



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

CURRICULUM IN SICUREZZA INFORMATICA

TurtleVPN: Un Nuovo Testbed per Valutazioni Prestazionali su Traffico IP Sicuro

CANDIDATO

Orazio Cesarano

RELATORE

Arcangelo Castiglione

Anno Accademico 2022-2023

Sommario

Nell'ambito delle comunicazioni e soprattutto dell'utilizzo di flussi di dati atti a fornire un servizio a diversi host, è molto importante riuscire a capire attraverso quali strumenti poter offrire il miglior rapporto tra *performance* e *sicurezza* nella comunicazione stessa. Lo scopo di questo lavoro è quello di poter creare un dispositivo di testing il quale permetta di poter mettere a confronto diversi protocolli VPN in modo da stabilire quale di esso si comporta meglio in determinate situazioni, in particolare nel fornire servizi i quali si basano sull'invio di *flussi di dati* aventi precisi requisiti di *QoS* che devono essere rispettati affinché non ci siano problemi nella comunicazione. Per raggiungere questo obiettivo sono stati posti in analisi diversi *protocolli VPN* tra cui anche *WireGuard* versione *Post Quantum* il quale è stato progettato per poter garantire un ottimo grado di sicurezza anche contro *attaccanti* che potrebbero idealmente disporre di computer quantistici. Il dispositivo in questione ha come scopo secondario quello di poter fornire la possibilità di analizzare gli host presenti all'interno della rete locale *LAN* così da poter recuperare importanti informazioni riguardanti essi; oltre a questo dovrà anche fornire un punto di accesso VPN in modo da consentire l'utilizzo delle risorse presenti nella rete *LAN* del dispositivo anche da remoto, da parte degli host interessati.

Indice	ii
Elenco delle figure	v
1 Introduzione	1
1.1 Contesto ed obiettivo	2
1.1.1 Prestazioni	2
1.2 Motivazioni	2
1.3 Struttura della tesi	3
2 Background	4
2.1 Le reti informatiche	5
2.1.1 LAN	5
2.2 Sicurezza e VPN	5
2.2.1 WireGuard VPN	5
2.2.2 WireGuard Post Quantum	6
2.2.3 Introduzione alla crittografia Post Quantistica	6
2.3 Linux	7
2.3.1 Raspberry Pi	7
2.4 Tecnologie utilizzate	7
2.4.1 Docker	7
2.4.2 Celery	8
2.4.3 Redis	8

2.4.4	Nmap	8
2.4.5	Flask	8
2.5	Metodologia di testing	9
2.5.1	iPerf	9
3	Obiettivi della tesi	10
3.1	Obiettivo della tesi	11
3.2	Identificazione delle funzionalità del dispositivo	11
3.3	Identificazione requisiti non funzionali e di sicurezza	11
3.4	Interazione con la API	12
3.4.1	Use case modulo scansione	12
3.4.2	Use case modulo VPN	12
3.5	Conclusioni	13
4	Progettazione del sistema	14
4.1	Progettazione dei test e scelta della metodologia	15
4.1.1	Metodologia	15
4.2	Architettura del sistema	16
4.3	Progettazione hardware	17
4.3.1	Architettura del dispositivo Raspberry Pi	17
4.3.2	Scelta del sistema operativo	18
4.4	Progettazione software	18
4.4.1	Installazione del sistema operativo	18
4.4.2	Integrazione di Nmap per le scansioni di rete	19
4.4.3	Integrazione dei servizi VPN	19
5	Realizzazione del sistema proposto	20
5.0.1	Utilizzo dei protocolli VPN	21
5.1	Sviluppo e integrazione modulo scansione	21
5.1.1	modulo.py	21
5.1.2	tasks.py	23
5.1.3	app.py	23
5.2	Sviluppo e integrazione modulo VPN	23
5.2.1	WireGuard e Docker	24
5.2.2	WireGuard client	25

