SAPIENZA
UNIVERSITÀ DI ROMA

Analisi del Traffico delle Reti Wireless con Grafana: Implementazione e Tecniche di Monitoraggio in Tempo Reale

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea in Ingegneria Informatica ed Automatica

Candidato

Simone Palumbo
Matricola 1938214

Relatore

Prof.ssa Francesca Cuomo

Correlatore

Dr. Andrea Lacava

Anno Accademico 2022/2023

Analisi del Traffico delle Reti Wireless con Grafana: Implementazione e Tecniche di Monitoraggio in Tempo Reale
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: simonepalumbospina@gmail.com

A mia madre e mio padre
Agli amici e alle amiche di una vita
Alle belle persone che ho incontrato durante questo percorso

A me stesso

Sommario

A differenza di altre discipline dell'*Information Technology*, lo studio delle reti di telecomunicazioni trova un profondo riscontro e una enorme utilità nella corretta visualizzazione e analisi dei dati. In questo progetto, ho avuto l'opportunità di creare un framework estensibile per la raccolta di dati generati da traffico di rete. Ho iniziato studiando le tecnologie utilizzate e ho acquisito le capacità per operare con i software proposti. Dopo la configurazione dello stack di software utilizzati, ho creato uno scenario personalizzato di ns-3 da usare per la raccolta dei dati. Ho poi lavorato sul design e l'implementazione di dashboards di controllo relative allo scenario proposto. Infine ho analizzato le metriche di traffico e discusso l'ottimizzazione della rete. I software che ho utilizzato sono Grafana, tool per la creazione di dashboard di controllo e monitoraggio eventi; InfluxDB, open-source time series database (TSDB) pensato per alte prestazioni e raccolta dati; Docker, famoso software di virtualizzazione; Wireshark, il famoso packet sniffer e ns-3, il simulatore di rete standard per l'analisi delle comunicazioni wireless.

Indice

Capitolo 1

Introduzione

Il lavoro è diviso nei seguenti capitoli:

Nel Capitolo 2 descrivo le tecnologie utilizzate

Nel Capitolo 3 si descrive il primo approccio tentato per la creazione del modello di canale: una ricerca in letteratura per modellare il canale di traffico.

Nel Capitolo 4 si descrive la creazione del modello di canale via campionamento di un applicazione già esistente

Nel Capitolo 5 si descrive la simulazione ns-3 nel dettaglio

Nel Capitolo 6 si conclude la relazione

Capitolo 2

Tecnologie Utilizzate

I software utilizzati in questo progetto rappresentano *cutting edge technologies* nei corrispettivi settori di applicazione. Illustrerò sinteticamente i loro aspetti chiave che hanno consentito di risolvere le sfide affrontate nel corso del progetto.

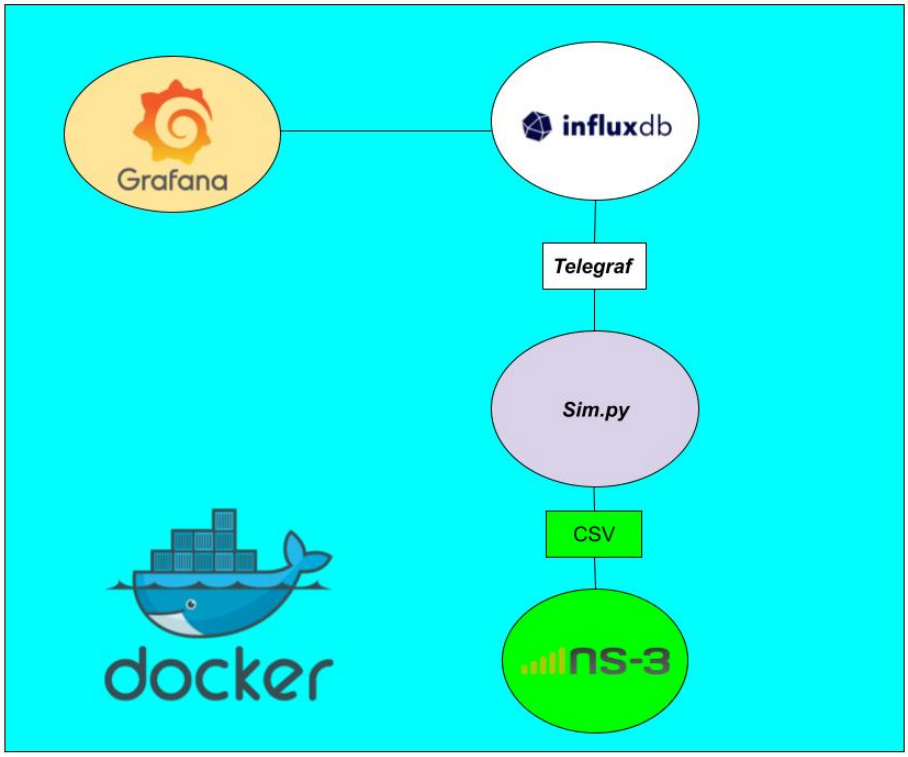


Figura 2.1. Diagramma dello Stack delle tecnologie utilizzate

2.1 Docker

Docker è un software open source che esegue processi in container isolati, semplificando il deployment di applicazioni. Si basa sulla virtualizzazione a livello di sistema operativo fornita dal kernel. I container Docker possono coesistere su un'unica istanza di Linux, eliminando la necessità di una macchina virtuale separata. Possono inoltre essere pacchettizzati con le dipendenze e eseguiti su qualsiasi macchina. Il loro compito è isolare l'applicazione dall'ambiente operativo, limitando ciò che l'applicazione può vedere in termini di processi, rete, filesystem e risorse (CPU, memoria, dispositivi I/O, rete).

2.1.1 Docker Compose, YAML e Dockerfile

Docker Compose è uno strumento che consente di definire e gestire facilmente l'orchestrazione di più container Docker. Con Docker Compose abbiamo descritto l'intera infrastruttura dell'applicazione, composta da più servizi, all'interno di un singolo file YAML. Questo file specifica le dipendenze tra i servizi, le configurazioni e altre informazioni necessarie per eseguire e coordinare l'intero stack dell'applicazione. Abbiamo inoltre utilizzato un Dockerfile per specificare la serie di istruzioni necessarie per creare la nostra immagine Docker, che è il pacchetto che comprende tutto il necessario per eseguire l'applicazione, compreso il codice sorgente, le dipendenze e le configurazioni.

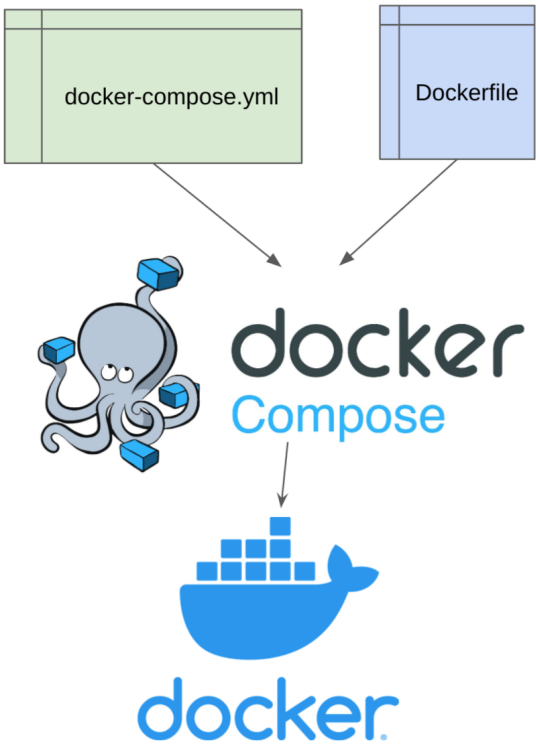


Figura 2.2. Diagramma della logica di Docker Compose

Il Dockerfile utilizzato è il seguente:

```
FROM wineslab/o-ran-sc-bldr-ubuntu18-c-go:9-u18.04 as buildenv
ARG log_level_e2sim=2
# log_level_e2sim = 0 ->  LOG_LEVEL_UNCOND   0
# log_level_e2sim = 1 -> LOG_LEVEL_ERROR     1
# log_level_e2sim = 2 -> LOG_LEVEL_INFO      2
# log_level_e2sim = 3 -> LOG_LEVEL_DEBUG     3

# Install E2sim
RUN mkdir -p /workspace
RUN apt-get update && apt-get install -y build-essential git cmake
    libsctp-dev autoconf automake libtool bison flex libboost-all-dev

WORKDIR /workspace

RUN git clone -b develop https://github.com/wineslab/ns-o-ran-e2-sim
    /workspace/e2sim

RUN mkdir /workspace/e2sim/e2sim/build
WORKDIR /workspace/e2sim/e2sim/build
RUN cmake .. -DDEV_PKG=1 -DLOG_LEVEL=${log_level_e2sim}

RUN make package
RUN echo "Going␣to␣install␣e2sim-dev"
RUN dpkg --install ./e2sim-dev_1.0.0_amd64.deb
RUN ldconfig

WORKDIR /workspace

# Install ns-3
RUN apt-get install -y g++ python3 python3-pip

RUN git clone -b release https://github.com/wineslab/ns-o-ran-ns3-
    mmwave /workspace/ns3-mmwave-oran
RUN git clone -b master https://github.com/o-ran-sc/sim-ns3-o-ran-e2
    /workspace/ns3-mmwave-oran/contrib/oran-interface

WORKDIR /workspace/ns3-mmwave-oran

RUN ./waf configure --enable-tests --enable-examples
RUN ./waf build

WORKDIR /workspace

# Install dependencies for script

RUN pip3 install statsd joblib numpy watchdog statsd-tags
RUN pip3 install statsd-telegraf --user
COPY sim_watcher.py /workspace/ns3-mmwave-oran/

WORKDIR /workspace

CMD [ "/bin/sh" ]
}
```

Listing 2.1. Codice del Dockerfile Utilizzato

Il codice presentato crea un'immagine per eseguire il simulatore E2sim e il framework ns-3. Di seguito viene fornita una descrizione del codice:

- La riga FROM wineslab/o-ran-sc-bldr-ubuntu18-c-go:9-u18.04 as buildenv specifica l'immagine di base su cui verrà costruita l'immagine finale.

- L'istruzione ARG log_level_e2sim=2 definisce una variabile di argomento che sarà utilizzata successivamente nel processo di build.
- Le righe successive iniziano l'installazione dei pacchetti necessari per il simulatore E2sim e il framework ns-3. Viene eseguito un apt-get update per aggiornare i repository dei pacchetti, seguito da un apt-get install per installare i pacchetti specifici.
- Successivamente, viene clonato il repository del simulatore E2sim da GitHub e viene configurato e compilato utilizzando CMake e Make. Viene inoltre installato il pacchetto resultante utilizzando dpkg.
- Dopo l'installazione di E2sim, viene clonato il repository del framework ns-3 mmWave da GitHub. Viene quindi eseguita la configurazione e la compilazione utilizzando lo script waf.
- Successivamente, vengono installate alcune dipendenze Python utilizzando pip3 install, inclusi i pacchetti statsd, joblib, numpy, watchdog e statsd-tags. Viene inoltre installato il pacchetto statsd-telegraf utilizzando l'opzione –user.
- Infine, viene copiato lo script sim_watcher.py nella directory del framework ns-3 mmWave.
- La sezione finale CMD ["/bin/sh"] specifica il comando predefinito da eseguire quando viene avviata un'istanza del container Docker.

Il Dockerfile, come già detto, viene utilizzato per creare l'immagine ns-3 che sarà uno dei componenti dell'architettura di servizi che il file YAML descrive. Precisamente, il codice YAML è il seguente:

```
version: '3.6'
services:
  telegraf:
    image: telegraf:1.18-alpine
    volumes:
    - ./telegraf/etc/telegraf.conf:/etc/telegraf/telegraf.conf:ro
    depends_on:
      - influxdb
    links:
      - influxdb
    ports:
    - '127.0.0.1:8125:8125/udp'

  influxdb:
    image: influxdb:1.8-alpine
    env_file: configuration.env
    ports:
      - '127.0.0.1:8086:8086'
    command: sh -c "influxd & sleep 10 && influx -database influx
    -execute 'delete from /\w*/'; tail -f /dev/null"
    volumes:
```

```
22        - ./:/imports
23        - influxdb_data:/var/lib/influxdb
24
25    grafana:
26      image: grafana/grafana:8.0.2
27      depends_on:
28        - influxdb
29      env_file: configuration.env
30      links:
31        - influxdb
32      ports:
33        - '3000:3000'
34      volumes:
35        - grafana_data:/var/lib/grafana
36        - ./grafana/provisioning/:/etc/grafana/provisioning/
37        - ./grafana/dashboards/:/var/lib/grafana/dashboards/
38
39    ns3:
40      build: ./statsd_ns3
41      stdin_open: true
42      tty: true
43      network_mode: "host"
44      depends_on:
45        - telegraf
46
47  volumes:
48    grafana_data: {}
49    influxdb_data: {}
```

La versione del formato del file YAML è '3.6'. La sezione "services" elenca i servizi che verranno eseguiti come container Docker:

- Il servizio "telegraf" utilizza l'immagine "telegraf:1.18-alpine". Viene montato un volume che collega il file di configurazione "telegraf.conf" dalla cartella locale "./telegraf/etc/" al percorso "/etc/telegraf/telegraf.conf" all'interno del container. Dipende dal servizio "influxdb" e crea un collegamento a esso. Inoltre, mappa la porta UDP 8125 del container alla porta UDP 8125 dell'host (127.0.0.1:8125:8125/udp).
- Il servizio "influxdb" utilizza l'immagine "influxdb:1.8-alpine". Viene utilizzato un file di ambiente denominato "configuration.env". Mappa la porta TCP 8086 del container alla porta TCP 8086 dell'host (127.0.0.1:8086:8086). Esegue un comando shell che avvia il demone InfluxDB, attende 10 secondi, esegue un'istruzione InfluxQL per cancellare i dati dal database "influx", e infine rimane in attesa senza terminare (tail -f /dev/null). Viene creato un volume per collegare la directory corrente (".") alla directory "/imports" all'interno del container InfluxDB. Un secondo volume è creato per la directory dei dati del database InfluxDB ("/var/lib/influxdb").
- Il servizio "grafana" utilizza l'immagine "grafana/grafana:8.0.2". Dipende dal servizio "influxdb" e crea un collegamento a esso. Mappa la por-

ta TCP 3000 del container alla porta TCP 3000 dell'host (3000:3000). Viene creato un volume per collegare la directory "/var/lib/grafana" all'interno del container alla directory "grafana_data" nel sistema host. Altri due volumi sono creati per la directory di provisioning di Grafana ("/etc/grafana/provisioning/") e per le dashboard di Grafana ("/var/lib/grafana/dashboards/").

- Il servizio "ns3" viene compilato utilizzando il nostro Dockerfile, fornito nella directory "./statsd_ns3". Vengono aperti stdin e tty per il container. Viene utilizzata la modalità di rete "host" per condividere il namespace di rete dell'host con il container. Dipende dal servizio "telegraf".
- La sezione "volumes" definisce due volumi senza specificare alcuna configurazione aggiuntiva: "grafana_data" e "influxdb_data".

Questa configurazione complessiva consente di eseguire e collegare i container Docker per la nostra stack. I servizi citati verranno approfonditi nel corso del capitolo.

2.2 NS-3

NS-3 (Network Simulator 3) è un simulatore di reti open-source scritto in C++. Una delle caratteristiche distintive di ns-3 è la sua architettura modulare, che consente agli utenti di personalizzare e estendere il simulatore secondo le proprie esigenze. Con ns-3 abbiamo sviluppato la nostra simulazione da utilizzare per raccogliere i dati, che sarà descritta estensivamente nel Capitolo 5

2.3 InfluxDB

InfluxDB è un database open source per dati di serie temporali utilizzato per archiviare, recuperare e monitorare informazioni in campi come l'IoT e l'analisi in tempo reale. Come mostrato in figura 2.3, i TSDB[1], a differenza dei classici database relazionali, organizzano i dati in misure, serie e punti. Ogni punto contiene coppie chiave-valore chiamate fieldset e timestamp. I punti sono raggruppati in serie tramite coppie chiave-valore chiamate tagset, e le serie sono raggruppate in misure tramite un identificatore di stringa. I punti sono indicizzati in base all'ora e ai tag. I criteri di conservazione definiscono come i dati vengono downsamplati ed eliminati per una misurazione. Le query continue vengono eseguite periodicamente e i risultati vengono memorizzati in una misurazione di destinazione.

