UNIVERSITÀ DEGLI STUDI
DI PERUGIA

Tesina di

INTELLIGENT AND SECURE NETWORKS

Corso di Laurea in Ingegneria Informatica e Robotica A.A. 2024/2025

Dipartimento di Ingegneria

docente

Prof. Mauro Femminella

Misurazioni delle prestazioni di una rete tramite perfSONAR

studenti

383464	**Giulia**	**Colasanti**	giulia.colasanti1@studenti.unipg.it
386161	**Armando**	**Palermo**	armando.palermo@studenti.unipg.it

0. Indice

1. Introduzione a perfSONAR

PerfSONAR è una suite di strumenti open-source che monitora le prestazioni di rete, individuando problemi di latenza, perdita di pacchetti e throughput, garantendo così il corretto funzionamento delle applicazioni che dipendono dalla rete, come videoconferenze, cloud computing e trasferimenti di dati.

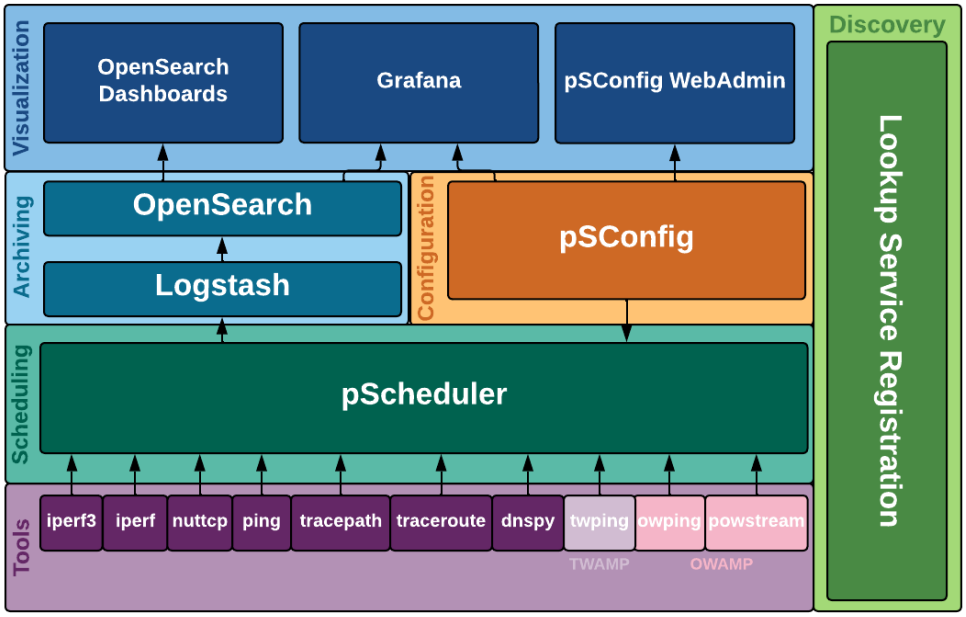

Figura 1.1: Struttura di perfSONAR

La sua struttura è divisa principalemnte in 6 blocchi :

- **Tools**: perfSONAR include numerosi strumenti per misurare la rete, come owamp, twamp, iperf, ping e molti altri. Tuttavia, anziché utilizzare direttamente questi strumenti, si utilizza il comando `pscheduler` dallo strato di pianificazione per eseguirli in modo più semplice e centralizzato.

- **Scheduling**: la pianificazione è gestita da un unico strumento, *pScheduler*, che si occupa di individuare le finestre temporali ottimali per eseguire gli strumenti di misurazione, evitando conflitti che potrebbero compromettere i risultati. Una volta individuate le finestre, pScheduler esegue i tools e raccoglie i dati ottenuti e se necessario li invia all'archivio per la conservazione.

- **Archiving**: memorizza i dati delle misurazioni. Utilizza una combinazione di Logstash e OpenSearch, dei componenti open source che aggiungono metadati alle misurazioni e permettono interrogazioni flessibili.

- **Configuration**: definisce le misurazioni desiderate e la loro destinazione di archiviazione. *pSConfig*, il componente principale, è un framework basato su template che descrive e configura le attività. pSConfig gestisce più host semplificando il processo grazie all'uso di agent che automatizzano le operazioni di configurazione e la visualizzazione dei risultati.
- **Visualization**: include strumenti che ti permettono di visualizzare i dati della rete e individuare facilmente i problemi. Con *Grafana*, si possono creare dashboard per monitorare le misurazioni della rete nel tempo, mentre *OpenSearch Dashboards* consente di visualizzare i dati e accedere alle impostazioni di OpenSearch tramite un'interfaccia intuitiva.
- **Discovery**: il *Lookup Service Registration Daemon* di perfSONAR registra automaticamente ogni nodo in un elenco pubblico o privato, raccogliendo informazioni utili per risolvere problemi e configurare test di rete. Di solito, non serve configurarlo, ma esiste una guida per le impostazioni avanzate.

Nell'ambito del seguente progetto, l'uso di perfSONAR ha richiesto alcune modifiche e adattamenti rispetto alla configurazione standard che comprende tutta la suite. In particolare questo è dovuto alla non compatibilità di perfSONAR con Ubuntu 24. Alcuni strumenti e servizi sono stati esclusi o sostituiti con alternative più adatte all'ambiente di lavoro e alle esigenze operative. Queste modifiche sono state effettuate cercando di mantenere l'affidabilità e l'efficacia delle misurazioni di rete, assicurando che perfSONAR fosse integrato nel sistema nel miglior modo possibile.

