoring" utilizzata per la comunicazione tra gli agenti sFlow che compongono la rete virtuale creata con Mininet ed eventuali applicazioni di monitoraggio.

In questo caso per il monitoraggio è stata utilizzato l'applicativo sFlow RT installabile con i seguenti comandi:

```
$ sudo apt install default-jre
$ wget https://inmon.com/products/sFlow-RT/sflow-rt_3.0-1696.deb
$ sudo dpkg -i sflow-rt_3.0-1696.deb
$ sudo systemctl enable sflow-rt
$ sudo systemctl start sflow-rt
```

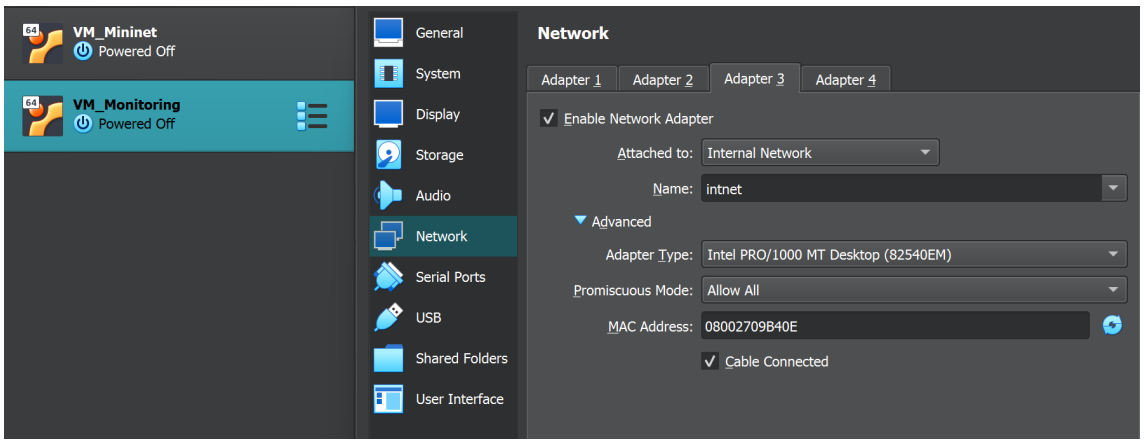

Figura 3.5: Scheda di rete interna della VM_Monitoring (enp0s9)

Gli agenti inviano periodicamente i dati di telemetria raccolti sulla porta 6343. L'interfaccia grafica di sFlow-RT può essere acceduta utilizzando un browser web e collegandosi sulla porta HTTP 8008, in questo caso per accedere all'interfaccia direttamente dalla macchina virtuale basterà collegarsi all'indirizzo 127.0.0.1:8008. La prima pagina che viene visualizzata è quella di Status in cui sono presenti diversi indicatori sullo stato di sFlow-RT e dei flussi ricevuti.