2.3.1 Telegraf

Telegraf è un agente di raccolta e elaborazione dati integrato in InfluxDB. Telegraf va configurato tramite un file di configurazione, aggiungendo input, output e plugin necessari per il task che si vuole eseguire. Nel nostro caso, abbiamo dovuto affrontare il problema legato al timestamp memorizzato da InfluxDB, che è quello di arrivo a Telegraf, e non quello reale della simulazione.

[1] Acronimo di Time Series DataBases

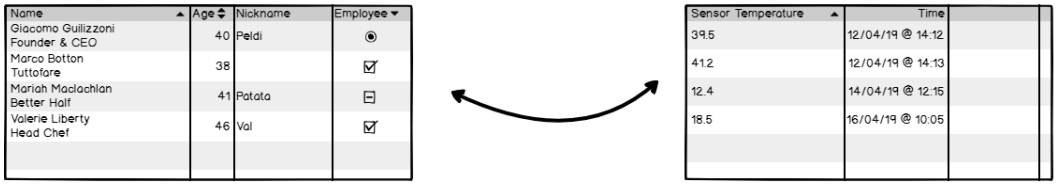


Figura 2.3. Confronto tra un Database Relazionale e un Database Time Series

```
name: L3servingSINR3gpp_00011_cp
time                host        metric_type timestamp           value
----                ----        ----------- ---------           -----
1687938058427000064 0f2ca7e1dd41 gauge       1687938058427000064 46
1687938058527000064 0f2ca7e1dd41 gauge       1687938058527000064 43
1687938058627000064 0f2ca7e1dd41 gauge       1687938058627000064 47
1687938058727000064 0f2ca7e1dd41 gauge       1687938058727000064 61
1687938058827000064 0f2ca7e1dd41 gauge       1687938058827000064 21
1687938058927000064 0f2ca7e1dd41 gauge       1687938058927000064 13
1687938059027000064 0f2ca7e1dd41 gauge       1687938059027000064 26
1687938059127000064 0f2ca7e1dd41 gauge       1687938059127000064 23
1687938059227000064 0f2ca7e1dd41 gauge       1687938059227000064 27
1687938059327000064 0f2ca7e1dd41 gauge       1687938059327000064 25
1687938059427000064 0f2ca7e1dd41 gauge       1687938059427000064 24
1687938059527000064 0f2ca7e1dd41 gauge       1687938059527000064 26
1687938059627000064 0f2ca7e1dd41 gauge       1687938059627000064 25
1687938059727000064 0f2ca7e1dd41 gauge       1687938059727000064 22
1687938059827000064 0f2ca7e1dd41 gauge       1687938059827000064 30
1687938059927000064 0f2ca7e1dd41 gauge       1687938059927000064 32
1687938060027000064 0f2ca7e1dd41 gauge       1687938060027000064 23
1687938060127000064 0f2ca7e1dd41 gauge       1687938060127000064 78
1687938060227000064 0f2ca7e1dd41 gauge       1687938060227000064 29

name: L3servingSINR3gpp_00012_cp
time                host        metric_type timestamp           value
----                ----        ----------- ---------           -----
1687938058427000064 0f2ca7e1dd41 gauge       1687938058427000064 65
1687938058527000064 0f2ca7e1dd41 gauge       1687938058527000064 38
1687938058627000064 0f2ca7e1dd41 gauge       1687938058627000064 41
1687938058727000064 0f2ca7e1dd41 gauge       1687938058727000064 44
1687938058827000064 0f2ca7e1dd41 gauge       1687938058827000064 32
1687938058927000064 0f2ca7e1dd41 gauge       1687938058927000064 37
1687938059027000064 0f2ca7e1dd41 gauge       1687938059027000064 34
1687938059127000064 0f2ca7e1dd41 gauge       1687938059127000064 74
1687938059227000064 0f2ca7e1dd41 gauge       1687938059227000064 54
1687938059327000064 0f2ca7e1dd41 gauge       1687938059327000064 44
1687938059427000064 0f2ca7e1dd41 gauge       1687938059427000064 50
1687938059527000064 0f2ca7e1dd41 gauge       1687938059527000064 37
1687938059627000064 0f2ca7e1dd41 gauge       1687938059627000064 45
1687938059727000064 0f2ca7e1dd41 gauge       1687938059727000064 45
1687938059827000064 0f2ca7e1dd41 gauge       1687938059827000064 47
1687938059927000064 0f2ca7e1dd41 gauge       1687938059927000064 40
1687938060027000064 0f2ca7e1dd41 gauge       1687938060027000064 47
1687938060127000064 0f2ca7e1dd41 gauge       1687938060127000064 45
1687938060227000064 0f2ca7e1dd41 gauge       1687938060227000064 46
```

Figura 2.4. Visualizzazione tramite terminale di due tabelle influx.

Per risolvere questo problema abbiamo utilizzato il plugin *Starlark*, che ci ha permesso di aggiungere il timestamp corretto ad InfluxDB tramite l'utilizzo di *tag*. In figura 2.4 è possibile vedere come il tag *timestamp* non abbia sovrascritto il timestamp di influx, nella colonna *time*.

Configurazione di Telegraf:

nel file telegraf.conf è possibile vedere quali plugin telegraf ha a disposizione di default e aggiungerne di nuovi. Il file di configurazione è inoltre necessario per dichiarare da quali input le metriche vanno raccolte e a quali output vanno inviate. Nel nostro caso, è stato necessario configurare Telegraf per utilizzare il plugin Starlark, in modo da poter utilizzare il timestamp della simulazione per visualizzare i dati che da Influx arrivano a Grafana. Notare che diversi attributi del file di configurazione non sono stati mostrati poiché non rilevanti opppure commentati, quindi mostriamo ora solo le parti rilevanti del file di configurazione:

```
[agent]
```

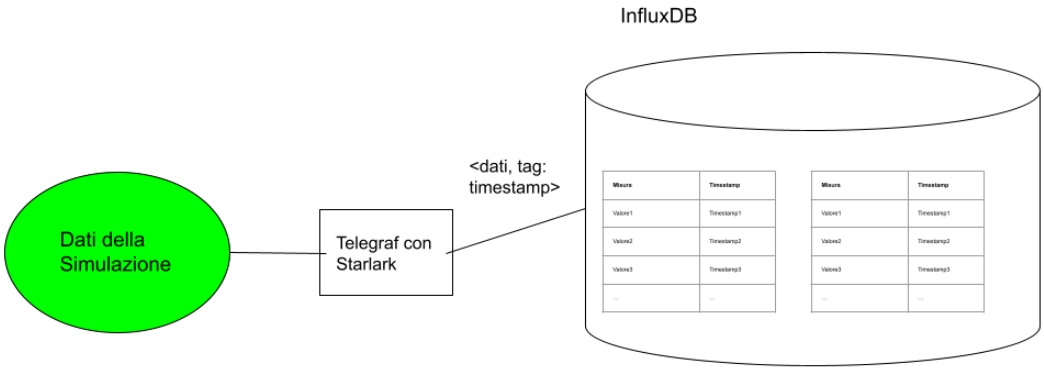


Figura 2.5. Starlark non modifica il timestamp interno di InfluxDB: aggiunge ad ogni dato un tag che indica il timestamp della simulazione

```
  interval = "5s"
  round_interval = true
  metric_buffer_limit = 10000
  flush_buffer_when_full = true
  collection_jitter = "0s"
  flush_interval = "1s"
  flush_jitter = "0s"
  debug = false
  quiet = false
  hostname = ""
```

La sezione precedente configura l'agente Telegraf e contiene impostazioni di default che non sono state modificate:

- *interval*, che descrive l'ntervallo di raccolta predefinito per tutti gli input, è lasciato al suo valore di default, in modo che le metriche vengano raccolte ogni 5 secondi.
- *metric_buffer_limit* indica il limite di buffer per ogni output prima di eseguire il flushing, mentre *flush_buffer_when_full* indica che il flushing del buffer viene eseguito ogni volta che questo è pieno.
- collection_jitter introduce una variazione casuale nell'intervallo di raccolta per evitare sovraccarichi simultanei.

Vediamo ora la configurazione degli output.

```
[[outputs.influxdb]]
  urls = ["http://influxdb:8086"] # required
  database = "influx" # required
  precision = "s"
  timeout = "5s"
```

Nel nostro caso c'è un solo output, InfluxDB:

- *urls* specifica l'URL completo dell'istanza InfluxDB a cui inviare le metriche. Nel nostro codice l'URL è impostato su "http://influxdb:8086", che indica che Telegraf si connetterà all'istanza InfluxDB all'indirizzo "influxdb" sulla porta 8086.

- *database* specifica il nome del database di destinazione in InfluxDB. Se il database non esiste, Telegraf lo creerà automaticamente.
- *precision* specifica la precisione delle scritture per le metriche inviate a InfluxDB. I valori validi sono "ns" (nanosecondi), "us" o "µs" (microsecondi), "ms" (millisecondi), "s" (secondi), "m" (minuti) e "h" (ore). Nel nostro codice, la precisione è impostata su "s" per i secondi.
- *timeout* specifica il timeout per le richieste di scrittura verso il client InfluxDB, nel nostro caso, è impostato su "5s", che è il valore di default se non viene fornito.

Vediamo ora gli input di Telegraf:

```
[[inputs.statsd]]
  protocol = "udp"
  max_tcp_connections = 250
  tcp_keep_alive = false
  service_address = ":8125"
  delete_gauges = true
  delete_counters = true
  delete_sets = true
  delete_timings = true
  percentiles = [90]
  metric_separator = "_"
  parse_data_dog_tags = false
  allowed_pending_messages = 10000
  percentile_limit = 1000
```

Il primo input, nonché quello rilevante per i nostri scopi, è il protocollo StatsD. Analizziamo la configurazione:

- *protocol* specifica il protocollo utilizzato per la comunicazione con il server StatsD. I valori possibili sono "tcp", "udp4", "udp6" o "udp", nel nostro caso è impostato su "udp", l'opzione predefinita.
- *max_tcp_connections* è applicabile solo se il protocollo è impostato su "tcp" e specifica il numero massimo di connessioni TCP consentite, con valore predefinito 250.
- *tcp_keep_alive* abilita o disabilita le sonde di keep-alive TCP. Il valore predefinito è false.
- *service_address* specifica l'indirizzo IP e la porta su cui il listener UDP di Telegraf per StatsD è in ascolto. Nel codice fornito, l'indirizzo è impostato su ":8125", il che significa che Telegraf ascolterà su tutte le interfacce su cui è disponibile il traffico in ingresso sulla porta 8125.
- *delete_gauges*, *delete_counters*, *delete_sets*, *delete_timings* controllano quando Telegraf cancella la sua cache dei valori precedenti. Se impostate su true (valore predefinito), Telegraf cancellerà i valori di gauge, counter, set, timing e histogram ad ogni intervallo. Se impostate su false, la cache verrà cancellata solo quando il demone di Telegraf viene riavviato.

- *percentiles* specifica i percentili da calcolare per le statistiche di timing e histogram. Nel nostro codice, viene calcolato solo il percentile 90.
- *metric_separator* specifica il separatore da utilizzare tra gli elementi di una metrica StatsD, impostato su "_".
- *parse_data_dog_tags* specifica se analizzare o meno i tag nel formato di metrica StatsD di DataDog. Nel nostro caso non è rilevante, quindi la settiamo a false.
- *allowed_pending_messages* specifica il numero massimo di messaggi UDP consentiti in coda. Una volta raggiunto il limite, il server StatsD inizierà a eliminare i pacchetti. Settiamo il limite a 10000.
- *percentile_limit* specifica il numero massimo di valori di timing/histogram da tenere traccia per ogni misurazione nel calcolo dei percentili. Aumentare questo limite aumenta l'accuratezza dei percentili, ma aumenta anche l'utilizzo della memoria e il tempo della CPU. Limite impostato a 1000.

Sono poi presenti altri input non rilevanti rilevanti per la comunicazione all'interno della nostra stack, ma presenti di default nel file di configurazione e riguardanti le prestazioni della macchina, come [[inputs.cpu]] per leggere metriche sull'utilizzo della CPU o [[inputs.disk]] per leggere metriche sull'utilizzo dei dischi.

```
[[processors.starlark]]
source = '''
def apply(metric):
  if "timestamp" in metric.tags:
    metric.time = int(metric.tags["timestamp"])
    return metric
'''
```

Infine configuriamo il processore Starlark. Questo plugin utilizza il linguaggio di scripting Starlark per applicare una trasformazione alle metriche durante l'elaborazione da parte di Telegraf.

- *source* definisce il codice Starlark che verrà eseguito dal processore.
- La funzione *apply* prende in input una metrica e controlla se il tag "timestamp" è presente nella metrica. Se il tag è presente, il valore del tag viene utilizzato per impostare il timestamp della metrica e convertito in intero. Infine, la funzione restituisce la metrica con il timestamp aggiornato.

In sintesi, usiamo Starlark per aggiungere il timestamp della simulazione come tag (dato che non possiamo modificare il timestamp di InfluxDB).

2.4 Statsd

StatsD è un *daemon* open-source per la raccolta e l'aggregazione di dati di monitoraggio. Utilizza un protocollo di rete basato su UDP o TCP per inviare le metriche a un server StatsdD. Le metriche raccolte possono essere di

diversi tipi: contatori, gauge, insiemi, campioni ecc.. Noi ci siamo concentrati esclusivamente sul tipo *gauge*, cioè un punto di dati instantaneo. I gauge sono il tipo di metrica più adatto per monitorare i dati che avevamo a disposizione, e inoltre ci hanno permesso di evitare il problema dell'aggregazione (non voluta nel nostro caso), che invece gli altri tipi di metriche hanno, e poter leggere istantaneamente il singolo valore con un certo timestamp. Per usare StatsD è stato necessario scrivere uno script in python *sim.py* che raccoglie i dati della simulazione da dei file CSV e li invia a Telegraf.