6 Testing e valutazione delle prestazioni del sistema	26
6.1 Progettazione dei test	27
6.2 Implementazione dei flussi con iPerf	27
6.3 Definizione metriche di valutazione	28
6.4 Esecuzione degli esperimenti e analisi dei risultati	29
6.4.1 Esempio di esecuzione test	29
6.5 Analisi dei risultati	30
6.5.1 Primo esperimento	31
6.5.2 Secondo esperimento	36
6.5.3 Terzo esperimento	41
6.5.4 Quarto esperimento	46
6.5.5 Quinto esperimento	51
7 Conclusioni e sviluppi Futuri	56
7.1 Risultati ottenuti	57
7.1.1 Valutazione dei risultati	57
7.2 Contributi della ricerca	57
7.3 Sviluppi futuri	58
7.3.1 Ulteriori esperimenti	58
7.3.2 Infrastruttura di rete	58
7.3.3 Realizzazione interfaccia web	58
Ringraziamenti	60
Bibliografia	62

Elenco delle figure

3.1	Use case modulo scansione.	12
3.2	Use case modulo VPN.	12
4.1	Flusso di esecuzione degli esperimenti.	16
4.2	Schema TurtleVPN.	17
4.3	Raspberry Pi 4 model B.	17
4.4	Raspberry Pi imager.	19
5.1	File del progetto.	22
5.2	Esempio scansione: scan_port.	22
5.3	Task per Celery.	23
5.4	Task per Flask.	23
5.5	File docker-compose.yaml.	24
5.6	Configurazione WireGuard client.	25
6.1	Comando iPerf server.	29
6.2	Comando iPerf client.	30
6.3	Grafici 1° esecuzione.	31
6.4	Grafici 5° esecuzione.	31
6.5	Grafici 10° esecuzione.	31
6.6	Grafici 1° esecuzione.	32
6.7	Grafici 5° esecuzione.	32
6.8	Grafici 10° esecuzione.	32

6.9 Grafici 1° esecuzione.	33
6.10 Grafici 5° esecuzione.	33
6.11 Grafici 10° esecuzione.	33
6.12 Grafici 1° esecuzione.	34
6.13 Grafici 5° esecuzione.	34
6.14 Grafici 10° esecuzione.	34
6.15 Grafici riassuntivi.	35
6.16 Grafici 1° esecuzione.	36
6.17 Grafici 5° esecuzione.	36
6.18 Grafici 10° esecuzione.	36
6.19 Grafici 1° esecuzione.	37
6.20 Grafici 5° esecuzione.	37
6.21 Grafici 10° esecuzione.	37
6.22 Grafici 1° esecuzione.	38
6.23 Grafici 5° esecuzione.	38
6.24 Grafici 10° esecuzione.	38
6.25 Grafici 1° esecuzione.	39
6.26 Grafici 5° esecuzione.	39
6.27 Grafici 10° esecuzione.	39
6.28 Grafici riassuntivi.	40
6.29 Grafici 1° esecuzione.	41
6.30 Grafici 5° esecuzione.	41
6.31 Grafici 10° esecuzione.	41
6.32 Grafici 1° esecuzione.	42
6.33 Grafici 5° esecuzione.	42
6.34 Grafici 10° esecuzione.	42
6.35 Grafici 1° esecuzione.	43
6.36 Grafici 5° esecuzione.	43
6.37 Grafici 10° esecuzione.	43
6.38 Grafici 1° esecuzione.	44
6.39 Grafici 5° esecuzione.	44
6.40 Grafici 10° esecuzione.	44
6.41 Grafici riassuntivi.	45
6.42 Grafici 1° esecuzione.	46

6.43 Grafici 5° esecuzione.	46
6.44 Grafici 10° esecuzione.	46
6.45 Grafici 1° esecuzione.	47
6.46 Grafici 5° esecuzione.	47
6.47 Grafici 10° esecuzione.	47
6.48 Grafici 1° esecuzione.	48
6.49 Grafici 5° esecuzione.	48
6.50 Grafici 10° esecuzione.	48
6.51 Grafici 1° esecuzione.	49
6.52 Grafici 5° esecuzione.	49
6.53 Grafici 10° esecuzione.	49
6.54 Grafici riassuntivi.	50
6.55 Grafici 1° esecuzione.	51
6.56 Grafici 5° esecuzione.	51
6.57 Grafici 10° esecuzione.	51
6.58 Grafici 1° esecuzione.	52
6.59 Grafici 5° esecuzione.	52
6.60 Grafici 10° esecuzione.	52
6.61 Grafici 1° esecuzione.	53
6.62 Grafici 5° esecuzione.	53
6.63 Grafici 10° esecuzione.	53
6.64 Grafici 1° esecuzione.	54
6.65 Grafici 5° esecuzione.	54
6.66 Grafici 10° esecuzione.	54
6.67 Grafici riassuntivi.	55

CAPITOLO 1

Introduzione

Nel presente capitolo verrà effettuata una panoramica generale sulle problematiche trattate dal lavoro svolto, motivando la necessità della realizzazione del dispositivo in questione. Verrà successivamente presentata l'idea della soluzione proposta ad alto livello e del confronto prestazionale tra i diversi protocolli VPN posti in analisi.