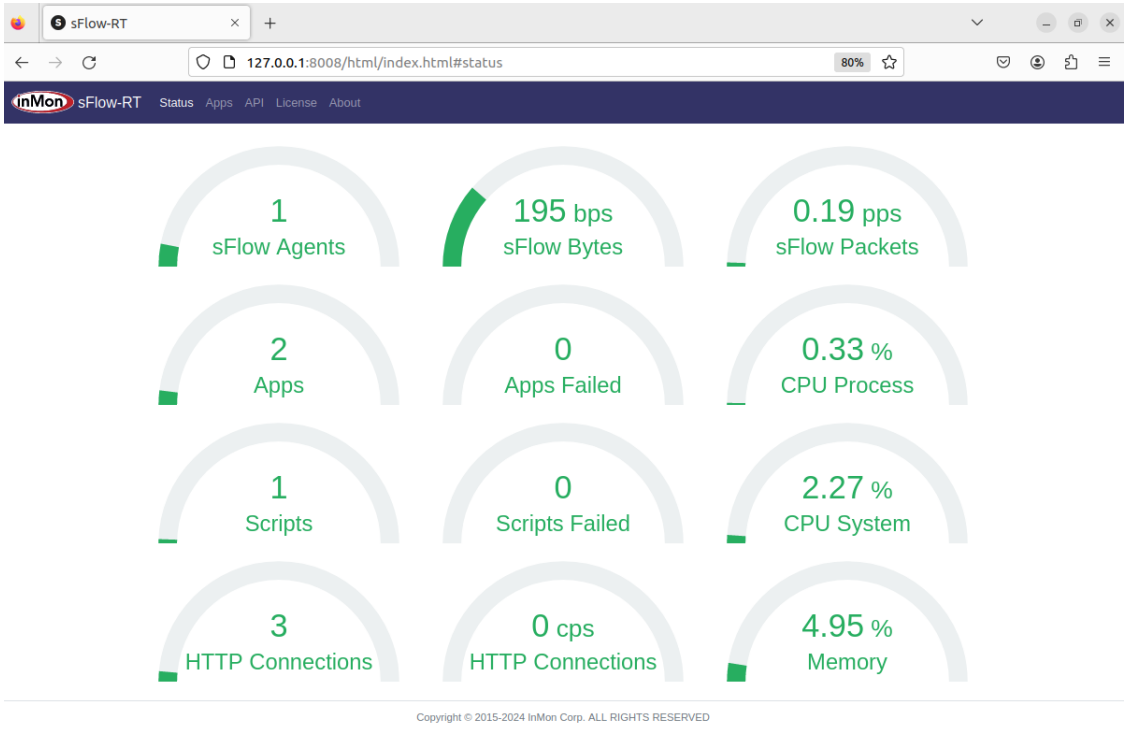

Figura 3.6: sezione Status di sFlow-RT

Nella sezione Apps si possono utilizzare le applicazioni di sFlow-RT, in questo caso sono state installate browse-flows e browse-metrics utilizzando i seguenti comandi:

```
$ sudo /usr/local/sflow-rt/get-app.sh sflow-rt browse-metrics
```

```
$ sudo /usr/local/sflow-rt/get-app.sh sflow-rt browse-flows
$ sudo /usr/local/sflow-rt/get-app.sh sflow-rt flow-trend
```

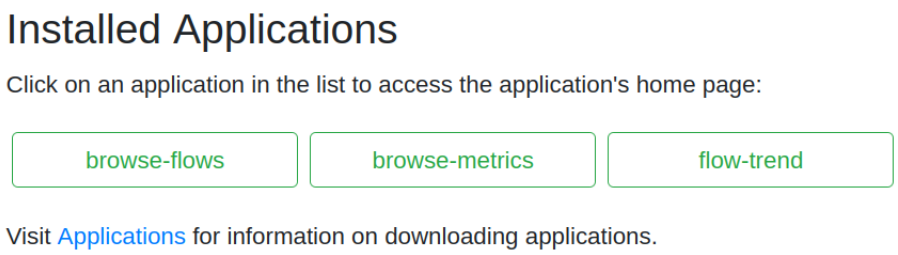

Figura 3.7: sezione Status di sFlow-RT

Inoltre è presente una sezione dedicata alle API in cui sono documentate le API REST messe a disposizione per creare applicazioni esterne basate sui servizi di sFlow-RT.