2.4.1 Watchdog in Python:

Lo script Python monitora i file che ci interessano nella directory corrente, li analizza e invia i dati a un server Telegraf utilizzando il protocollo StatsD. Per fare ciò usiamo la libreria watchdog:

```
import csv
from typing import Dict, List, Set, Tuple
import numpy as np
from watchdog.events import PatternMatchingEventHandler
from watchdog.observers import Observer
import threading
import time
from statsd import StatsClient
import re

lock = threading.Lock()
```

Definiamo la classe SimWatcher, che eredita da PatternMatchingEventHandler della libreria watchdog. Questa classe contiene ci permette di gestire gli eventi nella directory e l'invio dei dati a Telegraf. Il codice è stato opportunamente documentato:

```
class SimWatcher(PatternMatchingEventHandler):

    """
    A Python event handler that looks for specific .txt
       formatted as csv files, parses data from them, and
       sends it to Telegraf as part of a watchdog object.

    Attributes
    ----------
    patterns : list
        A list of file patterns that the event handler will
           monitor.

    kpm_map : Dict[Tuple[int, int, int], List]
        A dictionary that contains the data for every
           measurement in the csv file.

    consumed_keys : Set[Tuple[int, int, int]]
        A list of keys already used in the dictionary.

    telegraf_host : str
        The host address of the Telegraf server "localhost"
           by default.

    telegraf_port : int
```

```
        The port number of the Telegraf server, 8125 by
            default.

    statsd_client : StatsClient
        The StatsD client for sending metrics to Telegraf
            automatically created by default with the above
            parameters.


    Methods
    -------
    __init__(self)
        Initializes the event handler

    on_created(self, event)
        Handles the on_created event triggered when a new
            file is created.

    on_closed(self, event)
        Handles the on_closed event triggered when a file is
            closed.

    on_modified(self, event)
        Handles the on_modified event triggered when a file
            is modified.

    send_to_telegraf(self, ue, values, fields, file_type)
        Formats and sends data to the Telegraf agent
    """

    patterns = ['cu-up-cell-*.txt', 'cu-cp-cell-*.txt',
        "du-cell-*.txt"]
    kpm_map: Dict[Tuple[int, int, int], List] = {}
    consumed_keys: Set[Tuple[int, int, int]]
    telegraf_host = "localhost"
    telegraf_port = 8125
    statsd_client = StatsClient(telegraf_host, telegraf_port,
        prefix = None)

    def __init__(self):

        """
        Initializes the class
        """

        PatternMatchingEventHandler.__init__(self,
            patterns=self.patterns,
        ignore_patterns=[], ignore_directories=True,
            case_sensitive=False)
        self.directory = ''
        self.consumed_keys = set()

    def on_created(self, event):

        """
        Handles the on_created event triggered when a new
            file is created.
        """
```

```
68
69          super().on_created(event)
70
71      def on_modified(self, event):
72
73          """
74          Handles the on_modified event triggered when a file
                is modified.
75          """
76
77          super().on_modified(event)
78
79          lock.acquire()
80          with open(event.src_path, 'r') as file:
81              reader = csv.DictReader(file)
82
83              for row in reader:
84                  timestamp = int(row['timestamp'])
85                  ue_imsi = int(row['ueImsiComplete'])
86                  ue = row['ueImsiComplete']
87
88                  if re.search('cu-up-cell-[2-5].txt',
                        file.name):
89                      key = (timestamp, ue_imsi, 0)
90                  if re.search('cu-cp-cell-[2-5].txt',
                        file.name):
91                      key = (timestamp, ue_imsi, 1)
92                  if re.search('du-cell-[1-5].txt', file.name):
93                      key = (timestamp, ue_imsi, 2)
94                  if  file.name == './cu-up-cell-1.txt':
95                      key = (timestamp, ue_imsi, 3)   # to see
                            data for eNB cell
96                  if file.name == './cu-cp-cell-1.txt':
97                      key = (timestamp, ue_imsi, 4)   # same
                            here
98
99
100                 if key not in self.consumed_keys:
101
102                     if key not in self.kpm_map:
103                         self.kpm_map[key] = []
104
105                     fields = list()
106
107                     for column_name in reader.fieldnames:
108                         if row[column_name] == '':
109                             continue
110                         self.kpm_map[key].append(
111                         float(row[column_name]))
112                         fields.append(column_name)
113
114                     regex = re.search(r"\w*-(\d+)\.txt",
                            file.name)
115                     fields.append('file_id_number')
116                     self.kpm_map[key].append(regex.group(1))
                                # last item of list will be
                            file_id_number
117
```

```
                    self.consumed_keys.add(key)
                    self._send_to_telegraf(ue=ue,
                        values=self.kpm_map[key],
                        fields=fields, file_type=key[2])

        lock.release()

    def on_closed(self, event):

        """
        Handles the on_closed event triggered when a file is
            closed.
        """

        super().on_closed(event)

    def _send_to_telegraf(self, ue:int, values:List,
        fields:List, file_type:int):

        """
        Formats and sends data to the Telegraf agent.

        Parameters
        ----------
        ue : int
            Value extracted from csv, represents the ue.
        values : List
            List of metrics to send to Telegraf.
        fields : List
            List of field names corresponding to the metrics
                in the values parameter.
        file_type: int
            Ranges from 1 to 3. Represents if the metrics
                come from a up, cu or du file respectively.

        Notes
        -----
        The "stat" parameter of the gauge() function will
            then be shown as the table name in InfluxDB, this
            explains the stat string formatting for each
            metric.
        """

        # send data to telegraf
        pipe = self.statsd_client.pipeline()

        # convert timestamp in nanoseconds (InfluxDB)
        timestamp = int(values[0]*(pow(10,6))) # int because
            of starlark

        i = 0
        for field in fields:

            if field == 'file_id_number':
                continue

            # convert pdcp_latency
            if field == 'DRB.PdcpSduDelayDl.UEID
```

```
                (pdcpLatency)':
                    values[i] = values[i]*pow(10, -1)

            if field == 'DRB.PdcpSduDelayDl
                (cellAverageLatency)':
                    stat = 'DRB.PdcpSduDelayDl
                        (cellAverageLatency)_cell_' + values[-1]
                    stat = stat.replace(' ','')
                    pipe.gauge(stat=stat, value=values[i],
                        tags={'timestamp':timestamp})
                    i+=1
                    continue

            if field == 'm_pDCPBytesDL (cellDlTxVolume)':
                    stat = 'm_pDCPBytesDL (cellDlTxVolume)_cell_'
                        + values[-1]
                    stat = stat.replace(' ','')
                    pipe.gauge(stat=stat, value=values[i],
                        tags={'timestamp':timestamp})
                    i+=1
                    continue

            if field == 'numActiveUes':
                    stat = 'numActiveUes_cell_' + values[-1]
                    pipe.gauge(stat=stat, value=values[i],
                        tags={'timestamp':timestamp})
                    i+=1
                    continue

            stat = field + '_' + ue
            if file_type == 0 or file_type == 3:
                    stat += '_up'
            if file_type == 1 or file_type == 4:
                    stat += '_cp'
            if file_type == 2:
                    stat += '_du'
            stat = stat.replace(' ','')
            pipe.gauge(stat=stat, value = values[i],
                tags={'timestamp':timestamp})
            i+=1
        pipe.send()


if __name__ == "__main__":
    event_handler = SimWatcher()
    observer = Observer()
    observer.schedule(event_handler, ".", False)
    observer.start()

    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()

    observer.join()
```

In sintesi, il codice permette di monitorare l'updating dei file CSV cu, cp, e du

prodotti dalla simulazione nella directory, analizzarli e inviare i dati a Telegraf utilizzando StatsD per la raccolta e la visualizzazione delle metriche.

2.5 Grafana

Grafana è uno strumento web per la visualizzazione e l'analisi interattiva dei dati. Consente di creare infografiche e pannelli di monitoraggio, unificando eventualmente diverse fonti di dati. I dati possono essere visualizzati su pannelli personalizzati chiamati dashboard. La creazione dei pannelli può essere fatta utilizzando la GUI web o tramite JSON.

2.5.1 Creazione delle Dashboard

Quattro sono le dashboard create:

- Una dashboard per i dati provenienti dai file cp-up
- Una per i dati provenienti dai file du-up
- Una per i dati provenienti dai file up-up
- Una per dati aggregati

Dato l'alto numero di elementi da graficare, il grosso del lavoro riguardante la creazione delle dashboard è stato svolto utilizzando JSON piuttosto che la GUI. La struttura JSON di una dashboard è la seguente:

```
{
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": "--Grafana--",
        "enable": true,
        "hide": true,
        "iconColor": "rgba(0,211,255,1)",
        "name": "Annotations&Alerts",
        "type": "dashboard"
      }
    ]
  },
  "editable": true,
  "gnetId": null,
  "graphTooltip": 0,
  "links": [],
```

Listing 2.2. proprietà della dashboard

Grafana impone delle proprietà all'oggetto JSON, come *annotations* che specifica le annotazioni o gli avvisi da visualizzare nel pannello ed *editable* che specifica se il pannello è modificabile o meno. *panels* invece contiene un array di oggetti che rappresentano i pannelli dove verranno mostrati i grafici all'interno della dashboard. In questo pezzo di codice[2], inseriamo solo due pannelli.

[2] estratto dalla dashboard Up_Metrics (v. sezione 2.6)

```
"panels": [
  {
    "datasource": null,
    "description": "txPdcpPduBytesNrRlc",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisLabel": "QosFlow_PdcpPduVolumeDL_Filter_UEID",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "lineInterpolation": "linear",
          "lineWidth": 1,
          "pointSize": 5,
          "scaleDistribution": {
            "type": "linear"
          },
          "showPoints": "auto",
          "spanNulls": false,
          "stacking": {
            "group": "A",
            "mode": "none"
          },
          "thresholdsStyle": {
            "mode": "off"
          }
        },
        "mappings": [],
        "thresholds": {
          "mode": "absolute",
          "steps": [
            {
              "color": "green",
              "value": null
            },
            {
              "color": "red",
              "value": 80
            }
          ]
        },
        "unit": ""
      },
      "overrides": []
    },
    "gridPos": {
      "h": 8,
      "w": 12,
      "x": 0,
      "y": 0
    },
    "id": 18,
```

```
      "options": {
        "legend": {
          "calcs": [],
          "displayMode": "list",
          "placement": "bottom"
        },
        "tooltip": {
          "mode": "single"
        }
      },
      "targets": [
        {
          "query":
          "SELECT␣\"value\"␣FROM␣/QosFlow_PdcpPduVolumeDL_Filter_UEID
              \(txPdcpPduBytesNrRlc\\)_[0-9]{5}_up/␣WHERE␣$timeFilter"
            ,
          "rawQuery": true,
          "refId": "A"
        }
      ],
      "title": "PDCP␣PDU␣Volume",
      "transformations": [
        {
          "id": "renameByRegex",
          "options": {
            "regex":
            "QosFlow_PdcpPduVolumeDL_Filter_UEID\\(
               txPdcpPduBytesNrRlc\\)_([0-9]{5})_up",
            "renamePattern": "$1"
          }
        }
      ],
      "type": "timeseries"
    },
```

Listing 2.3. proprietà di "panels" e primo panel

Ogni pannello deve avere un ID univoco. Il primo pannello, con ID 18, è di tipo "timeseries", quindi mostrerà un grafico delle serie temporali. Come si può notare, è possibile configurare gli assi, le soglie di colori per i valori, e molteplici altri aspetti estetici del grafico. È ovviamente inoltre possibile configurare la query che vogliamo usare per estrarre i dati dalla fonte. In questo caso, la query utilizza delle regex per estrarre dati da tabelle numerate in base all'utente al quale appartengono i dati. Infine, è anche possibile configurare altre opzioni di visualizzazione come le unità di misura.

```
    {
      "datasource": null,
      "description": "pdcpLatency",
      "fieldConfig": {
        "defaults": {
          "color": {
            "mode": "palette-classic"
          },
          "custom": {
            "axisLabel": "DRB.PdcpSduDelayDl.UEID",
            "axisPlacement": "auto",
            "barAlignment": 0,
            "drawStyle": "line",
            "fillOpacity": 0,
            "gradientMode": "none",
            "hideFrom": {
```

```
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "lineInterpolation": "linear",
          "lineWidth": 1,
          "pointSize": 5,
          "scaleDistribution": {
            "type": "linear"
          },
          "showPoints": "auto",
          "spanNulls": true,
          "stacking": {
            "group": "A",
            "mode": "none"
          },
          "thresholdsStyle": {
            "mode": "off"
          }
        },
        "mappings": [],
        "thresholds": {
          "mode": "absolute",
          "steps": [
            {
              "color": "green",
              "value": null
            },
            {
              "color": "red",
              "value": 80
            }
          ]
        },
        "unit": "ms"
      },
      "overrides": []
    },
    "gridPos": {
      "h": 8,
      "w": 12,
      "x": 12,
      "y": 0
    },
    "id": 2,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom"
      },
      "tooltip": {
        "mode": "single"
      }
    },
  },
],
```

Listing 2.4. secondo panel

Con il secondo panel, analoga alla prima, si chiude la lista "panels". Utilizzare JSON invece che la GUI di Grafana ha permesso la creazione di dashboard con più di 50 elementi in modo molto rapido.

```
  "refresh": false,
  "schemaVersion": 30,
  "style": "dark",
  "tags": [],
  "templating": {
    "list": []
  },
  "time": {
    "from": "2023-05-23T13:27:49.000Z",
    "to": "2023-05-23T13:27:51.000Z"
  },
  "timepicker": {
    "refresh_intervals": [
      "5s"
    ]
  },
  "timezone": "",
  "title": "Up_Metrics",
  "uid": "SDOyaFLVz",
  "version": 7
}
```

Listing 2.5. ulteriori proprietà della dashboard

Infine, ci sono altre proprietà dell'oggetto JSON che specificano opzioni di configurazione aggiuntive per la dashboard:

- *refresh* specifica se il pannello deve essere aggiornato automaticamente o meno
- *schemaVersion* specifica la versione dello schema di Grafana utilizzata per la configurazione del pannello.
- *style* specifica lo stile della dashboard. Nel nostro caso è impostato sulla *dark mode*.
- *tags* specifica i tag associati alla dashboard. Nel nostro caso non ci sono tag specificati. É importante sottolineare che questi tag non sono in nessun modo collegati con i tag di Telegraf di cui si è parlato precedentemente.
- *templating* specifica eventuali template di variabili utilizzati nella dashboard.
- *time* specifica l'intervallo di tempo dei dati visualizzati nella dashboard. La proprietà *from* indica il timestamp di inizio dell'intervallo e la proprietà *to* indica il timestamp di fine dell'intervallo.
- *timepicker* specifica le opzioni del selettore di intervallo di tempo nella dashboard. Nel nostro caso l'intervallo di aggiornamento è impostato su 5 secondi.
- *timezone* specifica un eventuale fuso orario alternativo da utilizzare. Se non impostato, come nel nostro caso, viene usato quello predefinito, estratto dalle opzioni del browser su cui si esegue Grafana
- *title* specifica il titolo della dashboard.
- *uid* specifica l'ID univoco della dashboard.
- *version* specifica la versione della configurazione del pannello.

2.6 Esempio con simulazione 5G:

Utilizzando uno scenario 5G di ns-3 da https://github.com/wineslab/, andiamo a visualizzare i grafici delle prestazioni tramite il nostro framework. Lo scenario scelto ha un massimo di 12 utenti che si collegano a celle 5G numerate. Per eseguire la stack:

- lanciamo i container dei servizi con

  ```
  $ docker compose up
  ```

- eseguiamo il watchdog: su un altro terminale, entriamo nel container del servizio ns-3 con

  ```
  $ docker compose exec ns3 /bin/bash
  ```

  Avremo ora a disposizione una shell bash grazie alla quale potremo interagire col servizio. Eseguiamo quindi il watchdog con

  ```
  $ cd ns3-mmwave-oran
  $ python3 sim_watcher.py
  ```

- facciamo partire la simulazione 5G: su un altro terminale:

  ```
  $ docker compose exec ns3 /bin/bash
  $ cd ns3-mmwave-oran
  $ ./waf --run "scratch/scenario-zero.cc --enableE2FileLogging=1"
  ```

  Nell'ultima istruzione è possibile eventualmente utilizzare

  1. `--RngRun=n` con n intero positivo, per impostare il seed per il generatore di numeri pseudo-randomici. Con lo stesso seed ogni esecuzione produrrà gli stessi risultati, rendendo quindi lo scenario deterministico.
  2. `--simTime=x` per impostare a x secondi il tempo di simulazione. Di default sono 2 secondi.

  la simulazione produce dei csv divisi, come già detto, in cu-cp-cell, cu-up-cell, du-cell.

- Il watchdog, monitorando la directory, noterà l'aggiornamento dei file e inizierà a estrarre e inviare i dati ad Telegraf. Starlark aggiungerà ad ogni riga di dati il tag contenente il timestamp di simulazione. Notare che è possibile accedere al servizio di InfluxDB con

  ```
  $ docker compose exec influxdb influx
  $ use influx  # per selezionare il database chiamato "influx"
  ```

e visualizzare le metriche con:

```
$ show measurements
```

o le tabelle intere con i dati già arrivati grazie alle query. Ad esempio

```
$ select * from /L3servingSINR3gpp_[0-9]{5}_cp/
```

mostra le tabelle, provenienti da dati estratti da file cp-cell, che contengono i valori del SINR[3] misurato a livello di rete (L3) per la cella 5G che fornisce un servizio, in una rete cellulare, conforme agli standard 3GPP. Notare l'utilizzo delle regex per considerare tutte le tabelle L3servingSINR3gpp, poiché sono divise per utente. Ogni utente ha infatti come identificatore un numero naturale, perciò la tabella L3servingSINR3gpp_1_cp conterrà i dati della metrica per l'utente 1.

- Grafana, con un refresh di 5s (se impostato) andrà a estrarre con le query, analoghe a quelle viste nel codice JSON di esempio, i dati da InfluxDB per visualizzarli.

In figura 2.7, 2.9 e 2.10 vengono mostrati alcuni grafici provenienti dalla simulazione. Essendo le dashboard di Grafana composte da tanti panel (fino a 50+ per du-cell), non è possibile mostrare tutta la dashboard per intero. Sulla leggenda è possibile cliccare sull'identificativo di un singolo utente per filtrare la query è ottenere il grafico solo di quell'utente, come mostrato in figura 2.8.