2. Topologia rete

Le macchine della rete analizzata sono state create in VirtualBox. Per simulare l'interazione tra due reti diverse, sono state definite tre macchine: un client e un server, che fungono da testpoint, e un router che le collega. Il server, oltre a svolgere il ruolo di testpoint, archivia i dati raccolti dalle misurazioni in un database MySQL e, grazie a Grafana, ne consente la visualizzazione e l'analisi attraverso grafici intuitivi.

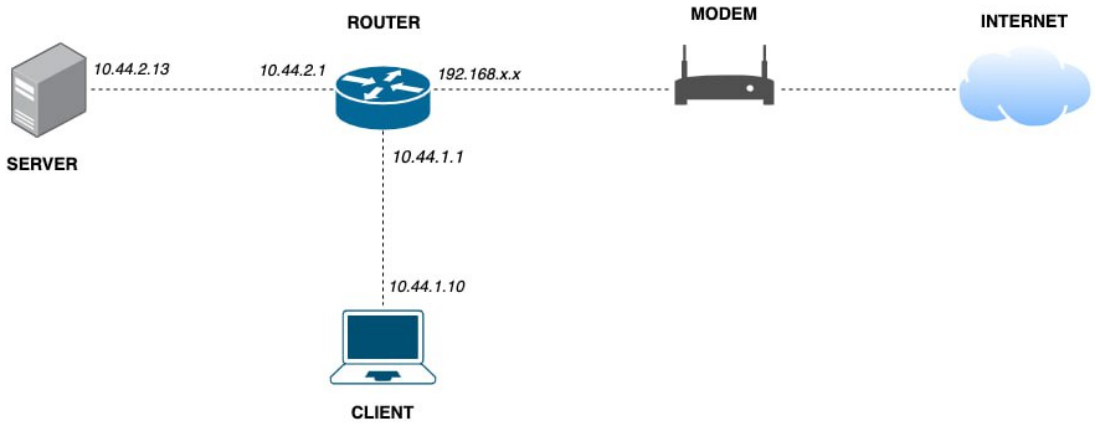

Figura 2.1: Topologia rete

2.1 Configurazione router

La topologia mostrata in figura 2.1 illustra la struttura della rete. Il router dispone di tre interfacce: `enp0s3`, configurata in modalità scheda con bridge e connessa alla rete esterna ottiene un indirizzo IP assegnato da quest'ultima, mentre `enp0s8` ed `enp0s9` fungono da gateway per le due sottoreti create, utilizzando rispettivamente gli indirizzi IP `10.44.1.1` e `10.44.2.1`.
Dopo aver avviato la macchina virtuale, è necessario configurare le interfacce di rete. L'interfaccia bridged deve ricevere automaticamente la configurazione via DHCP, mentre le interfacce interne vanno impostate con un indirizzo statico. Questo si realizza modificando il file `YAML` nella directory `/etc/netplan/`, con la seguente configurazione:

```
network:
  ethernets:
    enp0s3:
      dhcp4: true
    enp0s8:
      addresses:
        - 10.44.1.1/24
    enp0s9:
      addresses:
```

```
        - 10.44.2.1/24
  version: 2
```

Successivamente, bisogna abilitare l'inoltro dei pacchetti tra le interfacce e, per attivare il NAT, configurare il mascheramento delle porte. Per automatizzare questa operazione, si può creare uno script `Bash` da eseguire dopo ogni avvio.

```
#!/bin/bash
sudo sysctl net.ipv4.ip_forward=1
sudo iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE
```

Per il server, la gestione degli indirizzi IP è definita tramite una configurazione statica, che sarà analizzata successivamente. Per i client all'interno delle reti locali, la gestione è affidata a un server DHCP configurato sull'interfaccia interna del router. Il pacchetto richiesto è `isc-dhcp-server`, che può essere installato con il comando:

```
sudo apt install isc-dhcp-server
```

Dopodiché, è necessario modificare due file di configurazione. Il primo è

```
/etc/default/isc-dhcp-server
```

in cui si specifica l'interfaccia di rete che gestirà il servizio DHCP:

```
INTERFACESv4="enp0s8 enp0s9"
```

Il secondo è

```
/etc/dhcp/dhcpd.conf
```

in cui si definiscono i parametri per l'assegnazione dinamica degli indirizzi IP dei client:

```
option domain-name-servers 8.8.8.8;
ignore-client-uids true;
one-lease-per-client on;

subnet 10.44.1.0 netmask 255.255.255.0 {
  range 10.44.1.10 10.44.1.20;
  option routers 10.44.1.1;
}
subnet 10.44.2.0 netmask 255.255.255.0 {
  range 10.44.2.20 10.44.2.30;
  option routers 10.44.2.1;
}
```

2.2 Configurazione di rete per server e client

Il client e il server dispongono ciascuno di una sola interfaccia di rete, collegata a una delle due reti interne gestite dal router:

```
indirizzo IP client : 10.44.1.10
indirizzo IP server: 10.44.2.13
```

Come accennato nella sezione precedente, per garantire che l'indirizzo IP del server rimanga invariato, viene configurato staticamente modificando il file `/etc/netplan` come segue:

```
network:
  version: 2
  ethernets:
```

```
    enp0s3:
      dhcp4: no
      addresses:
        - 10.44.2.13/24
      routes:
        - to: default
          via: 10.44.2.1
      nameservers:
        addresses:
          - 8.8.8.8
          - 8.8.4.4
```

Per il client l'indirizzo IP viene affidato tramite DHCP:

```
network:
  ethernets:
    enp0s3:
      dhcp4: yes
   version: 2
```

3. Set-up ambiente

Il client e il server fungono da testpoint per perfSONAR, che è stato installato all'interno di container Docker, mentre Grafana e MySQL sono stati installati direttamente sul server al di fuori dell'ambiente Docker.