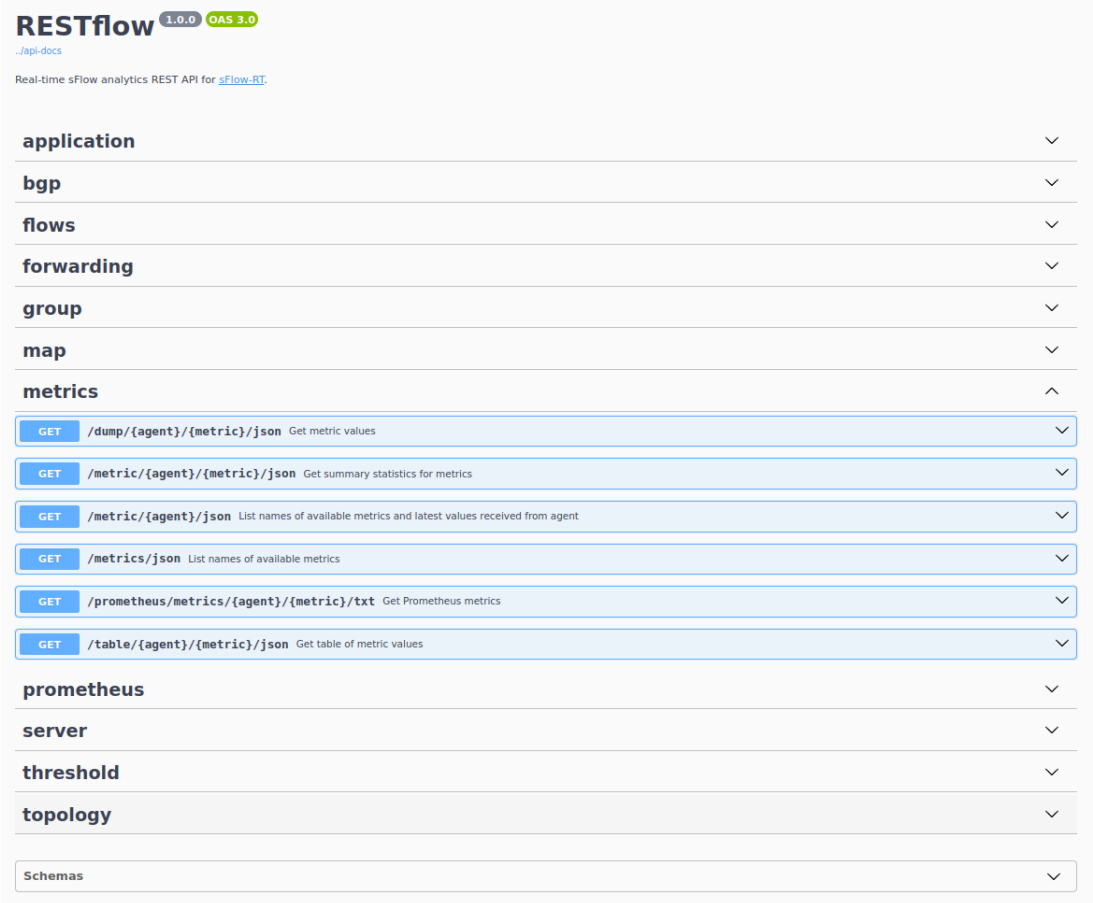

Figura 3.8: sezione Status di sFlow-RT

3.4 Monitoraggio della rete Mininet con l'utilizzo di sFlow

In questa fase si ha l'obiettivo di monitorare la rete Mininet utilizzando sFlow-RT per visualizzare eventuali andamenti e misurazioni. Il primo passo è quello di abilitare sFlow nei due Switch che compongono la rete in esecuzione nella prima macchina virtuale. Rispettivamente per s1 e s2 si eseguono i seguenti comandi:

```
$ sudo ovs-vsctl -- --id=@sflow create sflow agent=enp0s8
    target=\"10.13.13.102:6343\" sampling=10 polling=20 -- -- set bridge
    s1 sflow=@sflow


$ sudo ovs-vsctl -- --id=@sflow create sflow agent=enp0s9
    target=\"10.13.13.102:6343\" sampling=10 polling=20 -- -- set bridge
    s2 sflow=@sflow
```

I parametri hanno i seguenti significati: **agent** rappresenta l'interfaccia dello Switch collegata alla rete interna di monitoraggio, il parametro **target** si va invece ad indicare l'indirizzo IP del terminale di monitoraggio e la porta in cui esso è in ascolto, il parametro **sampling** definisce

- **agent**: interfaccia dello Switch collegata alla rete interna di monitoraggio
- **target**: l'indirizzo IP del terminale di monitoraggio e la porta in cui esso è in ascolto
- **sampling**: rate di campionamento
- **polling**: intervallo di campionamento in secondi