Metriche Aggregate:

Come mostrato in figura 2.11 abbiamo configurato una dashboard per contenere grafici di dati aggregati da tutti e tre i tipi di file. Precisamente, questa dashboard presenta:

- un istogramma che mostra in tempo reale il *load* di utenti di ogni cella 5G.
- una heatmap che mostra lo storico del *load* di utenti di ogni cella 5G.

[3] Il SINR (Signal-to-Interference plus Noise Ratio) misura la qualità del segnale wireless. Precisamente $SINR = \frac{potenza\ segnale\ desiderato}{potenza\ interferenza\ +\ potenza\ rumore}$

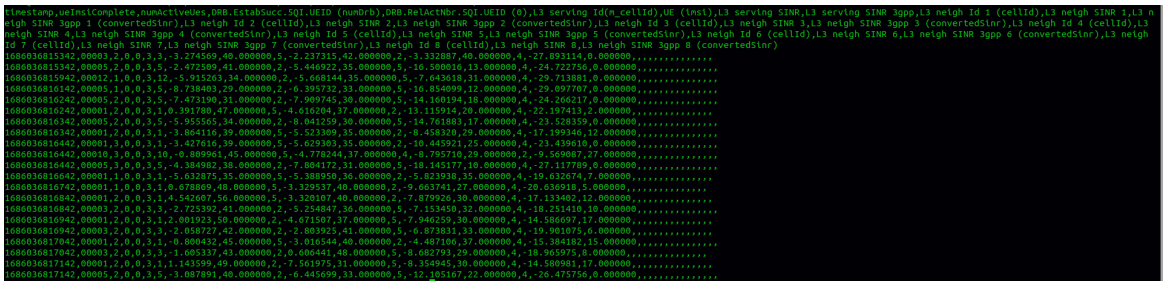


Figura 2.6. Esempio di un csv proveniente da un file up-cell.

Figura 2.7. Visualizzazione di alcune metriche relative ai file up-cell

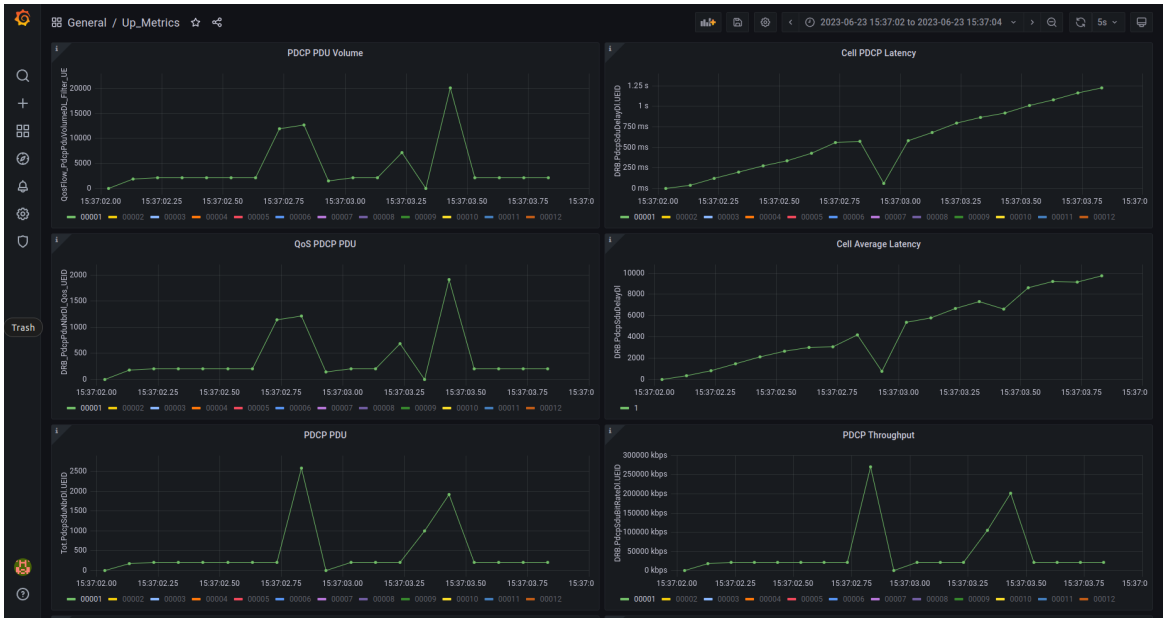


Figura 2.8. Visualizzazione di alcune metriche relative all'utente 1 per i file up-cell

Figura 2.9. Visualizzazione di alcune metriche relative ai file cp-cell

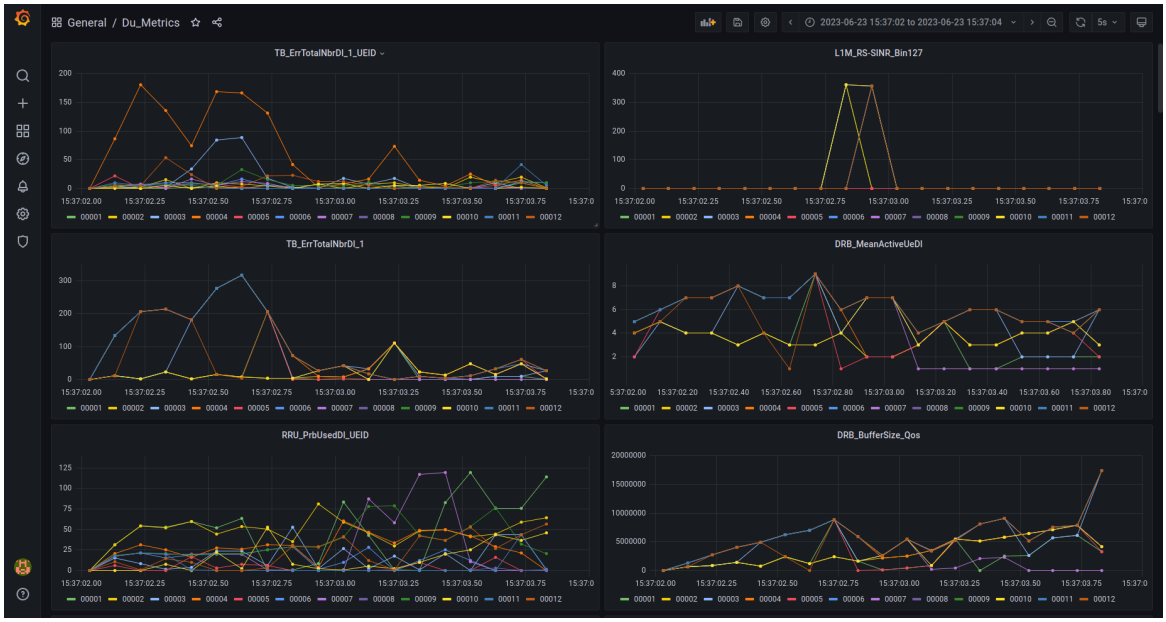


Figura 2.10. Visualizzazione di alcune metriche relative ai file du-cell

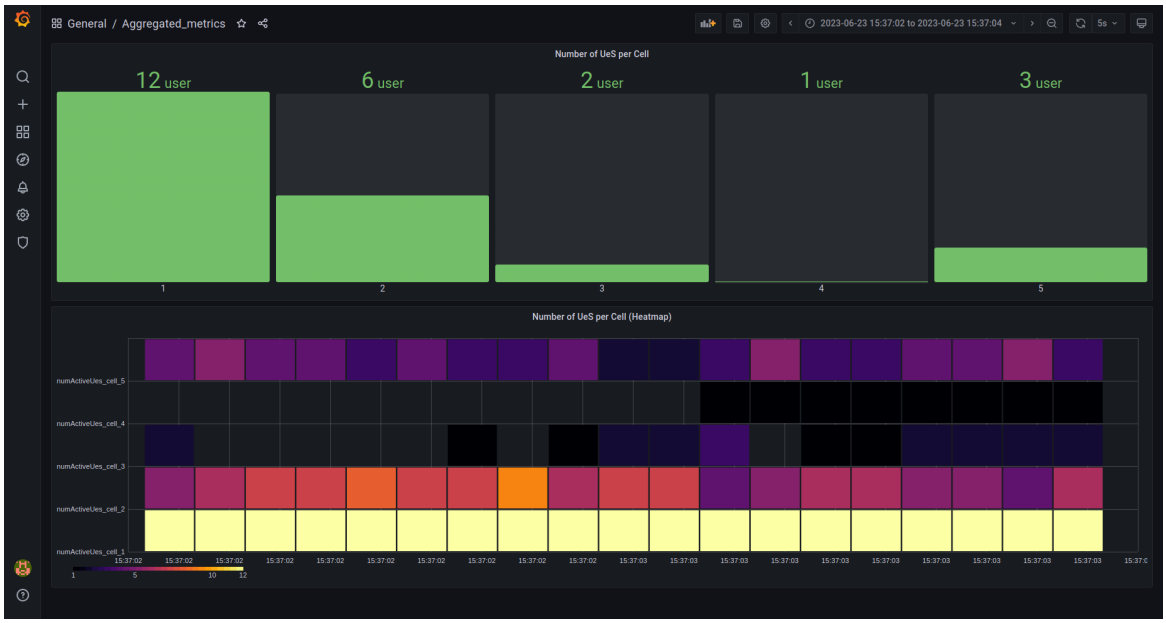


Figura 2.11. Visualizzazione di alcune metriche relative ai file up-cell

Capitolo 3

Creazione della Simulazione: Ricerca in Letteratura

Fino ad ora ci siamo affidati a simulazioni esterne per testare il framework. In questo capitolo si descriverà la creazione della simulazione personalizzata su cui far lavorare la stack. Inizialmente, abbiamo pensato di costruire il modello di canale tramite una ricerca in letteratura. Successivamente, abbiamo trovato più pertinente costruirlo tramite campionamento di un applicazione già esistente. La ricerca in letteratura ha comunque evidenziato delle relazioni e degli standard utili per la nostra simulazione, per cui vale la pena discuterne. Tra i vari paper analizzati, i più rilevanti sono:

- *Performance monitoring of a wireless campus area network* [1]: l'obiettivo del paper era dimostrare la fattibilità e praticità di una Campus Area Network WaveLAN. Per noi l'utilità risiedeva nelle relazioni evidenziate dagli esperimenti.
- *Traffic modelling of smart city internet of things architecture* [2]: propone una tecnica di modellazione del traffico ON/OFF per dispositivi IoT. Lo use case sono delle smart city. Potrebbe esserci utile perché i nostri nodi comunicheranno proprio con un modello ON/OFF. Il modello di traffico è diviso in pattern e dettagliatamente descritto negli appunti.
- *Towards a Smart Environment: Optimization of WLAN Technologies to Enable Concurrent Smart Services* [3]: suggerisce una configurazione WLAN per ambiente smart che sprechi poche risorse. Studia la distribuzione spaziale e l'impatto sulla QoS di 5 tipi differenti di servizi. Il paper propone anche una classifica di rendimento delle varie tecnologie IEEE 802.11 nel contesto.

Un approfondimento su i tre paper verrà ora fornito.

3.1 Relazioni tra Performance e Ostacoli

Da esperimenti reali [1], è possibile estrarre le seguenti relazioni.

3.1.1 Throughput e Potenza:

Con la configurazione presente in Figura 3.1 si prova sperimentalmente

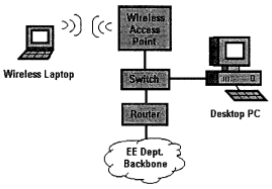


Figura 3.1. setup utilizzato

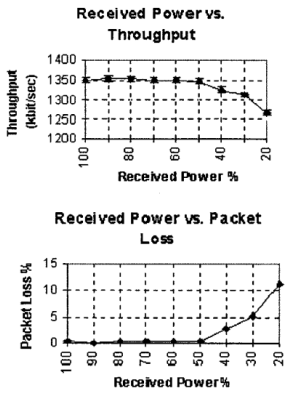


Figura 3.2. relazione tra potenza ricevuta e throughput e tra potenza ricevuta e perdita di pacchetti

che il throughput diminuisce al diminuire della potenza ricevuta, e quindi all'aumentare della distanza del nodo dall'Access Point. Inoltre al diminuire della potenza ricevuta, aumenta il Packet Loss. Per i dispositivi da loro utilizzati[1] si spiega che il "received power" non è una misura della vera potenza, ma una misura più o meno qualitativa del "power percentage". Esempio: se due laptop e l'AP sono molto vicini, avremmo un received power tra il 90 e il 100%, mentre sui bordi dell'area coperta dall'AP avremo il 20%. Le performance del

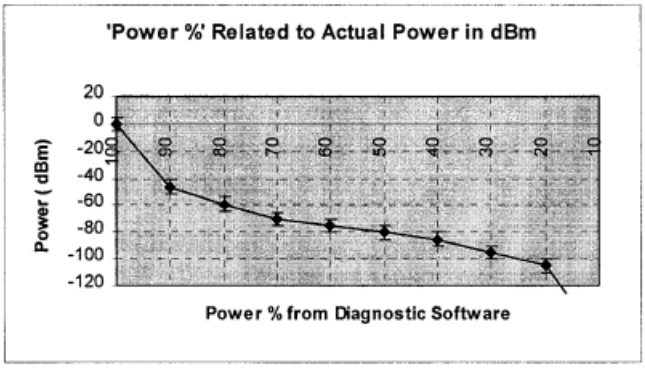


Figura 3.3. Relazione approssimativa tra potenza in percentuale e effettiva potenza in dBm

traffico sono circa costanti fino al 50% di potenza (circa -80 dBm), mentre calano drasticamente dopo. Comunque, anche ai margini (20%) il TH è ancora accettabile.

[1] il paper è del 1997, i laptop usati per l'esperimento non sono potenti come gli odierni.

3.1.2 Effetto dei Muri:

Utilizzando un laboratorio[2], con la disposizione dei nodi mostrata in Figura 3.4, risulta che le performance sono buone fino a 5 muri attraversati, con una piccola perdita lineare del throughput. Precisamente: dopo 5 muri la potenza è calato del 40%. Un sesto muro potrebbe abbassare sotto il 20% e quindi far perdere il segnale. I risultati dell'esperimento sono riassunti in Figura 3.5.

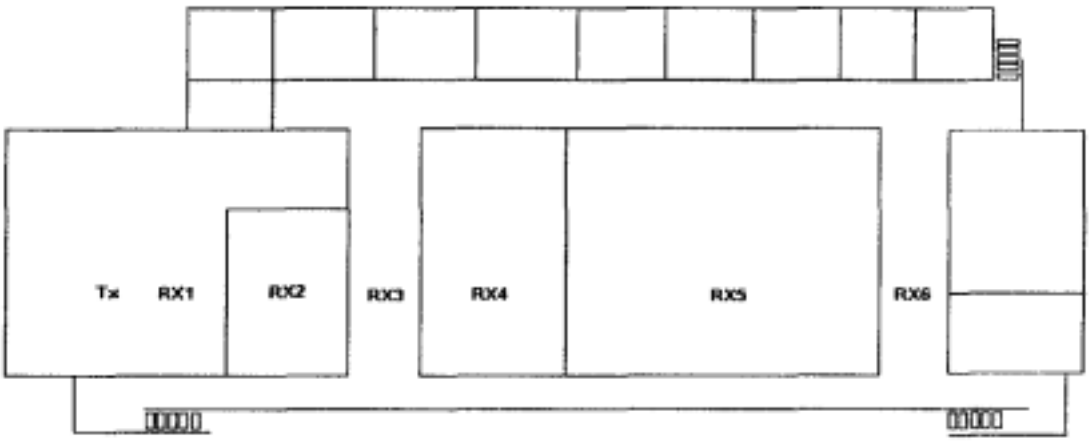


Figura 3.4. Pianta del laboratorio. 'TX' rappresenta l'AP, mentre 'RX' i laptop wireless.