1.1 Contesto ed obiettivo

Questo progetto mira a sviluppare un dispositivo di testbed [1] il quale ha come obiettivo principale quello di poter eseguire una valutazione delle prestazioni su particolari flussi di traffico IP, impiegando VPN convenzionali e non in modo da poter estrapolare dai risultati, quale protocollo VPN si comporta meglio con i suddetti flussi. Il dispositivo realizzato avrà anche la capacità di eseguire una serie di scansioni per raccogliere una vasta gamma di informazioni sugli host connessi alla stessa rete locale (LAN). Tali informazioni saranno utili per chi gestisce la rete, fornendo una panoramica dettagliata degli host connessi. Inoltre, il dispositivo offrirà un servizio di connessione sicura realizzato mediante l'implementazione di un canale VPN (Virtual Private Network), che garantirà che tutte le comunicazioni passanti attraverso esso siano crittografate e sicure.

1.1.1 Prestazioni

Per quanto riguarda il confronto tra le diverse tecnologie VPN, esso sarà incentrato in primo luogo sulla valutazione del livello di sicurezza offerto dalle diverse versioni dei protocolli in modo da stabilire quali offrono un grado maggiore di sicurezza alle utenze coinvolte; successivamente saranno poi presi in considerazione anche i parametri *Bitrate* e *Bandwidth* per stabilire quale protocollo VPN si comporta meglio in termini di trasferimento dati [2].

1.2 Motivazioni

Le motivazioni dietro questo progetto sono molteplici. In primis, con l'avvento dei computer quantistici, la crittografia post-quantistica sta diventando sempre più importante. Portando avanti una ricerca che coinvolge le prestazioni di WireGuard PQ, il progetto può contribuire a guidare lo sviluppo futuro di protocolli di sicurezza di rete, assicurando che siano pronti per l'era post-quantistica. In secondo luogo, la crescente complessità delle reti locali (LAN) ha reso sempre più difficile per gli amministratori di rete monitorare e gestire gli host connessi; la creazione di un dispositivo in grado di eseguire scansioni dettagliate e raccogliere informazioni sugli host può semplificare notevolmente questa gestione. Infine, la sicurezza delle comunicazioni è diventata una preoccupazione fondamentale nell'era digitale. Fornendo un servizio di connessione sicura attraverso un canale VPN, il dispositivo può garantire che tutte le comunicazioni siano crittografate e protette, aumentando così la sicurezza generale della rete. Queste sono le principali motivazioni che hanno guidato la realizzazione di questa testbed.

1.3 Struttura della tesi

La tesi è strutturata come segue:

- **Capitolo 2 (Background):** questo capitolo fornisce una panoramica sui principali concetti indispensabili per comprendere a pieno le caratteristiche del dispositivo e dei protocolli VPN;
- **Capitolo 3 (Obiettivi della tesi):** questo capitolo descrive il cuore dello studio: stabilire gli obiettivi ci permetterà di delineare le funzionalità chiave che il sistema dovrà implementare per rispondere in modo efficace alle esigenze degli utenti finali e soprattutto stabilire quale protocollo VPN offre prestazioni adeguate in specifici casi;
- **Capitolo 4 (Progettazione del sistema):** In questo capitolo, trasformeremo i requisiti definiti nel capitolo precedente, in un progetto ben strutturato. Questo permetterà di costruire un sistema che non solo soddisfi le esigenze degli utenti, ma sia anche robusto, scalabile e manutenibile nel tempo;
- **Capitolo 5 (Realizzazione del sistema proposto):** In questo capitolo, le idee e i concetti, definiti precedentemente, saranno realizzate sotto forma di codice funzionante. Attraverso l'implementazione, il sistema prenderà forma, diventando un prodotto software pronto per essere utilizzato per raggiungere gli obiettivi prefissati;
- **Capitolo 6 (Testing e valutazione delle prestazioni del sistema):** In questa sezione, il sistema sarà messo alla prova, misurando le sue prestazioni in termini di velocità, efficienza e affidabilità. Questo permetterà di individuare eventuali aree di miglioramento e di ottimizzare il sistema per garantire un buon grado di affidabilità;
- **Capitolo 7 (Conclusioni e sviluppi futuri):** In questa sezione, saranno riassunti i risultati ottenuti ed inoltre, saranno discussi i potenziali miglioramenti che potrebbero essere apportati in futuro.