Per quanto riguarda la macchina virtuale di monitoraggio si avvia l'applicativo sFlow-RT mediante il seguente comando:

```
$ ./sflow-rt/start.sh
```

Per simulare diverse casistiche di monitoraggio sono stati eseguiti alcuni esperimenti andando a modificare i parametri di campionamento e i metodi di visualizzazione. Inizialmente sono state impostate frequenze di sampling con valori uguali e frequenze di polling con valori significativamente diversi, come è possibile notare nel seguente screenshot. In particolare l'agente enp0s8 rappresenta l'interfaccia dello Switch s1 ed ha indirizzo ip 10.13.13.104, mentre l'agente enp0s9 rappresenta l'interfaccia dello Switch s2 ed ha indirizzo ip 10.13.13.101.

```
studente@Ubuntu22Mininet:~$ sudo ovs-vsctl list sflow
_uuid               : 8cb76cd1-c2f5-4896-af6c-e39fd5e12aa4
agent               : enp0s8
external_ids        : {}
header              : []
polling             : 3
sampling            : 3
targets             : ["10.13.13.102:6343"]

_uuid               : 162e877b-4e28-4a21-a186-9e304b6798f7
agent               : enp0s9
external_ids        : {}
header              : []
polling             : 10
sampling            : 3
targets             : ["10.13.13.102:6343"]
studente@Ubuntu22Mininet:~$
```

Figura 3.9: Frequenze di sampling e polling dei due agent

Per quantificare le differenze dovute alle diverse configurazioni sono stati eseguiti due ping paralleli diretti verso una macchina esterna presente in internet. I dati di monitoraggio sono stati raggruppati per origine e destinazione degli indirizzi IP e per agente. Il primo grafico rappresenta i bit al secondo inviati durante l'esecuzione dei comandi ping, mentre il secondo grafico rappresenta i frame al secondo. Dai due grafici non è possibile dedurre differenze particolari nel traffico trasmesso o nelle tecniche di monitoraggio, d'altra parte, i dati di misura rispecchiano l'andamento real-time dei flussi e possono essere richiesti tramite API REST da applicativi come Grafana per effettuare, ad esempio, eventuali operazioni di detection.