3.1.3 Interferenza tra laptop:

Con 2 laptop nella rete, viene registrato th e packet loss medi; dopo 10 minuti si inserisce un altro laptop e si registra di nuovo th e packet loss. Questo procedimento è stato ripetuto per altri 3 laptop. Anche con 4 laptop (oltre alla coppia iniziale) il packet loss è molto basso, poiché il CSMA/CA sta evitando che avvengano collisioni. Il th decresce fino a 650 kb/s, e la banda non è distribuita in modo uguale in tutti i laptop. Se questo non fosse vero infatti, si avrebbe 650 kb/s * 6 = 3.9 Mb/s, ma il massimo throughput del canale è di 2 Mb/s. Dall'esperimento si verifica una tendenza *first come first serve* per queste applicazioni con stream costante di dati. Comunque è importante considerare che questa distribuzione non uniforme della banda è dovuta al fatto che le applicazioni usate per i testi richiedono tanta banda e inviano stream di dati, mentre nella maggior parte delle applicazioni, come il nostro caso, l'invio è a burst di dati.

[2] i muri del laboratorio usato nell'esperimento [1] sono in cartongesso su legno con telai in travi di acciaio, sono di grandi dimensioni e separati da grande distanza, quindi le specifiche prestazioni risultanti dall'esperimento non sono rilevanti per il nostro modello, che riguarda un ambiente smart di dimensioni modeste

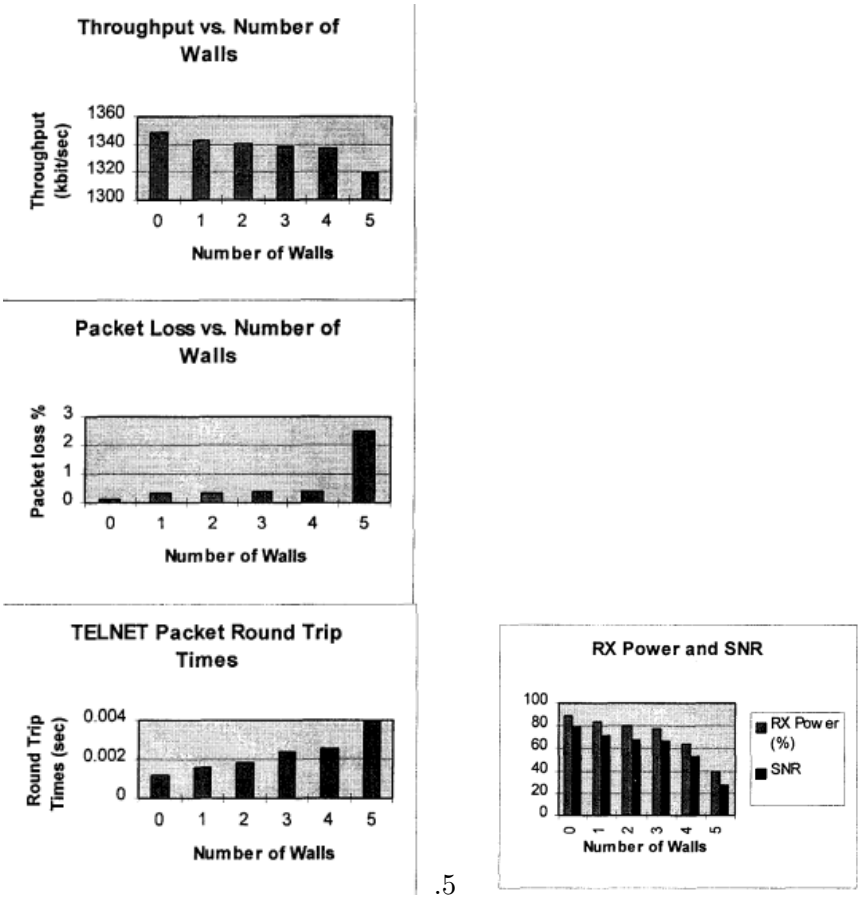


Figura 3.5. riassunto dei risultati ottenuti per gli esperimenti sull'effetto dei muri

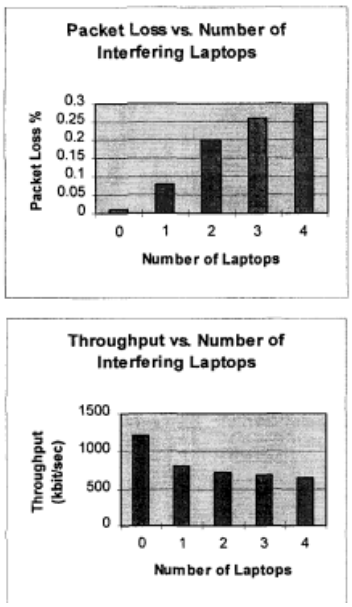


Figura 3.6. Effetto degli ultimi 4 laptop aggiunti alla coppia iniziale

3.2 Modello di Traffico ON/OFF

Proponiamo ora [2] una tecnica di modellazione del traffico ON/OFF di dispositivi IoT migliorata. Lo use case sono delle smart city, modellate come interconnessioni di smart home. Tale modello risulta utile per la costruzione di nodi con modello di comunicazione ON/OFF nel nostro modello di traffico.

3.2.1 5 pattern di traffico

Possiamo dividere in 5 pattern di traffico per l'ON STATE, 2 periodici e 3 aperiodici. Ogni pattern include le seguenti 4 variabili aleatorie da tenere in considerazione:

- *inter-arrival time*: tempo tra un arrivo di un pacchetto e il seguente
- *packet size*: dimensione del pacchetto
- *packet arrival time*: tempo di arrivo di un pacchetto rispetto a quando è stato inviato
- *number of packet arrival*: numero di pacchetti arrivati nell'unità di tempo

Nell'OFF state non c'è trasmissione di pacchetti.

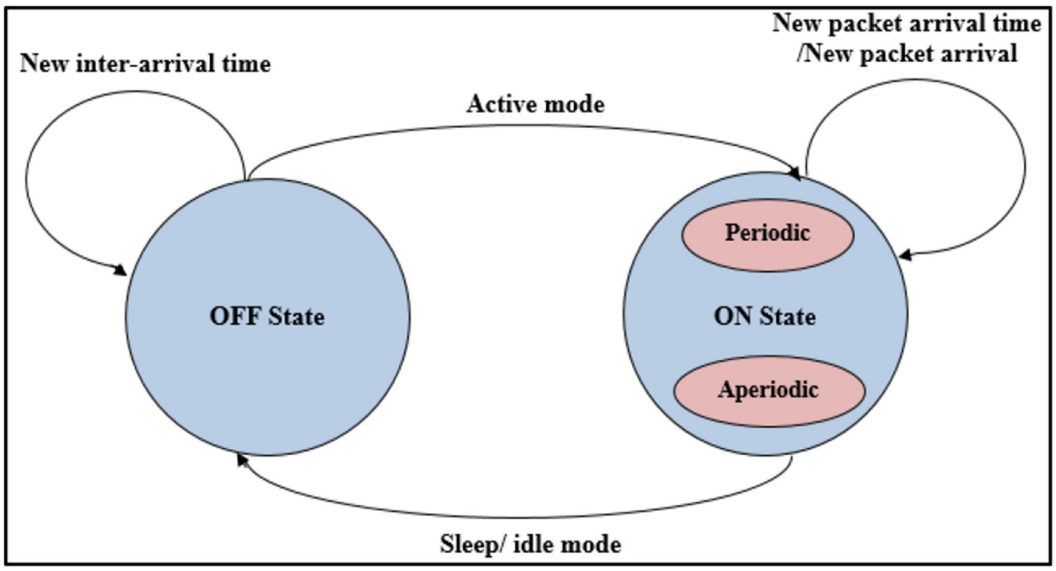


Figura 3.7. Automa degli stati di un nodo con modello di comunicazione ON/OFF

Pattern di Traffico Periodico:

che hanno inter-arrival time costante, quindi i pacchetti sono ricevuti ogni k secondi, dove k è una costante. Si dividono in pattern di traffico periodico:

di dati: in cui i nodi trasmettono dati regolarmente. Si ha traffico *low power consumption*, *high reliability* e non è adatto per il *real time*.

1. packet size packet arrival sono distribuiti uniformemente
2. number of packet arrival segue una distribuzione di Poisson:

$$P_\lambda(n) = \frac{\lambda^n}{n!} e^{-\lambda} \tag{3.1}$$

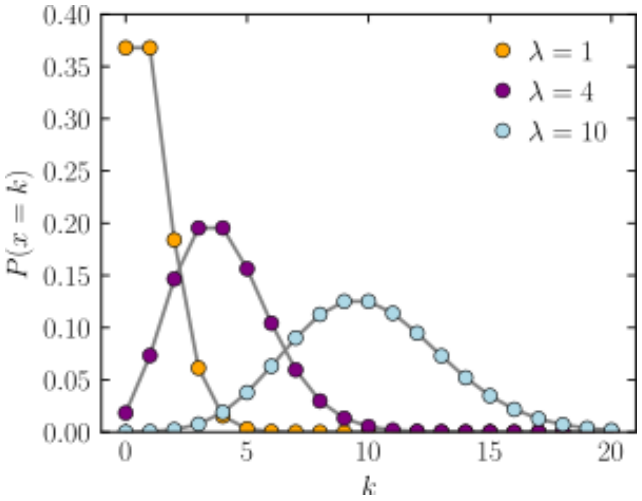


Figura 3.8. Funzione di distribuzione della distribuzione di Poisson

di controllo (Keep-Alive Status): in cui ogni nodo invia K-A MSG ai *peer*. Se questi messaggi non arrivano segnalano un possibile drop o un failure. Questo pattern è evidentemente pensato per il traffico *real time* o urgente. Le distribuzioni statistiche sono le stesse del pattern di traffico periodico di dati.

Pattern di Traffico Aperiodico:

normale: presenta un interarrival moderato (in secondi)

1. packet arrival non omogeneo: distribuzione uniforme
2. packet size: distribuzione uniforme
3. packet number: distribuzione di Poisson
4. interarrival time: distribuzione geometrica. Se infatti pensiamo ai pacchetti come "successi" e ai tempi di inter-arrivo come "tentativi", allora il tempo tra i successivi pacchetti (cioè il tempo di inter-arrivo) può essere considerato come il numero di "tentativi" necessari per ottenere il prossimo "successo". La probabilità che un pacchetto arrivi è p, quindi la probabilità di fallire x tentativi, cioè di aspettare x tempi, prima di ricevere il pacchetto è:

$$P(X = x) = (1 - p)^{x-1} p \tag{3.2}$$

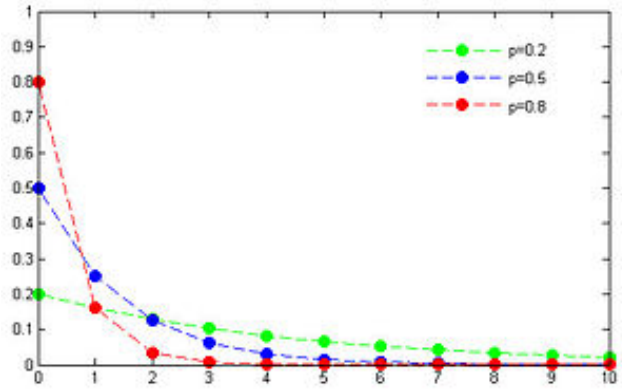


Figura 3.9. Funzione di Distribuzione della distribuzione geometrica

payload exchange: che viene usato per applicazioni real-time. Si ha trasmissione continua durante l'ON period.

1. interarrival time molto lungo (in minuti o ore): viene usata la distribuzione di Birnbaum-Saunders[3]:

$$f(t;\alpha,\beta) = \frac{1}{\sqrt{2\pi}} \frac{\sqrt{\frac{t}{\beta}} + \sqrt{\frac{\beta}{t}}}{2\alpha t} e^{-\frac{(\sqrt{\frac{t}{\beta}} - \sqrt{\frac{\beta}{t}})^2}{2\alpha^2}} \tag{3.3}$$

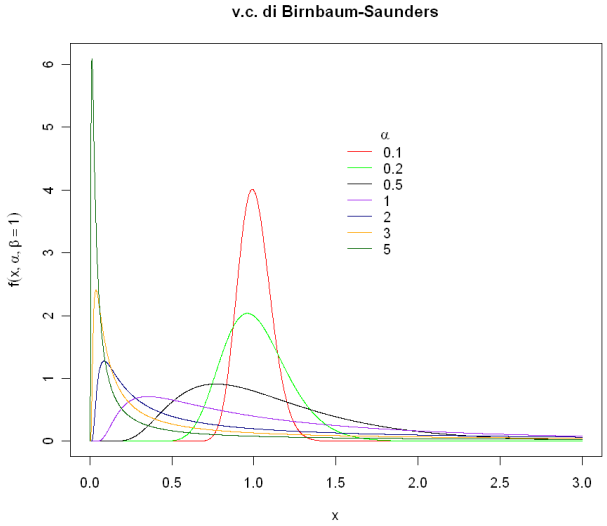


Figura 3.10. funzione di densità di probabilità della distribuzione di Birnbaum-Saunders per beta = 1

2. lunghi ON period (centinaia di secondi) per cui viene usata la distribuzione gamma:

$$f(x) = \frac{1}{\theta^k \Gamma(k)} x^{k-1} e^{-\frac{x}{\theta}} = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \tag{3.4}$$

[3]nota anche come la distribuzione della vita a fatica

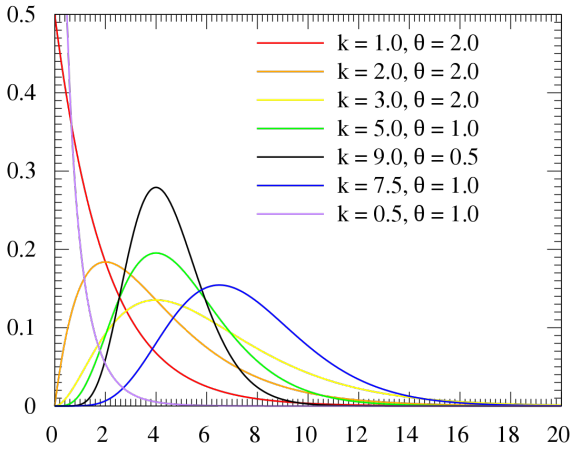


Figura 3.11. Funzione di densità di probabilità della distribuzione Gamma

3. grande numero continuo di pacchetti e di dimensione di pacchetti: distribuzione uniforme.

ED: trasmette poco e consuma poco, utile per il real-time.

1. interarrival molto corto (nell'ordine degli ms). Può essere modellato con la distribuzione di Pareto:

$$f(x) = \begin{cases} \frac{\alpha x_m^\alpha}{x^{\alpha+1}}, & \text{se } x \geq x_m \\ 0, & \text{altrimenti} \end{cases}$$

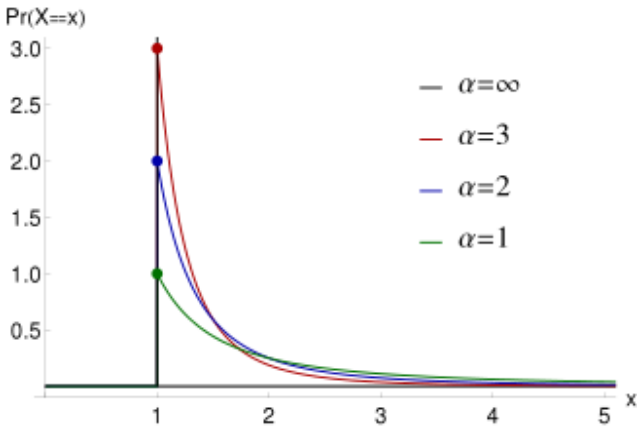


Figura 3.12. Funzione di densità di probabilità di Pareto

2. Ha ON period molto corti. Modellato con la distribuzione di Weibull:

$$f(x) = \frac{k}{\lambda^k} x^{k-1} e^{-(\frac{x}{\lambda})^k} \tag{3.5}$$

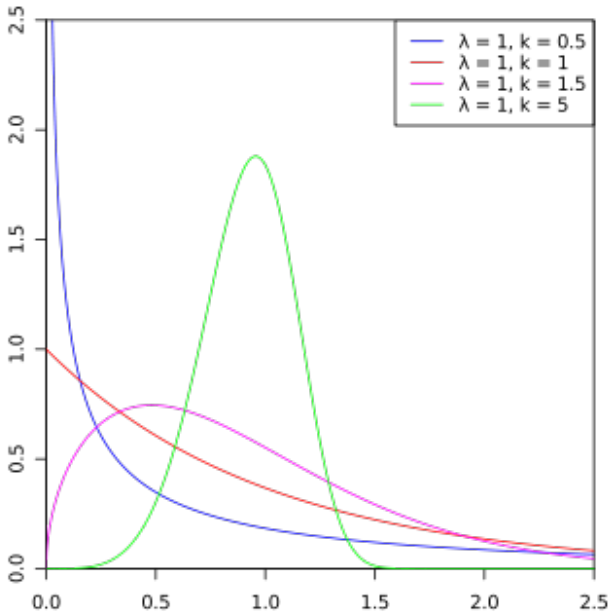


Figura 3.13. Funzione di densità di probabilità di Weibull

3. number of packet piccolo, modellato con Poisson.
4. packet arrival random e con distribuzione discreta uniforme.