3.1 Installazione perfSONAR tramite Docker

Per semplificare l'installazione e la gestione di perfSONAR, è stato utilizzato Docker, una piattaforma che consente di eseguire applicazioni all'interno di container, ovvero ambienti isolati e leggeri che includono tutto il necessario per il funzionamento del software, senza interferire con il sistema operativo host. Il comando installazione docker:

```
sudo apt install docker.io
```

L'installazione di perfSONAR è stata effettuata facendo riferimento alla documentazione ufficiale.

```
sudo docker pull perfsonar/testpoint:systemd
```

Questo comando recupera dal repository Docker l'immagine di perfSONAR Testpoint, contenente tutto il necessario per eseguire il software all'interno di un container.

```
docker run -td --name perfsonar-testpoint --net=host --tmpfs /run --tmpfs
   /run/lock --tmpfs /tmp -v /sys/fs/cgroup:/sys/fs/cgroup:rw --cgroupns
   host perfsonar/testpoint:systemd
```

Il comando precedente esegue il container in modalità detached (in background), assegnandogli il nome perfsonar-testpoint. L'opzione `--net=host` permette al container di utilizzare direttamente la rete del sistema host, garantendo prestazioni migliori nei test di rete. Inoltre, vengono montati alcuni filesystem temporanei e abilitato l'accesso ai cgroup per garantire la compatibilità con systemd.

```
sudo docker start perfsonar-testpoint
```

Questo comando avvia il container perfsonar-testpoint senza bisogno di eseguirlo nuovamente da zero. Infine, per accedere alla shell del container ed eseguire comandi direttamente all'interno di perfSONAR, si utilizza:

```
docker exec -it perfsonar-testpoint bash
```

3.2 Installazione MySQL su server

Per garantire la raccolta e l'archiviazione dei dati di misurazione generati da perfSONAR, il server è stato configurato con un database MySQL. L'installazione di MySQL Server è stata effettuata con il comando:

```
sudo apt install mysql-server
```

Di default, MySQL comunica sulla porta 3306, consentendo ai client di connettersi e interrogare il database.Tuttavia, per motivi di sicurezza, l'accesso al database è stato limitato esclusivamente al server stesso, impedendo connessioni da altri nodi della rete.Per la gestione dei dati, è stato creato un utente dedicato con i seguenti comandi:

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY
  'monitorIns123456';
GRANT ALL PRIVILEGES ON *.* TO 'admin'@'localhost' WITH GRANT OPTION;
FLUSH PRIVILEGES;
```

3.2.1 Creazione Tabelle

Ogni misurazione viene inserita all'interno di una tabella a seconda del tipo di misurazione effettuata. Sulla base dei tipi di test configurati e sull'analisi dei risultati forniti da essi sono state generate le seguenti tabelle:

```
CREATE TABLE IF NOT EXISTS measurements_throughput (
    id INT AUTO_INCREMENT PRIMARY KEY,
    added DATETIME,
    state VARCHAR(50),
    end_time DATETIME,
    start_time DATETIME
);

CREATE TABLE IF NOT EXISTS intervals_throughput (
    id INT AUTO_INCREMENT PRIMARY KEY,
    measurement_id INT,
    end_time FLOAT,
    rtt INT,
    start_time FLOAT,
    stream_id INT,
    retransmits INT,
    tcp_window_size INT,
    throughput_bits FLOAT,
    throughput_bytes INT,
    FOREIGN KEY (measurement_id) REFERENCES
                measurements_throughput(id)
);
```

La prima tabella archivia informazioni generali sui test di throughput, registrando l'ID della misurazione, gli orari di inizio e fine, lo stato del test e la data di aggiunta nel database. La seconda tabella memorizza i dettagli relativi agli intervalli di throughput, includendo informazioni come RTT (Round Trip Time), stream ID, retransmits, TCP window size e i valori di throughput in bit e byte. Ogni riga è collegata a una misurazione specifica tramite una chiave esterna che fa riferimento alla tabella più generale `measurements_throughput`.

Per gestire i dati relativi ai test di Round Trip Time (RTT), sono state create due tabelle

```
CREATE TABLE rtt_tests (
    id VARCHAR(36) PRIMARY KEY,
    added DATETIME,
    source_ip VARCHAR(15),
    dest_ip VARCHAR(15),
    packet_count INT,
    packet_size INT,
    intervallo VARCHAR(10),
```

```
    result_max FLOAT,
    result_min FLOAT,
    result_mean FLOAT,
    result_stddev FLOAT,
    packets_sent INT,
    packets_received INT,
    packets_lost INT,
    state VARCHAR(20),
    start_time DATETIME,
    end_time DATETIME
);

CREATE TABLE rtt_roundtrips (
    id INT AUTO_INCREMENT PRIMARY KEY,
    test_id VARCHAR(36),
    ip VARCHAR(15),
    rtt FLOAT,
    seq INT,
    ttl INT,
    packet_length INT,
    FOREIGN KEY (test_id) REFERENCES rtt_tests(id)
                ON DELETE CASCADE
);
```

La prima tabella memorizza le informazioni principali di ogni test RTT eseguito, includendo dettagli sulla sorgente e destinazione del test, il numero e la dimensione dei pacchetti inviati, oltre ai risultati statistici delle misurazioni come il valore massimo, minimo, medio e la deviazione standard dell'RTT.

La seconda tabella contiene i dettagli di ciascun pacchetto inviato durante un test RTT, registrando il tempo di andata e ritorno per ogni pacchetto ricevuto, insieme a informazioni sul TTL e sulla dimensione del pacchetto.

Per gestire i dati relativi alla misurazioni di latenza, sono state create due tabelle