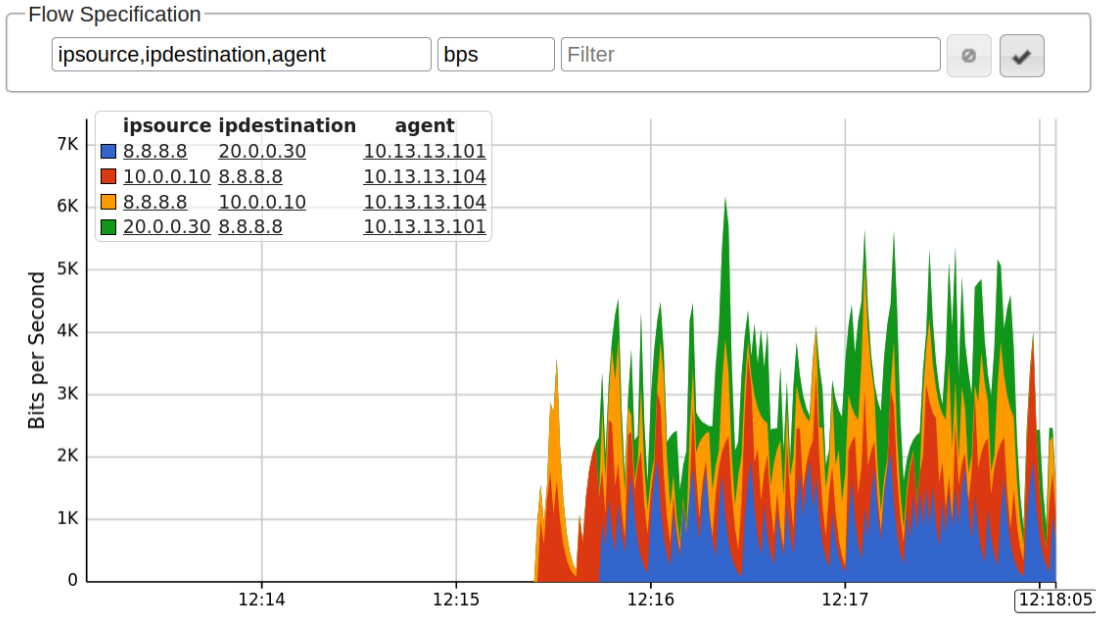

Figura 3.10: Plot dei Bit/s per i ping da H1 a 8.8.8.8 e da H3 a 8.8.8.8

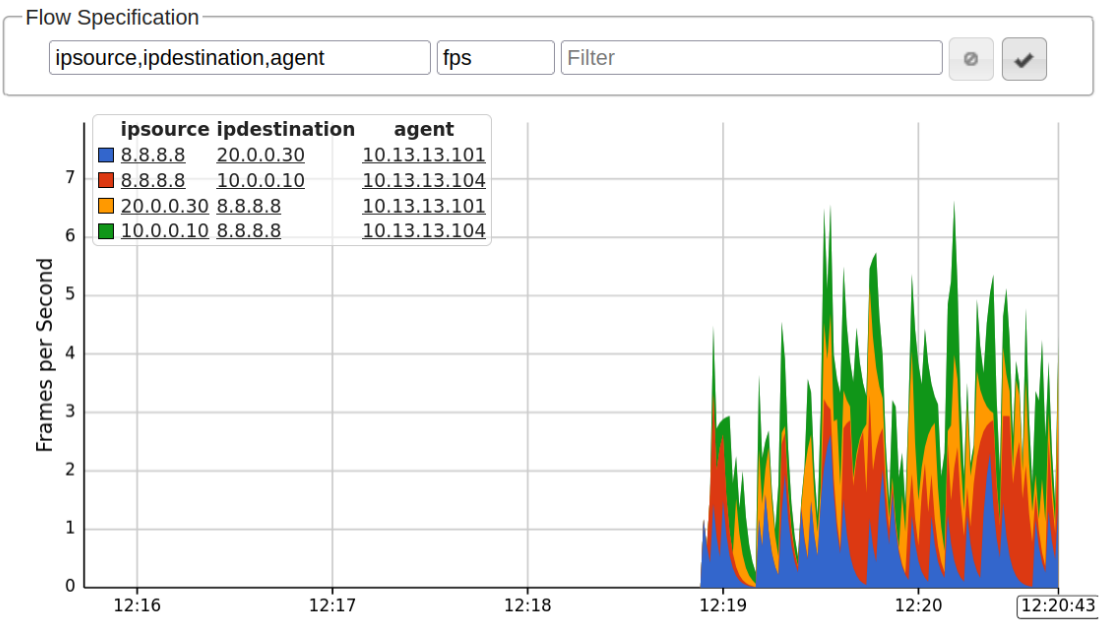

Figura 3.11: Plot dei Frame/s per i ping da H1 a 8.8.8.8 e da H3 a 8.8.8.8

Per esacerbare possibili differenze nelle tecniche di monitoraggio sono state modificate le frequenze di sampling e polling in modo drastico. Come si può vedere nella seguente immagine, l'agente enp0s8 risulta invariato, mentre l'agente enp0s9 viene configurato con rate di sampling e polling pari a 20. Il grafico rappresenta

```
_uuid               : 8cb76cd1-c2f5-4896-af6c-e39fd5e12aa4
agent               : enp0s8
external_ids        : {}
header              : []
polling             : 3
sampling            : 3
targets             : ["10.13.13.102:6343"]

_uuid               : 162e877b-4e28-4a21-a186-9e304b6798f7
agent               : enp0s9
external_ids        : {}
header              : []
polling             : 20
sampling            : 20
targets             : ["10.13.13.102:6343"]
```

Figura 3.12: Frequenze di sampling e polling dei due agent

correttamente le differenze, mostrando, a fronte di una elevata sampling rate, un maggiore numero di frame nell'unità di tempo. La differenza è resa evidente dalla presenza di picchi molto più alti rispetto all'interfaccia dello switch con sampling=3. D'altra parte una sample rate e una finestra di campionamento minori mostrano un andamento più lineare.