3.3 Smart Environment ed Efficienza delle Risorse

Proponiamo ora uno studio [3] tramite una simulazione che suggerisce una configurazione di rete WLAN di un ambiente smart che sprechi poche risorse. Si studia la distribuzione spaziale (circolare, random e uniforme) e l'impatto sulla QoS di 5 tipi differenti di servizi: Voice over Internet Protocol (VoIP), Video Conferencing (VC), HTTP e mail elettronica, tutti attivati simultaneamente a diverse percentuali di attività. Il paper [3] propone anche una classifica di rendimento delle varie teconologie IEEE 802.11. I parametri utilizzati per misurare la qualità del servizio sono Packet Loss, Jitter, Throughput e Delay. Il numero di nodi va da 1 a 65 e gli standard analizzati sono 802.11, 802.11a, 802.11b, 802.11g, 802.11e, 802.11n. La simulazione è svolta con OPNET.

3.3.1 Parametri di Qos

1. Latency: tempo in secondi per viaggiare da un nodo A ad un nodo B della rete
2. Jitter (in secondi): variazione della latenza causata da accodamento

3. Throughput (bps): rate al quale i pacchetti sono mandati da un punto all'altro durante un intervallo di tempo

4. Packet Loss: percentuale di pacchetti persi lungo il canale di trasmissione dopo che l'utente li ha già mandati

5. Coefficiente ICP: "coefficiente di importanza" assegnato ad ogni parametro dell'applicazione. Misura l'effetto che un certo parametro avrà sulla qualità del servizio. In figura 3.14 è presente una tabella che mostra i coefficienti di importanza attribuiti ai parametri delle varie applicazioni considerate: **??**

Application	Importance (I) and Threshold (T)	Jitter (sec)	Packet Loss Rate (%)	Delay (sec)	Throughput (kbps)
E-mail	I	Very Low	Low	Low	Low
	T	0	10	1	30
HTTP	I	Very Low	Low	Medium	Low
	T	0	10	1	30
VC	I	High	Medium	High	High
	T	0.03	1	0.15	250
FTP	I	Very Low	High	Low	Medium
	T	0	5	1	45
VoIP	I	High	Low	High	Medium
	T	0.04	5	0.15	45

Figura 3.14. Coefficiente ICP

3.3.2 Configurazione

Nella simulazione le applicazioni real-time sono distribuite con un 50% VoIP e 50% VC. Quelle best-effort sono distribuite per un 30% e-mail, 30% FTP e 40% HTTP. Le 3 architetture della rete proposte sono riportate in figura 3.15. Le architetture BSS e ESS hanno un access point, mentre la IBSS è ad hoc. La simulazione è stata testata in uno spazio di 2/3 x 10/14 metri con una durata di 20 minuti. Il traffico delle applicazioni è stato configurato con i seguenti parametri:

1. inter-arrival time per le applicazioni real-time: 15 frame al secondo

2. frame size per le applicazioni real-time: 128x240/128x240 pixels (bytes)

3. dimensione massima pacchetto FTP: 50KB

4. dimensione massima file e-mail: 20KB

Per determinare il setup migliore viene utilizzato un algoritmo che prende in input la cumulative distribution function (CDF) e i threshold di QoS precedentemente menzionati (Per brevità non riporto il funzionamento dell'algoritmo). In figura 3.16 sono riportati i parametri di simulazione per ogni applicazione.

3.3.3 Analisi delle performance

Nell'esperimento si è analizzato il rendimento delle varie applicazioni nelle 3 diverse distribuzioni spaziali: circolare (C), random (R) e uniforme (U). I

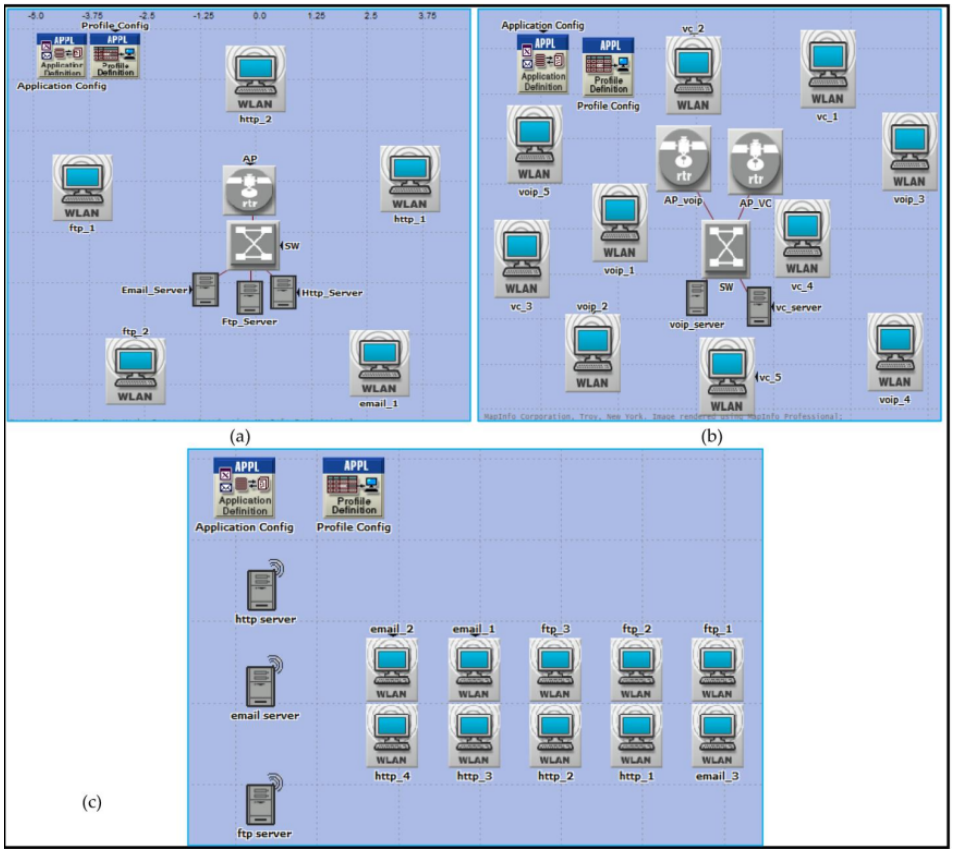


Figura 3.15. Architetture simulate

diagrammi seguenti mostrano i risultati con la tecnologia IEEE consigliata (evidenziata in rosso) ordinata per numero di nodi e distribuzione spaziale. Il primo riguarda le architetture di rete con l'access point (BSS e ESS), l'altro l'architettura ad hoc (IBSS). Entrambe le tecnologie 11e e 11n hanno prodotto performance quasi ottimali per tutte le distribuzioni spaziali, simile ad IBSS, con la differenza che le distribuzioni uniformi e random sono dominanti per un numero medio-grande di nodi. Un altro dato è che le performance delle architetture ESS e BSS per ogni gruppo di nodi è più o meno la stessa, entrambi gli approcci generano una performance quasi ottimale per ogni distribuzione spaziale. Dai risultati si evince che le uniche tecnologie che lavorano in maniera ottimale nelle architetture uniformi e circolari sono le IEEE 802.11 e e 11.n. Questi risultati possono essere utili per ottimizzare il rendimento del nostro ambiente smart utilizzando le tecnologie appropriate.

System Settings			
1	Profile start time (sec)	60	
2	Simulation time (min)	20	
3	Value Per Statistic	200	
4	IP Routing	EIGRP Enable	
5	VC	Parameters	Values
		Frame Interarrival Time Information	10–15 frames/sec
		Symbolic Destination Name	Video Destination
		Frame Size Information (bytes)	128 × 120/128 × 240 pixels
		Type of service (TOS)	Interactive multimedia
6	VoIP	Parameters	Values
		Voice frame per packet	1
		Application	Voice
		Codec	G.711
		Compression and Decompression delay	0.02 sec
		Types of service (TOS)	Interactive voice
7	HTTP	Parameters	Values
		HTTP Specification	HTTP 1.1
		Page Interval Time (sec)	Exponential (60)
		Types of service (TOS)	Best Effort
8	FTP	Parameters	Values
		Command Mix (Get/Total)	50%
		Inter-Request Time (sec)	Exponential (360)
		File Size (bytes)	50,000
		Types of service (TOS)	Best Effort
9	Email	Parameters	Values
		Send Interarrival Time (sec)	Exponential (360)
		Receive Interarrival Time (sec)	Exponential (360)
		E-Mail Size (bytes)	20,000
		Symbolic Server Name	Email Server
		Types of service (TOS)	Best Effort

Figura 3.16. parametri di sumulazione

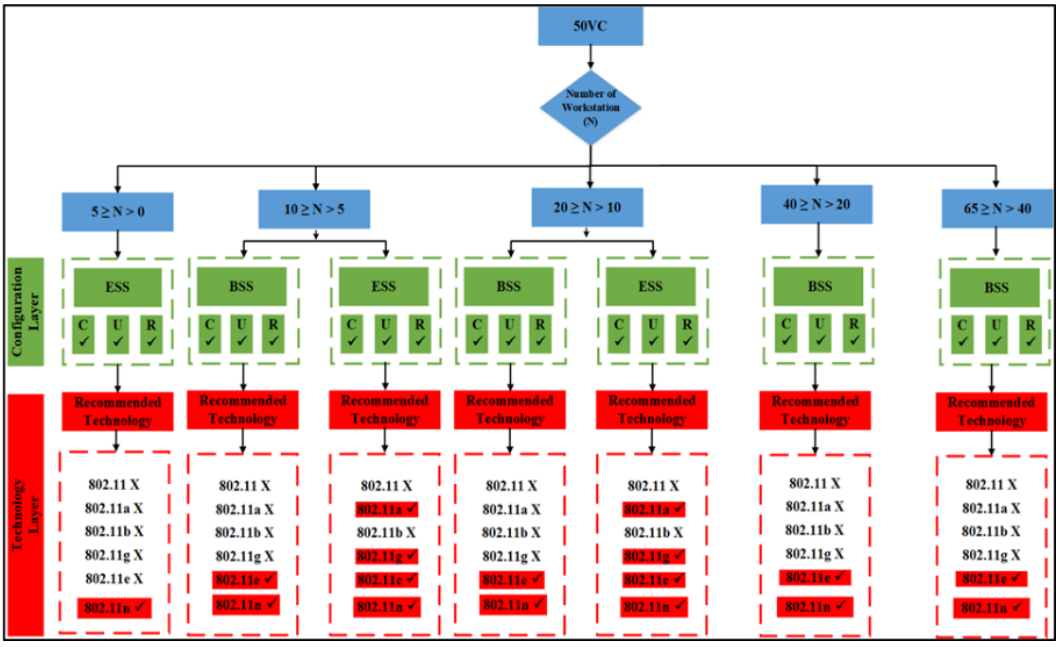


Figura 3.17. BSS e ESS

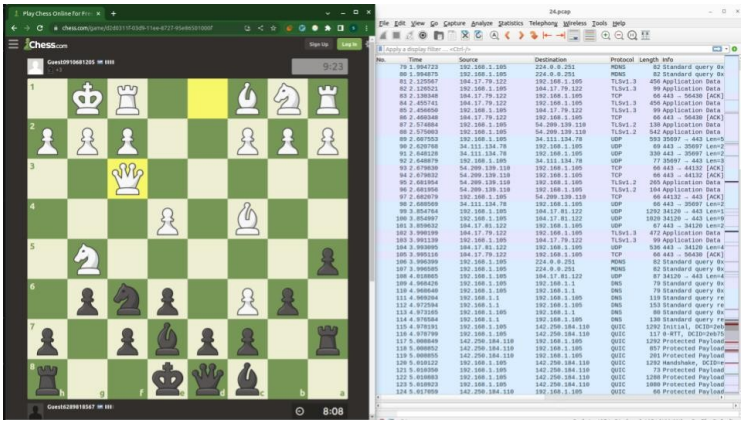
Figura 3.18. Solo IBSS

Capitolo 4

Creazione della Simulazione: Campionamento di un applicazione

In questo capitolo viene descritto l'approccio utilizzato per la creazione della simulazione. Il campionamento e l'analisi del traffico di chess.com ha permesso di costruire un modello di canale semplice ma funzionale per i nostri obiettivi di raccolta dati.

4.1 Campionamento con Wireshark



Connesso a una rete Ethernet, ho fatto 30 catture da partite *bullet* di chess.com tramite Wireshark. Una partita *bullet* è una partita online in cui ogni giocatore ha 1 minuto di tempo, dopo di cui perde automaticamente la partita. Questo ha permesso di limitare la durata delle catture a massimo 2 minuti. È importante notare che, data la sua freneticità, questa modalità di gioco porta a un traffico leggermente differente dal traffico di una classica partita di scacchi, come evidenziato dai grafici I/O di Wireshark successivamente mostrati. Questo perché una partita con una durata maggiore permette al giocatore di pensare alla mossa da fare e agire lentamente, rendendo quindi più evidenti gli spike periodici intervallati da momenti di "silenzio".

4.2 Estrazione con Pyshark

```
import pyshark
import datetime
from math import *
import pandas as pd
```

Per raccogliere le prestazioni estratte dai singoli pacchetti sono stati usati due *dataframe pandas*. Un dataframe conteneva i dati dei pacchetti trasmessi dalla rete privata al server di chess.com e l'altro i dati dei pacchetti trasmessi in direzione opposta. Entrambi i dataframe presentano le seguenti colonne, che sono le metriche di interesse:

- 'id cap'
- 'npack spike'
- 'sizepack spike'
- 'freq spike'
- 'npack game'
- 'sizepack game'
- 'freq game'

```
for i in range(1, 31):

    n_file = str(i) + ".pcap"
    cap = pyshark.FileCapture(n_file)

    num_pack_src_spike = 0
    size_pack_src_spike = 0
    num_pack_src_game = 0
    size_pack_src_game = 0
    num_pack_dst_spike = 0
    size_pack_dst_spike = 0
    num_pack_dst_game = 0
    size_pack_dst_game = 0

    first_timestamp = -1
    last_timestamp = -1
```

Iteriamo sui 30 pacchetti, aprendone uno alla volta

```
    for packet in cap:

        if 'ip' in packet:

            ip_src = packet.ip.src
            ip_dst = packet.ip.dst

            if '192.0.0.0' < ip_src < '193.0.0.0':

                if '104.0.0.0' < ip_dst < '105.0.0.0':

```

```
12                      timestamp = packet.sniff_time
13
14                      if (num_pack_src_spike == 0):
15                          first_timestamp = timestamp
16
17                      differenza = timestamp - first_timestamp
18
19                      due_sec = datetime.timedelta(seconds=2)
20                      if (differenza < due_sec):
21                          num_pack_src_spike += 1
22                          size_pack_src_spike +=
                                int(packet.length)
23                      else:
24                          num_pack_src_game += 1
25                          size_pack_src_game +=
                                int(packet.length)
26
27              if '192.0.0.0' < ip_dst < '193.0.0.0':
28
29                  if '104.0.0.0' < ip_src < '105.0.0.0':
30
31                      timestamp = packet.sniff_time
32
33                      if (num_pack_dst_spike == 0):
34                          first_timestamp = timestamp
35
36                      differenza = timestamp - first_timestamp
37
38                      due_sec = datetime.timedelta(seconds=2)
39                      if (differenza < due_sec):
40                          num_pack_dst_spike += 1
41                          size_pack_dst_spike +=
                                int(packet.length)
42                      else:
43                          num_pack_dst_game += 1
44                          size_pack_dst_game +=
                                int(packet.length)
45
46          last_timestamp = packet.sniff_time
```