```
CREATE TABLE latency_measurements (
    id VARCHAR(36) PRIMARY KEY,
    start_time DATETIME,
    end_time DATETIME,
    source_ip VARCHAR(15),
    destination_ip VARCHAR(15),
    packets_sent INT,
    packets_received INT,
    packets_lost INT,
    packets_reordered INT,
    packets_duplicated INT,
    max_clock_error FLOAT
);

CREATE TABLE latency_histogram (
    measurement_id VARCHAR(36),
    latency FLOAT,
    count INT,
    FOREIGN KEY (measurement_id) REFERENCES
                latency_measurements(id)
);
```

La prima tabella archivia le informazioni generali sui test di latenza, includendo dettagli sull'indirizzo IP di origine e destinazione, il numero di pacchetti inviati, ricevuti, persi, riordinati e duplicati, oltre a

misurazioni di errore di orologio tra i due host (incertezza temporale). La seconda tabella memorizza la distribuzione della latenza in un formato istogramma. Ogni riga nel database rappresenta una determinata latenza e la frequenza di tale latenza nel corso del test.

3.3 Installazione Grafana su server

Grafana è una piattaforma open-source per la visualizzazione e l'analisi di dati in tempo reale, che permette la creazione di dashboard dinamiche e personalizzate contenenti ad esempio grafici e tabelle. Inoltre è possibile impostare notifiche basate su soglie predefinite per ricevere avvisi in caso di problemi. Nel contesto di questo progetto, Grafana è stata utilizzata per monitorare e rappresentare i dati archiviati nel database MySQL, raccolti dai testpoint di perfSONAR. L'installazione di Grafana su server con sistema operativo Debian/Ubuntu prevede i seguenti passaggi:

- Installazione dei pacchetti necessari:

```
apt-get install -y apt-transport-https software-properties-common
   wget
```

- Importazione chiave GPG:

```
sudo mkdir -p /etc/apt/keyrings/wget -q -O - https://apt.grafana.com/
   gpg.key | gpg --dearmor | sudo tee /etc/apt/keyrings/grafana.gpg >
   /dev/null
```

- Aggiunta repository per la stable release di Grafana:

```
echo "deb␣[signed-by=/etc/apt/keyrings/grafana.gpg]https://apt.
   grafana.com␣stable␣main" | sudo tee -/etc/apt/sources.list.d/
   grafana.list
```

- Aggiornamento e installazione Grafana:

```
sudo apt-get update
sudo apt-get install grafana
```

- Avviare e abilitare il servizio Grafana:

```
sudo systemctl start grafana-server
sudo systemctl enable grafana-server
```

Una volta installato e avviato, Grafana è accessibile tramite un browser all'indirizzo: `http://10.44.2.13:3000`.

4. Flusso dei dati

In questa sezione viene descritto il flusso dei dati, illustrando i processi e le tecnologie utilizzate per la raccolta, l'archiviazione e l'analisi delle misurazioni di rete. L'obiettivo è presentare un sistema scalabile ed efficace per la gestione delle informazioni raccolte.

4.1 Raccolta e archiviazione automatica delle misurazioni

Nel contesto di questo progetto, è stato necessario raccogliere automaticamente i dati di misurazione della rete dai testpoint di perfSONAR e archiviarli in un database MySQL. A tale scopo è stato utilizzato pSConfig, uno strumento che consente di definire e distribuire configurazioni per l'esecuzione automatizzata di test di rete. Attraverso un file di configurazione JSON, pSConfig stabilisce quali test devono essere eseguiti, con quale frequenza e quali dati raccogliere.
Di seguito il file di configurazione che si trova sul client:

```
{
  "addresses": {
    "server": {
      "address": "10.44.2.13"
    },
    "client": {
      "address": "10.44.1.10"
    }
  },
  "groups": {
    "throughput_group": {
      "type": "mesh",
      "addresses": [
        {
          "name": "server"
        },
        {
          "name": "client"
        }
      ]
    }
  },
  "tests": {
    "throughput_test": {
      "type": "throughput",
      "spec": {
        "source": "{% address[0] %}",
        "dest": "{% address[1] %}",
        "duration": "PT30S"
      }
    },
    "latency_test": {
      "type": "latencybg",
      "spec": {
        "source": "{% address[0] %}",
        "dest": "{% address[1] %}",
        "packet-interval": 0.1,
        "packet-count": 600
      }
    },
    "rtt_test": {
      "type": "rtt",
      "spec": {
        "source": "{% address[0] %}",
        "dest": "{% address[1] %}",
        "count": 10,
        "interval": "PT1S",
        "length": 1000,
        "ip-version": 4
      }
    }
  },
  "archives": {
    "throughput_archive": {
      "archiver": "http",
      "data": {
        "_url": "http://10.44.2.13:5000/
            ThroughputTests",
        "op": "post"
      },
      "transform": {
        "script": "if (.test.type == \"throughput
            \") then . else null end"
      },
      "ttl": "PT5M"
    },
    "latency_archive": {
      "archiver": "http",
      "data": {
        "_url": "http://10.44.2.13:5000/
            LatencyTests",
        "op": "post"
      },
      "transform": {
        "script": "if (.test.type == \"latencybg
            \") then . else null end"
      },
      "ttl": "PT5M"
    },
    "rtt_archive": {
      "archiver": "http",
```