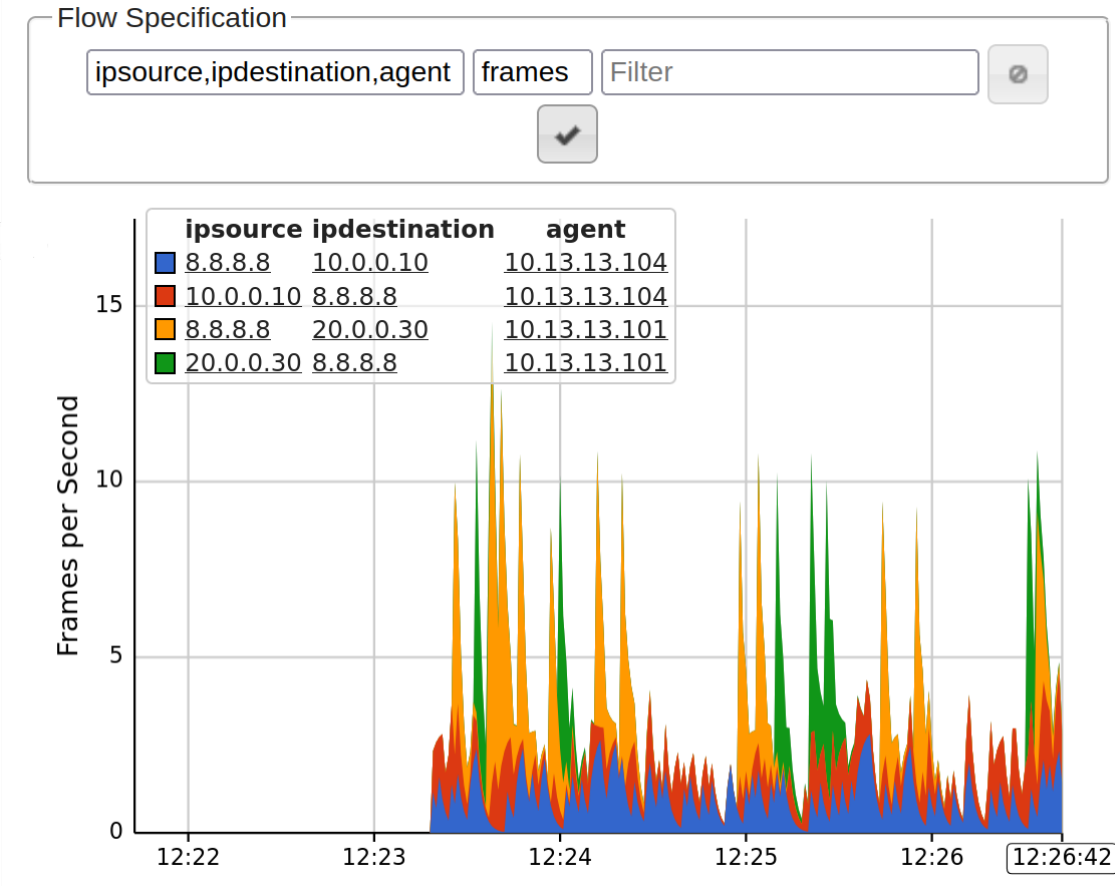

Figura 3.13: Plot dei Frame/s per i ping ad 8.8.8.8 con rate di sampling e polling differenti