All'interno del loop, vengono controllati i pacchetti che contengono l'header IP. Viene estratto l'indirizzo IP di origine e l'indirizzo IP di destinazione dal pacchetto corrente e viene verificato se l'indirizzo IP di origine o destinazione rientra nell'intervallo tra "192.0.0.0" e "193.0.0.0", e se l'indirizzo IP di origine o destinazione rientra nell'intervallo tra "104.0.0.0" e "105.0.0.0". Questa condizione filtra il traffico che interessa all'analisi. Notare che se num_pack_src_spike è 0, viene impostato first_timestamp con il valore corrente di timestamp. Questo serve a individuare il primo pacchetto dello spike. Successivamente viene calcolata la differenza tra timestamp e first_timestamp per determinare da quanto tempo è stato catturato il pacchetto corrente e viene controllato se la differenza tra timestamp e first_timestamp è inferiore a 2 secondi. Se la condizione è vera, il pacchetto viene considerato parte dello spike e le relative metriche vengono aggiornate. Se il pacchetto non è parte dello spike il pacchetto è parte del "game". Viene aggiornato continuamente il timestamp finale (last_timestamp) con il valore di timestamp dell'ultimo pacchetto nel file pcap corrente. In questo modo alla fine del pacchetto avremo che last_timestamp

contiene effettivamente l'ultimo timestamp del pacchetto, con cui riusciamo a calcolare la durata della partita (duration).

```
    duration = last_timestamp - first_timestamp

    secondi_totali = duration.total_seconds()
```

Si stampano poi le prestazioni per il singolo pcap e si chiude il pcap per passare al prossimo

```
    print("\t pcap:" + n_file)
    print("\t num_pack_src_spike:" + str(num_pack_src_spike)
        + ' sizepack_src_spike:' + str(size_pack_src_spike /
        num_pack_src_spike) + " num_pack_src_game:" +
        str(num_pack_src_game) + ' sizepack_game:' +
        str(size_pack_src_game / num_pack_src_game) + "
        num_pack_dst_spike:" + str(num_pack_dst_spike) +
        'sizepack_spike' + str(size_pack_dst_spike /
        num_pack_dst_spike) +
    " num_pack_dst_game:" + str(num_pack_dst_game) + '
        sizepack_game:' + str(size_pack_dst_game /
        num_pack_dst_game) + " secondi totali:" +
        str(secondi_totali))
    print("\n\n")

    data = {'id_cap': i, 'npack_spike': num_pack_src_spike,
        'sizepack_spike': size_pack_src_spike /
        num_pack_src_spike, 'freq_spike': num_pack_src_spike /
        2, 'npack_game': num_pack_src_game, 'sizepack_game':
        size_pack_src_game / num_pack_src_game, 'freq_game':
        num_pack_src_game / secondi_totali}
    df_src = pd.concat([df_src, pd.DataFrame(data,
        index=[0])], ignore_index=True)
    data = {'id_cap': i, 'npack_spike': num_pack_dst_spike,
        'sizepack_spike': size_pack_dst_spike /
        num_pack_dst_spike, 'freq_spike': num_pack_dst_spike /
        2, 'npack_game': num_pack_dst_game, 'sizepack_game':
        size_pack_dst_game / num_pack_dst_game, 'freq_game':
        num_pack_dst_game / secondi_totali}
    df_dst = pd.concat([df_dst, pd.DataFrame(data,
        index=[0])], ignore_index=True)

    cap.close()
```

Infine, si calcolano le prestazioni medie per le metriche

```


src_npack_spike_mean = df_src['npack_spike'].mean()
src_sizepack_spike_mean = df_src['sizepack_spike'].mean()
src_freq_spike_mean = df_src['freq_spike'].mean()

src_npack_game_mean = df_src['npack_game'].mean()
src_sizepack_game_mean = df_src['sizepack_game'].mean()
src_freq_game_mean = df_src['freq_game'].mean()

# meta' dello spike dell'inizio
src_npack_end_mean = df_src['npack_spike'].mean() / 2
src_sizepack_end_mean = df_src['sizepack_spike'].mean() / 2
src_freq_end_mean = df_src['freq_spike'].mean() / 2
```

```
15
16 dst_npack_spike_mean = df_dst['npack_spike'].mean()
17 dst_sizepack_spike_mean = df_dst['sizepack_spike'].mean()
18 dst_freq_spike_mean = df_dst['freq_spike'].mean()
19
20 dst_npack_game_mean = df_dst['npack_game'].mean()
21 dst_sizepack_game_mean = df_dst['sizepack_game'].mean()
22 dst_freq_game_mean = df_dst['freq_game'].mean()
23
24 # meta' dello spike dell'inizio
25 dst_npack_end_mean = df_dst['npack_spike'].mean() / 2
26 dst_sizepack_end_mean = df_dst['sizepack_spike'].mean() / 2
27 dst_freq_end_mean = df_dst['freq_spike'].mean() / 2
28
29 print("\n\n\
30  \n numero pacchetti medio spike iniziale (src): " +
       str(src_npack_spike_mean) +
31 "\n dimensione media pacchetti spike iniziale (src): " +
     str(src_sizepack_spike_mean) + " bytes" +
32 "\n frequenza media pacchetti spike iniziale (src): " +
     str(src_freq_spike_mean) + " pacchetti al secondo" +
33
34 "\n numero pacchetti medio game (src): " +
     str(src_npack_game_mean) +
35 "\n dimensione media pacchetti game (src): " +
     str(src_sizepack_game_mean) + " bytes" +
36 "\n frequenza media pacchetti game (src): " +
     str(src_freq_game_mean) + " pacchetti al secondo" +
37
38 "\n numero pacchetti medio end (src): " +
     str(src_npack_end_mean) +
39 "\n dimensione media pacchetti end (src): " +
     str(src_sizepack_end_mean) +
40 "\n frequenza media pacchetti end (src): " +
     str(src_freq_end_mean) +
41
42 "\n numero pacchetti medio spike iniziale (dst): " +
     str(dst_npack_spike_mean) +
43 "\n dimensione media pacchetti spike iniziale (dst): " +
     str(dst_sizepack_spike_mean) + " bytes" +
44 "\n frequenza media pacchetti spike iniziale (dst): " +
     str(dst_freq_spike_mean) + " pacchetti al secondo" +
45
46 "\n numero pacchetti medio game (dst): " +
     str(dst_npack_game_mean) +
47 "\n dimensione media pacchetti game (dst): " +
     str(dst_sizepack_game_mean) + " bytes" +
48 "\n frequenza media pacchetti game (dst): " +
     str(dst_freq_game_mean) + " pacchetti al secondo" +
49
50 "\n numero pacchetti medio end (dst): " +
     str(dst_npack_end_mean) +
51 "\n dimensione media pacchetti end (dst): " +
     str(dst_sizepack_end_mean) +
52 "\n frequenza media pacchetti end (dst): " +
     str(dst_freq_end_mean) +
53 "\n\n")
```

Queste prestazioni possono essere poi passate manualmente o tramite script

alla simulazione ns3. Bisogna però assicurarsi di averle prima approssimate nel modo corretto: il numero di pacchetti è conveniente approssimarlo sempre per eccesso, mentre per le altre due metriche, essendo medie, si può adottare la classica approssimazione dei numeri decimali.

Capitolo 5

Simulazione ns-3

5.1 Struttura della Rete:

La struttura della rete è rappresentata in figura 5.2. I due client sono nella stessa WLAN, connessa con un collegamento *point to point* a una rete ethernet con un dispositivo in fibra e il Server. Notare come i due client non comunichino tra di loro ma inviano e ricevono dal server, proprio come in chess.com il bianco e il nero non comunicano le mosse direttamente all'avversario, ma le comunicano al server che le comunica all'avversario.

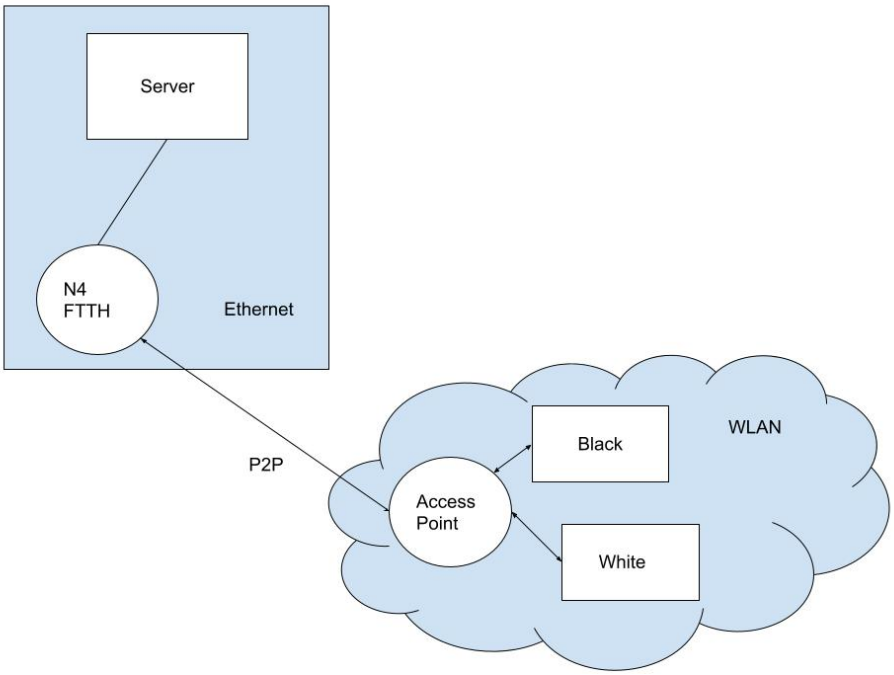


Figura 5.1. Diagramma della rete implementata in ns-3

Server e Client hanno gli stessi 5 stati: Idle, Associazione, Inizio, Game, Fine. Si inizia da Idle e si torna a Idle dopo la fine della partita. Nella fase di associazione il server deve occuparsi dell'associazione di due client quando questi hanno già fatto richiesta ad esso. Passare da uno stato all'altro necessita

di un certo numero di pacchetti inviati e ricevuti, che abbiamo calcolato nella sezione 4.2.

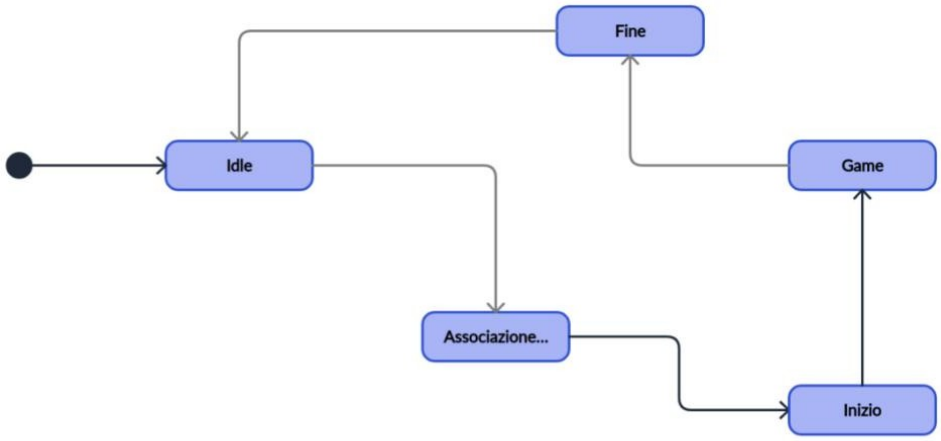


Figura 5.2. Automa degli stati di Client e Server

5.2 Implementazione:

Vediamo ora l'implementazione in ns-3:

```
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/csma-module.h"
#include "ns3/netanim-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/ssid.h"
#include "ns3/mobility-module.h"
#include "ns3/socket.h"
#include "ns3/ipv4-address.h"
```

Dopo aver incluso le librerie necessarie per la simulazione di NS-3, viene definita la classe MyApp che eredita dalla classe Application di NS-3. Questa classe viene utilizzata per creare un'applicazione personalizzata che invia pacchetti attraverso una socket di rete. MyApp ha diversi metodi per configurare e gestire l'applicazione, come il metodo Setup per impostare la socket e i parametri dei pacchetti, il metodo StartApplication per iniziare l'applicazione, il metodo StopApplication per terminare l'applicazione e il metodo SendPacket per inviare i pacchetti.

```

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("ChessSim");
```

```
5
6  typedef enum Status {SERVER_IDLE, CONNECTION, START, GAME,
       END} Status;
7  typedef enum Source {WHITE, BLACK, SERVER} Source;
8
9  class MyApp : public Application
10 {
11 public:
12
13   MyApp ();
14   virtual ~MyApp();
15
16   void Setup (Ptr<Socket> socket, Ipv4Address address,
         Ipv4Address peer1, Ipv4Address peer2, Source source,
         Address white, Address black);
17
18 private:
19   virtual void StartApplication (void);
20   virtual void StopApplication (void);
21
22   void SendPacket (int packetSize);
23   void Connection (void);
24   void Start (void);
25   void Game (void);
26   void End (void);
27   void Idle (void);
28   void Server (void);
29
30
31   Ptr<Socket>     m_socket;
32   Ipv4Address     m_peer1;
33   Ipv4Address     m_peer2;
34   Address         white_addr;
35   Address         black_addr;
36   EventId         m_sendEvent;
37   bool            m_running;
38   uint32_t        m_packetsSent;
39   Status          status;
40   Source          m_source;
41   uint32_t        startPacketsSent;
42   uint32_t        gamePacketsSent;
43   uint32_t        endPacketsSent;
44 };
45
46 MyApp::MyApp ()
47   : m_socket (0),
48     m_peer1 (),
49     m_peer2 (),
50     white_addr (),
51     black_addr (),
52     m_sendEvent (),
53     m_running (false),
54     m_packetsSent (0),
55     status (SERVER_IDLE),
56     startPacketsSent (0),
57     gamePacketsSent (0),
58     endPacketsSent (0)
59 {
```

```
}

MyApp::~MyApp()
{
  m_socket = 0;
}
```

Il metodo Setup configura l'applicazione. Prende in input un puntatore a una socket, indirizzi IP e informazioni sulla sorgente (source) dell'applicazione e li inizializza.

```
void
MyApp::Setup (Ptr<Socket> socket, Ipv4Address address,
    Ipv4Address peer1, Ipv4Address peer2, Source source,
    Address white, Address black)
{
  printf("Setting up application\n");
  m_socket = socket;
  m_source = source;
  white_addr = white;
  black_addr = black;
  std::cout << address << std::endl;
  int ret = m_socket->Bind (InetSocketAddress(address, 0));
  if(ret == -1) exit(EXIT_FAILURE);
  printf("%d\n", ret);
  m_peer1 = peer1;
  m_peer2 = peer2;
}
```

Il metodo StartApplication è chiamato quando l'applicazione viene avviata. Questo metodo contiene la logica principale dell'applicazione e gestisce il suo stato e il flusso di esecuzione. All'interno del metodo StartApplication, vengono eseguite diverse azioni in base allo stato corrente dell'applicazione, rappresentato dalla variabile status. Di seguito è descritto il flusso di esecuzione generale del metodo:

1. L'applicazione verifica se è il server o un client basandosi sulla sorgente (source) specificata durante la chiamata al metodo Setup.
   - Se l'applicazione è il server, viene eseguito il metodo Server, che gestisce la logica del server.
   - Se l'applicazione è un client, viene effettuata la connessione tramite la socket al peer1 (server) tramite la chiamata al metodo Connect.
2. Dopo la connessione, vengono eseguiti i seguenti stati in sequenza: START, GAME e END, ognuno dei quali ha il proprio metodo associato che definisce la logica specifica per ciascuno di essi.
3. Una volta completati gli stati, l'applicazione termina e il metodo StartApplication restituisce il controllo.

```
void
MyApp::StartApplication (void)
{

  printf("Application started\n");
```

```
  if(m_source == SERVER){
    printf("Proprio il server\n");
    Server();
  }
  printf("Invece no\n");
  int ret = m_socket->Connect (InetSocketAddress(m_peer1,
     0)); //connect socket to server
  printf("Ret connect = %d\n", ret);

  printf("Inizio gli stati\n");
  status = START;
  Start();

  status = GAME;
  Game();

  status = END;
  End();

  printf("Finished!\n");
}
```