```
      "data": {
        "_url": "http://10.44.2.13:5000/rttTests
            ",
        "op": "post"
      },
      "transform": {
        "script": "if (.test.type == \"rtt\")
            then . else null end"
      },
      "ttl": "PT5M"
    }
  },
  "schedules": {
    "every_2_minutes": {
      "repeat": "PT2M",
      "slip": "PT30S",
      "sliprand": true
    },
    "every_3_minutes": {
      "repeat": "PT3M",
      "slip": "PT10S",
      "sliprand": true
    }
  },
  "tasks": {
    "throughput_task": {
      "group": "throughput_group",
      "test": "throughput_test",
      "schedule": "every_2_minutes",
      "archives": [
        "throughput_archive"
      ]
    },
    "latency_task": {
      "group": "throughput_group",
      "test": "latency_test",
      "archives": [
        "latency_archive"
      ]
    },
    "rtt_task": {
      "group": "throughput_group",
      "test": "rtt_test",
      "schedule": "every_3_minutes",
      "archives": [
        "rtt_archive"
      ]
    }
  }
```

Il file di configurazione JSON è strutturato in modo tale da gestire l'esecuzione di test di rete. Ogni sezione ha un ruolo specifico nel definire, organizzare e archiviare i test.

La prima sezione, **addresses**, stabilisce gli indirizzi IP dei nodi coinvolti nei test. Qui vengono definiti i dispositivi che prenderanno parte alle operazioni, come ad esempio un server e un client.

Successivamente, la sezione **groups** organizza questi nodi in configurazioni specifiche per i test. Un esempio è il gruppo "`throughput_group`", che utilizza una topologia di tipo "`mesh`" per collegare il server e il client, consentendo così l'esecuzione delle misurazioni richieste.

Nella sezione **test** vengono descritti i test di rete che verranno effettuati. Ogni test ha un tipo specifico, come "`throughput`", "`latencybg`" o "`rtt`", e dispone di parametri configurabili per adattarlo alle esigenze. Inoltre, ogni test specifica una sorgente ("`source`") e una destinazione (`"dest"`), ossia gli indirizzi dei nodi coinvolti nella misurazione.

Per quanto riguarda la conservazione dei risultati, la sezione **archives** definisce dove e come verranno memorizzati i dati generati dai test. Le informazioni vengono inviate a specifici URL attraverso richieste HTTP con metodo "POST". Inoltre, ogni archivio utilizza uno script di trasformazione per filtrare i dati rilevanti e dispone di un parametro "`ttl`" (Time To Live), che determina per quanto tempo le informazioni saranno conservate. Ad esempio, nel caso dei test di tipo "`rtt`", solo i relativi risultati verranno salvati, mentre gli altri tipi di test saranno ignorati in questa fase.

Un altro aspetto fondamentale è la **schedules**, che stabilisce la frequenza di esecuzione dei test. Sono definiti intervalli regolari, come "`every_2_minutes`" (ogni 2 minuti) e "`every_3_minutes`" (ogni 3 minuti), con la possibilità di aggiungere un ritardo variabile per evitare sovraccarichi, indicato dal parametro "`slip`".

Infine, la sezione **tasks** determina come i test vengono effettivamente eseguiti. Ogni task associa un gruppo a un test specifico, eseguendolo in base alla pianificazione definita. Una volta completati, i risultati vengono archiviati secondo le impostazioni stabilite nelle sezioni precedenti.

La gestione di questa configurazione è affidata a **psconfig-pscheduler-agent** un software che interpreta il file di configurazione JSON e coordina l'esecuzione dei test di rete. Questo agente assicura che le attività vengano eseguite nei tempi previsti, raccoglie i risultati e li archivia nei rispettivi repository, semplificando la gestione e l'automazione del processo. Dopo aver letto i file di configurazione, il componente **pscheduler-scheduler** di perfSONAR si occupa di programmare ed eseguire i test secondo le specifiche definite nei file di configurazione.

4.2 API Flask e script Python per il caricamento nel db

Come accennato nella sezione precedente i dati delle misurazioni vengono inoltrati in un archivio remoto tramite HTTP attraverso una richiesta POST. Per poter gestire tali richieste si è fatto uso del framework Flask. Flask è un framework leggero scritto in Python che permette di creare applicazioni web e API RESTful in modo semplice ed efficace. Di seguito viene riportato lo script in Python che utilizza il framework Flask per la gestione delle richieste.

```
from flask import request,jsonify
import json
import pythonScriptThroughput as throughputHandler
import pythonScriptLatency as latencyHandler
import pythonScriptRtt as rttHandler
app = Flask(name)

@app.post("/ThroughputTests")
def handle_ThroughputTests():
 data = request.json
 if not data:
  return jsonify({"error":"Invalid␣JSON"}), 400
 throughputHandler.loadTestToDB(request.json)
 return jsonify({"result":"Funzione␣Eseguita"}),200


@app.post("/LatencyTests")
def handle_LatencyTests():
 data = request.json
 if not data:
  return jsonify({"error":"Invalid␣JSON"}), 400
 latencyHandler.loadTestToDB(request.json)
 return jsonify({"result":"Funzione␣Eseguita"}),200