In questo caso si va ad eseguire il comando iperf mantenendo le rate di sampling e di polling del caso precedente. iPerf è uno strumento che misura la larghezza di banda e valuta le prestazioni di una connessione di rete tramite test di trasferimento di dati tra due host, fornendo informazioni sulla velocità e la qualità della connessione. Questo comando supporta diversi protocolli, in questo caso il test è realizzato rispetto all'utilizzo del protocollo TCP.

```
mininet> h4 iperf -s &
mininet> h1 iperf -t 10000000000000000 -c h4
------------------------------------------------------------
Client connecting to 20.0.0.40, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.10 port 38020 connected with 20.0.0.40 port 5001
```

Figura 3.14: Impostazioni del comando iperf

In questo caso vediamo due flussi con andamenti simili dati dal funzionamento di iperf dove il client satura la rete e il server riceve i pacchetti effettuando eventuali misurazioni. Si nota una differenza di volume del traffico data dal fatto che il server riceve solo un sottoinsieme dei pacchetti che vengono inviati dal client.

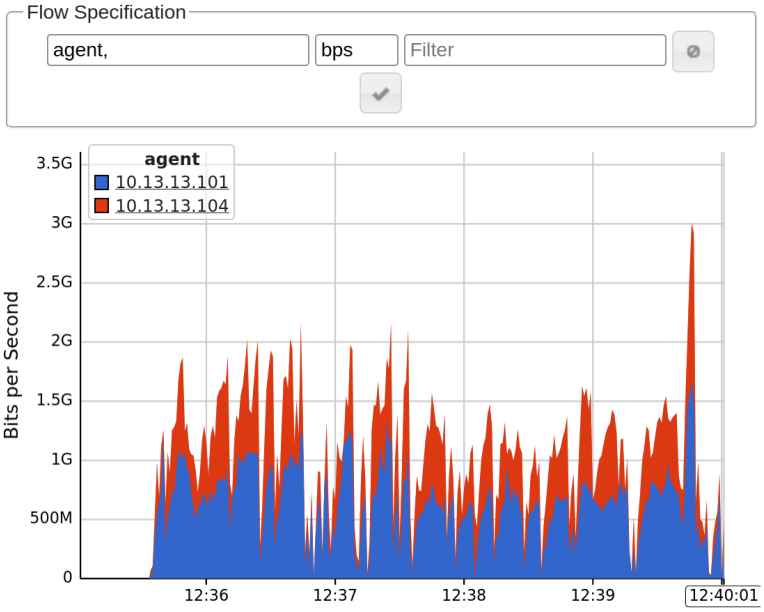

Figura 3.15: Plot dei Bit/s per la trasmissione Iperf

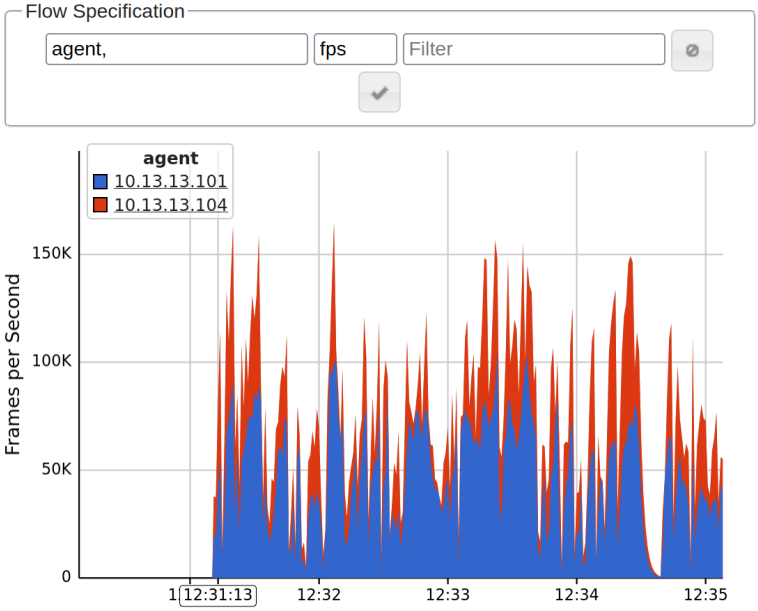

Figura 3.16: Plot dei Frame/s per la trasmissione Iperf