Il metodo StopApplication interrompe l'esecuzione dell'applicazione, cancella eventuali eventi pianificati e chiude la socket.

```
void
MyApp::StopApplication (void)
{
  m_running = false;
  m_socket->Close ();
}
```

Il metodo SendPacket viene utilizzati per inviare i pacchetti. SendPacket crea un pacchetto di dimensione specificata e lo invia tramite la socket.

```
void
MyApp::SendPacket (int packetSize)
{
  Ptr<Packet> packet = Create<Packet> (packetSize);
  int ret = m_socket->Send (packet);
  printf("Sent %d bytes\n", ret);
}
```

Il metodo Server rappresenta l'implementazione del comportamento del server nell'applicazione. La logica del server consiste nel ricevere pacchetti dai client (indirettamente) e inoltrare i pacchetti ad essi.

```
void
MyApp::Server (void){
  printf("Server started, Initializing\n");
  m_running = true;
  while(m_running){
    Address peerAddress = m_peer1;
    Ptr<Packet> packet1 = m_socket->Recv ();
    if(packet1){
      printf("Ricevuto\n");
      m_socket->Connect (InetSocketAddress(m_peer2, 0));
      m_socket->Send (packet1);
    }
```

```
13      peerAddress = m_peer2;
14      Ptr<Packet> packet2 = m_socket->Recv ();
15      if(packet2 != NULL){
16        printf("Ricevuto\n");
17        m_socket->Connect (InetSocketAddress(m_peer1, 0));
18        m_socket->Send(packet2);
19      }
20    }
21  }
```

Vediamo ora i metodi che implementano la logica dei 5 stati in cui server e client possono trovarsi:

1. Idle
   - Questo metodo rappresenta lo stato di attesa dell'applicazione.
   - All'interno del metodo, viene eseguito un ciclo while finché non viene ricevuto un pacchetto.
   - Utilizzando il metodo Recv() della socket dell'applicazione,
   - l'applicazione tenta di ricevere un pacchetto.
   - Se un pacchetto viene ricevuto (non è NULL), il ciclo while termina.
2. Connection:
   - Questo metodo rappresenta la fase di connessione dell'applicazione.
   - Si manda un pacchetto e si aspetta il suo ritorno per processarlo: può iniziare la partita.
3. Start
   - Questo metodo rappresenta la fase di avvio dell'applicazione.
   - Durante questa fase, vengono inviati pacchetti a intervalli regolari fino a raggiungere un numero specifico di pacchetti inviati.
   - Utilizzando il metodo SendPacket(), vengono inviati pacchetti con dimensioni diverse in base alla sorgente (source) dell'applicazione.
   - Il numero di pacchetti inviati viene controllato e incrementato fino a raggiungere un valore specifico.
4. Game
   - Questo metodo rappresenta la fase di gioco dell'applicazione.
   - Durante questa fase, vengono inviati pacchetti a intervalli regolari fino a raggiungere un numero specifico di pacchetti inviati.
   - La logica specifica del gioco e delle dimensioni dei pacchetti inviati dipende dall'implementazione specifica dell'applicazione.
5. End
   - Questo metodo rappresenta la fase di conclusione dell'applicazione.
   - Durante questa fase, vengono inviati pacchetti a intervalli regolari fino a raggiungere un numero specifico di pacchetti inviati.

- Come nelle fasi precedenti, le dimensioni dei pacchetti inviati possono variare in base alla sorgente (source) dell'applicazione.
- Il numero di pacchetti inviati viene controllato e incrementato fino a raggiungere un valore specifico.

```
void
MyApp::Idle (void)
{
  Ptr<Packet> packet;
  while(packet == NULL){
    packet = m_socket->Recv ();
  }
}

void
MyApp::Connection (void)
{
  int packetSize = 512;
  Ipv4Address sourceAddress = m_source == WHITE ? white_addr
     : black_addr;
  SendPacket(packetSize, sourceAddress);
  Ptr<Packet> receivedPacket;
  Address sourceAddress;
  while (!receivedPacket)
  {
    receivedPacket = m_socket->RecvFrom(sourceAddress);
  }
}

void
MyApp::Start (void)
{
  while (startPacketsSent < 32)
  {
    printf("Sending packets: %d\n", startPacketsSent);
    Simulator::Schedule(Seconds(startPacketsSent * (1/16)),
       &MyApp::SendPacket, this, m_source == WHITE ? 512 :
       347);
    ++startPacketsSent;
  }
}

void
MyApp::Game (void)
{
  if (m_source == WHITE){
    while (gamePacketsSent < 176){
      Simulator::Schedule(Seconds(gamePacketsSent * (1/2)),
         &MyApp::SendPacket, this, 274);
      ++gamePacketsSent;
    }
  }
  if(m_source == BLACK){
    while (gamePacketsSent < 219){
      Simulator::Schedule(Seconds(gamePacketsSent * (1/2)),
         &MyApp::SendPacket, this, 217);
      ++gamePacketsSent;
    }
```

```
  }
}

void
MyApp::End (void)
{
 while (endPacketsSent < 16)
  {
    Simulator::Schedule(Seconds(endPacketsSent * (1/8)),
       &MyApp::SendPacket, this, m_source == WHITE ? 259 :
       174);
    ++endPacketsSent;
    printf("Packet end %d\n", endPacketsSent);
  }
}
```

Nella funzione main, vengono analizzati gli argomenti della riga di comando e vengono creati i nodi della simulazione. Vengono creati due nodi client, un nodo access point (AP) e due nodi per la rete CSMA (N4 e Server). Viene poi impostata la configurazione per la simulazione Wi-Fi, inclusi strato fisico e strato di collegamento. Viene impostato uno standard Wi-Fi (802.11g) e vengono definiti i parametri per i dispositivi Wi-Fi dei nodi client e AP. Successivamente, viene impostata una connessione punto-punto e vengoono dati parametri specifici a quest'ultima e alla rete CSMA. Viene quindi installato lo stack di protocolli Internet sui nodi della simulazione e vengono assegnati indirizzi IPv4 ai dispositivi di rete.

```
int
main (int argc, char *argv[])
{
  CommandLine cmd (__FILE__);
  cmd.Parse (argc, argv);

  // Create nodes
  NodeContainer clients;
  clients.Create(2);

  NodeContainer ap;
  ap.Create(1);

  NodeContainer csmaNodes;
  csmaNodes.Create(2);

  // Setup wifi
  YansWifiChannelHelper channel =
     YansWifiChannelHelper::Default();
  YansWifiPhyHelper phy;
  phy.SetChannel(channel.Create());

  WifiHelper wifi;
  wifi.SetStandard(WifiStandard::WIFI_STANDARD_80211g);

  WifiMacHelper macCli;
  WifiMacHelper macAp;

  Ssid ssid = Ssid("Chess");
  macCli.SetType("ns3::StaWifiMac", "Ssid", SsidValue(ssid),
     "ActiveProbing", BooleanValue(false));
```

```
30  macAp.SetType("ns3::ApWifiMac", "Ssid", SsidValue(ssid));
31
32  NetDeviceContainer devicesClients;
33  NetDeviceContainer deviceAp;
34  devicesClients = wifi.Install(phy, macCli, clients);
35  deviceAp = wifi.Install(phy, macAp, ap);
36
37  // Setup p2p link with FTTH parameters
38  PointToPointHelper p2p;
39  p2p.SetDeviceAttribute("DataRate", StringValue("1Gbps"));
40  p2p.SetDeviceAttribute("Mtu", UintegerValue(1500));
41  p2p.SetChannelAttribute ("Delay", TimeValue (Seconds
        (0.001)));
42
43  NodeContainer p2pnodes;
44  p2pnodes.Add(csmaNodes.Get(0));
45  p2pnodes.Add(ap);
46
47  NetDeviceContainer p2pdevices;
48  p2pdevices = p2p.Install(p2pnodes);
49
50  // Setup csma
51  CsmaHelper csma;
52  csma.SetChannelAttribute ("DataRate", DataRateValue
        (DataRate ("100Gb/s")));
53  csma.SetDeviceAttribute ("Mtu", UintegerValue (9000));
54  csma.SetChannelAttribute ("Delay", TimeValue (Seconds
        (0.010)));
55
56  NetDeviceContainer csmaDevice;
57  csmaDevice = csma.Install(csmaNodes);
58
59  // Install internet stack
60  InternetStackHelper internet;
61  internet.Install(clients);
62  internet.Install(ap);
63  internet.Install(csmaNodes);
64
65  // Assign Ipv4 addresses;
66  Ipv4AddressHelper address;
67  address.SetBase("10.0.0.0", "255.255.255.0");
68
69  Ipv4InterfaceContainer ipcl1;
70  ipcl1 = address.Assign(devicesClients);
71
72  Ipv4InterfaceContainer ipAp;
73  ipAp = address.Assign(deviceAp);
74
75  address.SetBase("10.0.2.0", "255.255.255.0");
76
77  Ipv4InterfaceContainer ipP2p;
78  ipP2p = address.Assign(p2pdevices);
79
80  address.SetBase("10.0.1.0", "255.255.255.0");
81
82  Ipv4InterfaceContainer ipServ;
83  ipServ = address.Assign(csmaDevice);
84
```

```
85  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Successivamente creiamo la socket UDP del server e un'istanza dell'applicazione, che configuriamo tramite Setup. Impostiamo poi i temi di inizio e di stop dell'applicazione con SetStartTime e SetStopTime. Per quanto riguarda i client creiamo due socket UDP e due istanze dell'aplicazione, che configuriamo sempre con Setup. Per gestire la mobilità dei nodi usiamo un'istanza di MobilityHelper: impostiamo il modello di mobilità costante per il server e i nodi CSMA e il modello di mobilità "RandomWalk2dMobilityModel" per i client all'interno dei limiti specificati. Viene aggiunta anche un'interfaccia di animazione AnimationInterface per la visualizzazione della simulazione con NetAnim. Preciassmente viene creato un oggetto AnimationInterface utilizzando AnimationInterface anim(fileNameXML). Questo oggetto verrà utilizzato per registrare l'animazione della simulazione in un file XML. Dopodiché eseguiamo la simulazione, impostando prima il tempo di stop a 125 secondi.

```
1   // Application on server
2   Ptr<Socket> socket = Socket::CreateSocket (csmaNodes.Get
        (1), UdpSocketFactory::GetTypeId ());
3   Ptr<MyApp> app0 = CreateObject<MyApp> ();
4
5   app0->Setup (socket, csmaNodes.Get (1)->GetObject<Ipv4>
        ()->GetAddress(1, 0).GetLocal (), clients.Get
        (0)->GetObject<Ipv4> ()->GetAddress(1, 0).GetLocal (),
        clients.Get (1)->GetObject<Ipv4> ()->GetAddress(1,
        0).GetLocal (), SERVER, clients.Get (0)->GetDevice
        (0)->GetAddress (), clients.Get (1)->GetDevice
        (0)->GetAddress ());
6   csmaNodes.Get (1)->AddApplication (app0);
7   app0->SetStartTime (Seconds (1));
8   app0->SetStopTime (Seconds (121));
9
10  // Application on clients
11  Ptr<Socket> socket1 = Socket::CreateSocket (clients.Get
        (0), UdpSocketFactory::GetTypeId ());
12  if(socket1 == NULL) exit(EXIT_FAILURE);
13  Ptr<Socket> socket2 = Socket::CreateSocket (clients.Get
        (1), UdpSocketFactory::GetTypeId ());
14  Ptr<MyApp> app1 = CreateObject<MyApp> ();
15  Ptr<MyApp> app2 = CreateObject<MyApp> ();
16  app1->Setup (socket1, clients.Get (0)->GetObject<Ipv4>
        ()->GetAddress(1, 0).GetLocal (), csmaNodes.Get
        (1)->GetObject<Ipv4> ()->GetAddress(1, 0).GetLocal (),
        clients.Get (1)->GetObject<Ipv4> ()->GetAddress(1,
        0).GetLocal (), WHITE, clients.Get (0)->GetDevice
        (0)->GetAddress (), clients.Get (1)->GetDevice
        (0)->GetAddress ());
17  app2->Setup (socket2, clients.Get (1)->GetObject<Ipv4>
        ()->GetAddress(1, 0).GetLocal (), csmaNodes.Get
        (1)->GetObject<Ipv4> ()->GetAddress(1, 0).GetLocal (),
        clients.Get (0)->GetObject<Ipv4> ()->GetAddress(1,
        0).GetLocal (), BLACK, clients.Get (0)->GetDevice
        (0)->GetAddress (), clients.Get (1)->GetDevice
        (0)->GetAddress ());
18  clients.Get (0)->AddApplication (app1);
19  clients.Get (1)->AddApplication (app2);
20  app2->SetStartTime (Seconds (1));
```

```
21    app2->SetStopTime (Seconds (121));
22    app1->SetStartTime (Seconds (1));
23    app1->SetStopTime (Seconds (121));
24
25
26    // Setup mobility
27    MobilityHelper mobility;
28    mobility.SetPositionAllocator("ns3::GridPositionAllocator",
29                                  "MinX", DoubleValue (0.0),
30                                  "MinY", DoubleValue (0.0),
31                                  "DeltaX", DoubleValue (5.0),
32                                  "DeltaY", DoubleValue (10.0),
33                                  "GridWidth", UintegerValue
                                        (3),
34                                  "LayoutType", StringValue
                                        ("RowFirst"));
35    mobility.SetMobilityModel
36    ("ns3::ConstantPositionMobilityModel");
37    mobility.Install(ap);
38    mobility.Install(csmaNodes);
39
40    mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
          "Bounds", RectangleValue(Rectangle(-90, 90, -90, 90)));
41    mobility.Install(clients);
42
43    std::string fileNameXML = "ChessSim.xml";
44
45    AnimationInterface anim(fileNameXML);
46
47    Simulator::Stop (Seconds (125));
48    Simulator::Run ();
49    Simulator::Destroy ();
50
51    return 0;
52 }
```

La simulazione può essere facilmente estesa per produrre dei file csv su cui eseguire il nostro framework per ottenere, in ultimo e tramite query personalizzate, delle dashboard di Grafana che rendono più semplice l'interpretazione e l'analisi dei dati.

Capitolo 6

Conclusioni

Questo progetto parte dal bisogno di visualizzare in modo intuitivo i dati provenienti da csv prodotti da simulazioni. Il problema dei csv è che non sono facilmente leggibili, soprattutto quando le metriche e le quantità di dati in gioco sono tante, e ancora più difficile è interpretare e analizzare in modo funzionale i dati prodotti. Il framework sviluppato rende possibile una facile lettura e analisi delle simulazioni e, grazie alla sua estensibilità, può essere utilizzato come base per ulteriori sviluppi e miglioramenti.

Bibliografia

[1] A. Messier, J. Robinson, K. Pahlavan, "*Performance Monitoring of a Wireless Campus Area Network*", IEEE, November 1997 https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=630992

[2] Amin S. Ibrahim, Khaled Y. Youssef, Hesham Kamel, Mohamed Abouelatta, "*Traffic modelling of smart city internet of things architecture*", May 2020 https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/iet-com.2019.1252

[3] Ali Mohd Ali,Mohammad R. Hassan, et al., "*Towards a Smart Environment: Optimization of WLAN Technologies to Enable Concurrent Smart Services*", February 2023 https://www.mdpi.com/1424-8220/23/5/2432

[4] Andrea Lacava, Michele Polese, Rajarajan Sivaraj, Rahul Soundrarajan, Bhawani Shanker Bhati, Tarunjeet Singh, Tommaso Zugno, Francesca Cuomo, Tommaso Melodia, "*Programmable and customized intelligence for traffic steering in 5g networks using open ran architectures*", April 2023 https://scholar.google.it/citations?view_op=view_citation&hl=it&user=E4clOJEAAAAJ&citation_for_view=E4clOJEAAAAJ:qjMakFHDy7sC

[5] Andrea Lacava, Matteo Bordin, Michele Polese, Rajarajan Sivaraj, Tommaso Zugno, Francesca Cuomo, Tommaso Melodia, "*ns-O-RAN: Simulating O-RAN 5G Systems in ns-3*", May 2023 https://scholar.google.it/citations?view_op=view_citation&hl=it&user=E4clOJEAAAAJ&citation_for_view=E4clOJEAAAAJ:IjCSPb-OGe4C