@app.post("/rttTests")
def handle_rttTests():
 data = request.json
 if not data:
  return jsonify({"error":"Invalid␣JSON"}), 400
 rttHandler.loadTestToDB(request.json)
 return jsonify({"result":"Funzione␣Eseguita"}),200
```

Le librerie necessarie per l'applicazione Flask sono: Flask, che è la libreria principale per creare l'applicazione; request, che viene utilizzata per accedere ai dati delle richieste HTTP e jsonify, che serve per convertire le risposte in formato JSON. Vengono inoltre importati tre moduli personalizzati per gestire i test di throughput e latenza ed rtt. La riga `@app.post("/ThroughputTests")` definisce un endpoint per l'API.
Questo endpoint è accessibile tramite richieste HTTP di tipo POST all'URL `/ThroughputTests`. Tramite `@app.post`, quando arriva una richiesta di tipo POST a questo percorso, Flask invoca la funzione che segue, `handle_ThroughputTests()`. La prima operazione che la funzione esegue è il recupero dei dati inviati nella richiesta. Questi dati vengono estratti in formato JSON tramite `data = request.json`. Se i dati non ci sono, la funzione restituirà una risposta HTTP con il messaggio di errore con codice 400. Se invece i dati sono validi, il codice chiama la funzione `loadTestToDB(request.json)` del modulo `throughputHandler`. Questa funzione, analizzata in seguito, si occupa di prendere i dati ricevuti nella richiesta e caricarli in un database. Una volta che

i dati sono stati elaborati e memorizzati correttamente nel database, la funzione termina con una risposta HTTP positiva con codice 200. La struttura appena analizzata vale anche per gli altri tipi di test.

Il seguente script in Python ha lo scopo di caricare dati di test sulle prestazioni di rete nel database MySQL. In particolare, estrae informazioni da un file JSON strutturato e le inserisce in due tabelle: `measurements_throughput` e `intervals_throughput`.

```
import mysql.connector
import json
from os import listdir
from datetime import datetime

def loadTestToDB(json_data):
# Connessione al database MySQL
conn = mysql.connector.connect(
host="10.44.2.13",
user="admin",
password="monitorIns123456",
database="perfsonarDB"
)

cursor = conn.cursor()

# Trasformare il dizionario in una stringa JSON
#json_string = json.dumps(JsonFile, indent=4) # Usa indent=4 per una
    stringa formattata
#json_data = json.loads(JsonFile)

# Estrazione dei dati principali
print(json_data["run"]["added"])
added = datetime.fromisoformat(json_data["run"]["added"].replace("Z", "
    +00:00"))
state = json_data["run"]["state"]
end_time = json_data["run"]["end-time"]
start_time = json_data["run"]["start-time"]

# Inserimento dei dati nella tabella measurements
insert_measurement = """
INSERT␣INTO␣measurements_throughput(added,state,end_time,start_time)
VALUES␣(%s,%s,%s,%s)
"""
cursor.execute(insert_measurement, (added, state, end_time, start_time))
measurement_id = cursor.lastrowid

# Estrazione e inserimento dei dati degli intervalli
for interval in json_data["run"]["result-full"][0]["intervals"]:
for stream in interval["streams"]:
# Accedi ai dati di stream qui
tcp_window_size = stream["tcp-window-size"]
end_time = stream["end"]
rtt = stream["rtt"]
start_time = stream["start"]
stream_id = stream["stream-id"]
retransmits = stream["retransmits"]
throughput_bits = stream["throughput-bits"]
throughput_bytes = stream["throughput-bytes"]
```

```
insert_interval = """
INSERT INTO intervals_throughput(measurement_id,end_time,rtt,start_time,
    stream_id,retransmits,tcp_window_size,throughput_bits,throughput_bytes)
VALUES(%s,%s,%s,%s,%s,%s,%s,%s,%s)
"""
cursor.execute(insert_interval, (measurement_id, end_time, rtt, start_time
    , stream_id, retransmits, tcp_window_size, throughput_bits,
    throughput_bytes))

# Commit delle modifiche e chiusura della connessione
conn.commit()
cursor.close()
conn.close()
```

Lo script stabilisce una connessione con un database MySQL tramite la libreria mysql-connector utilizzando le credenziali specificate. Una volta connesso, estrae le informazioni principali da un file JSON ricevuto tramite una richiesta POST (analizzata precedentemente), ad esempio la data di aggiunta del test, il suo stato e gli orari di inizio e fine. Successivamente, i dati estratti vengono inseriti nella tabella principale del database, measurements_throughput, creando una nuova entry che registra i dettagli generali del test. L'ID della riga appena aggiunta viene utilizzata come chiave esterna per stabilire una relazione tra la misurazione generale (tabella measurament_throughput) e i dati più dettagliati sugli intervalli (tabella intervals_throughput). Per ogni intervallo vengono estratti parametri fondamentali come la finestra TCP, il tempo di inizio e fine, il round-trip time (RTT), l'ID dello stream, il numero di ritrasmissioni e i valori di throughput sia in bit che in byte. Questi dati vengono poi inseriti nella tabella intervals_throughput. Infine, dopo aver inserito tutte le informazioni necessarie, lo script esegue un commit per salvare le modifiche nel database e chiude la connessione, garantendo così la corretta memorizzazione dei dati.
La stessa struttura del precedente script viene utilizzata per le altre misurazioni.

5. Analisi dati

Nel seguente capitolo, vengono analizzati i risultati ottenuti dai test di rete eseguiti tra il client e il server utilizzando PerfSONAR. L'obiettivo principale di questi test è stato quello di raccogliere dati relativi ai principali parametri di performance della rete, ossia il throughput, il RTT (Round-Trip Time) e la latenza, per valutare le prestazioni complessive della connessione tra i due nodi.
I test sono stati effettuati in differenti condizioni di rete, includendo misurazioni a intervalli regolari per ciascun parametro monitorato.

Figura 5.1: Variazione Throughput_bytes ed tcp_window nel tempo

I grafici della figura 5.1 rappresentano il test di throughput per valutare la capacità di trasmissione dei dati tra il client e il server che funge anche da testpoint. L'asse delle ascisse indica il tempo di misurazione, mentre l'asse delle ordinate mostra i valori delle diverse metriche raccolte. Il primo grafico rappresenta la quantità di dati trasmessi in un determinato intervallo di tempo, mentre il secondo la dimensione della finestra TCP, che regola la quantità di dati che possono essere inviati prima di ricevere un'ACK (acknowledgment).

Durante le misurazioni, si osserva un andamento inizialmente stabile, seguito da un evidente rallentamento in un intervallo specifico (8:10 - 8:15). Questo rallentamento è stato introdotto arti-

ficialmente sul client tramite il tool `traffic control (tc)`. Per limitare la banda disponibile su un'interfaccia di rete, si utilizza il seguente comando:

```
tc qdisc add dev enp0s3 root tbf rate 500kbit burst 15kbit latency 600ms
```

Il parametro `rate 500 kbit` imposta la velocità massima di trasmissione dei dati, evitando che superi i 500 kbit al secondo. Il valore `burst 15 kbit` permette brevi picchi di trasmissione fino a `15 kbit` prima che il limite venga applicato, garantendo una maggiore fluidità. Infine, `latency 600 ms` definisce il tempo massimo che un pacchetto può restare in coda prima di essere trasmesso o eliminato, evitando congestioni nella rete.

Figura 5.2: Zoom dell'intervallo di rallentamento di throughput

Durante l'intervallo in cui la velocità dell'interfaccia del client è stata rallentata, il throughput è diminuito di circa un fattore 100 rispetto alle condizioni normali. Anche la dimensione della finestra TCP è diminuita, cercando di adattarsi al brusco cambiamento nella velocità di comunicazione. Una volta reimpostata la velocità dell'interfaccia, il throughput è tornato a comportarsi in modo simile a quanto osservato prima del rallentamento.

Un altro test effettuato riguarda la latenza, che rappresenta il tempo che intercorre tra l'invio di un pacchetto di dati e la sua ricezione da parte del destinatario. Con questo test, viene misurato quanto velocemente i dati viaggiano attraverso la rete, fornendo un'indicazione della reattività e della velocità della comunicazione tra due punti.

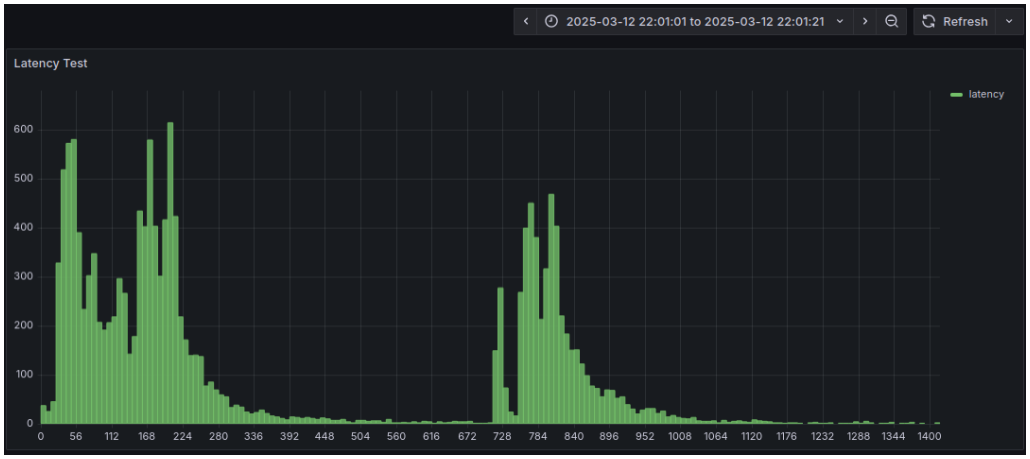

Figura 5.3: Grafico test latenza

La figura 5.3 mostra i risultati di un test di latenza, in cui l'asse delle ascisse rappresenta la latenza, mentre l'asse delle ordinate indica il numero di pacchetti. I test sono stati condotti sia in condizioni normali sia introducendo un ritardo sulle interfacce del router, utilizzando il seguente comando:

```
sudo tc qdisc add dev eth0 root netem delay Xms
```

Per evitare interferenze con gli altri test, che avrebbero potuto incrementare la latenza, sono state apportate modifiche alla schedulazione dell'esecuzione delle misurazioni di throughput e RTT rispetto a quanto descritto nella sezione 4.1. Inoltre è stato implementato un sistema di alert per monitorare le variazioni di latenza. Nello specifico, qualora la latenza media superi una determinata soglia (600ms), viene inviato automaticamente un avviso tramite un bot Telegram, consentendo un rapido intervento in caso di degrado delle prestazioni.

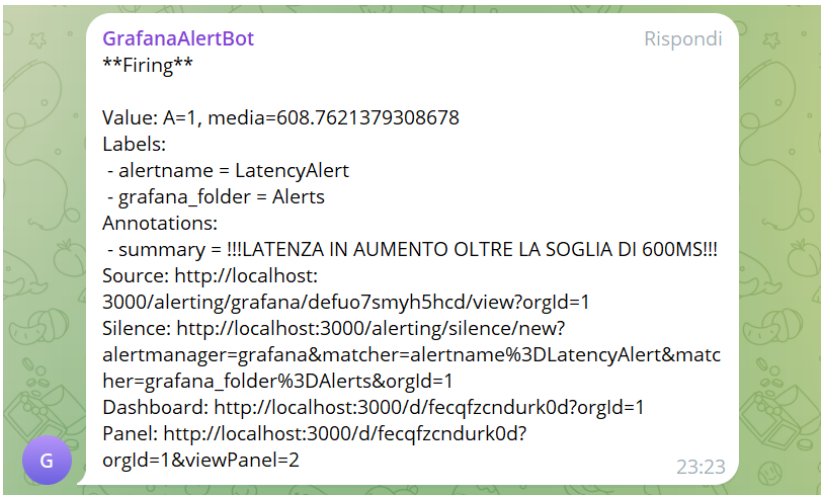

Figura 5.4: Notifica superamento soglia latenza

Un valore di latenza basso indica una rete rapida e reattiva, mentre valori elevati potrebbero indicare potenziali problemi nelle performance della rete, come ritardi nelle comunicazioni o inefficienze nella configurazione della rete.

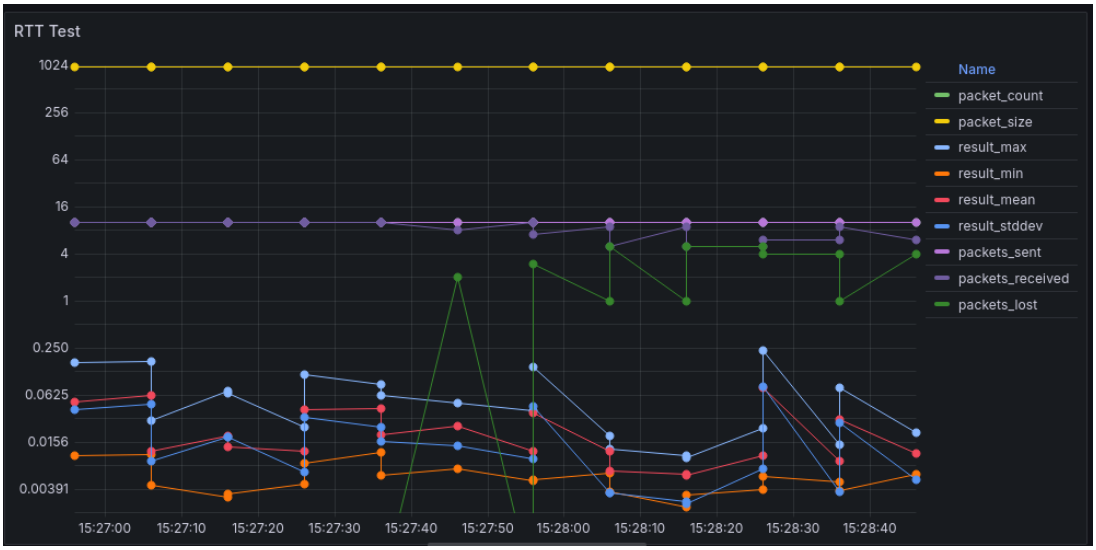

Figura 5.5: Notifica superamento soglia latenza

In figura 5.5 è riportato il risultato del test RTT, il quale consente di monitorare, tra le varie metriche, anche il numero di pacchetti trasmessi e quelli eventualmente persi durante l'esecuzione del test. Nella fase iniziale dell'analisi, in assenza di condizioni di rete degradate, non si registrano perdite di pacchetti, anche grazie alla natura controllata dell'ambiente di simulazione.

Per simulare condizioni di perdita, è stato introdotto un disturbo artificiale tramite il comando:

```
sudo tc qdisc add dev enp0s3 root netem loss X%
```

A partire dalle ore 15:27:30, è visibile un incremento nel numero di pacchetti persi, evidenziato nel grafico dalla linea verde, che riflette l'effetto della perdita indotta.

6. Conclusioni

Il progetto ha mostrato l'utilizzo di perfSONAR in un ambiente non completamente compatibile, adattandone la configurazione per il monitoraggio delle prestazioni di rete. Attraverso l'analisi di parametri come throughput, RTT e latenza, è stato possibile ottenere informazioni utili sulla qualità della connessione tra client e server.

L'integrazione di Grafana per la visualizzazione dei dati raccolti e l'utilizzo di MySQL per l'archiviazione ha permesso di gestire e monitorare le metriche in modo centralizzato e intuitivo. L'automazione della raccolta dei dati tramite pSConfig e l'uso del framework Flask per la gestione delle richieste HTTP ha reso l'intero sistema scalabile e facilmente adattabile a nuove necessità.

Un possibile sviluppo del progetto riguarda l'ampliamento della rete simulata, includendo ulteriori nodi e testpoint distribuiti, che permetterebbero di monitorare in modo più dettagliato le prestazioni su reti più ampie e complesse.

In sintesi, questo progetto ha posto le basi per un sistema robusto di monitoraggio delle prestazioni di rete, ma ci sono molteplici possibilità di espansione e miglioramento, che potrebbero contribuire ad adattarlo a contesti sempre più complessi e a garantire il miglior funzionamento delle reti moderne.

7. Sitografia

- perfSONAR
- Installazione perfSONAR in un container Docker
- Documentazione Installazione Grafana per Debian/Ubuntu
- Template per configurazione dei task tramite psConfig
- Documentazione Flask API