CAPITOLO 2

Background

Nell'ambito del presente capitolo sarà effettuata una panoramica sui principali concetti i quali rappresentano il punto cardine sul quale è stato sviluppato tutto il lavoro svolto.

2.1 Le reti informatiche

Una rete informatica è un sistema di comunicazione che è diventato di fondamentale importanza in questa epoca. Essa permette di scambiare informazioni tra dispositivi, anche di diversa natura tra cui *computer*, *periferiche*, *smartphone* ecc. Una rete informatica solitamente viene classificata in base alla sua dimensione geografica ed al mezzo trasmittivo che impiega per interconnettere i diversi dispositivi e scambiare informazioni.

2.1.1 LAN

La *LAN* (Local Area Network) [3] è una piccola rete di dispositivi che si estende lungo una ristretta area geografica, come un edificio o una casa. Essa consente la condivisione di risorse tra cui file e stampanti, rendendo semplice la collaborazione tra gli utenti. Le LAN possono essere cablate, utilizzando cavi Ethernet e dispositivi di rete per collegare fisicamente i dispositivi come gli *switch* ed i *router*; o wireless, utilizzando segnali radio per trasmettere dati. Inoltre, le LAN possono essere connesse tra loro per formare una rete più ampia, che prende il nome di *WAN* (Wide Area Network) [4], permettendo la condivisione di risorse e la comunicazione tra dispositivi su una scala molto più ampia, spesso su base globale.

2.2 Sicurezza e VPN

La sicurezza è un aspetto fondamentale delle reti, in particolare delle LAN. Con l'aumento delle minacce alla sicurezza informatica, è fondamentale proteggere le informazioni sensibili che viaggiano attraverso la rete. Questo può includere l'impiego di diverse tecniche e strumenti tra cui l'utilizzo di *firewall*, l'uso di software *antivirus*, la *crittografia dei dati*, l'implementazione di vari *protocolli di sicurezza* per proteggere le informazioni in transito. Uno di questi protocolli è quello *VPN* [5], o Virtual Private Network, che ha come obiettivo quello di creare un tunnel sicuro per il traffico di rete, proteggendo i dati da occhi indiscreti tramite il camuffamento dell'indirizzo IP reale dei dispositivi connessi.

2.2.1 WireGuard VPN

WireGuard [6] è un esempio di un protocollo VPN che è noto per la sua velocità e semplicità. Offre crittografia di alto livello e ha un codice sorgente relativamente piccolo, il che facilita l'ispezione del codice per eventuali vulnerabilità. WireGuard è anche molto versatile, con supporto per vari sistemi operativi e tipi di hardware. Esso è progettato per

essere facile da configurare e da utilizzare, rendendolo accessibile ad ogni tipo di utenza. Un aspetto unico di WireGuard è la sua implementazione della crittografia post-quantistica nella versione *WireGuard PQ* [7]. Questo offre una protezione aggiuntiva contro le potenziali minacce dei computer quantistici, garantendo che le comunicazioni rimangano sicure anche nell'era post-quantistica. In sintesi, WireGuard è un protocollo VPN all'avanguardia che combina velocità, sicurezza e facilità d'uso, offrendo una soluzione efficace per la sicurezza delle reti informatiche.

2.2.2 WireGuard Post Quantum

La versione post quantistica di WireGuard [8] che è stata scelta per condurre i successivi esperimenti apporta importanti migliorie, in particolare alla fase di *handshake* tramite il quale un qualsiasi *client* ed un *server*, possono reciprocamente autenticarsi in modo da poter stabilire un canale di comunicazione sicuro. Tale versione apporta delle modifiche al meccanismo di handshake impiegando l'algoritmo *Kyber* appartenente alla famiglia degli algoritmi *Crystals* [9] in quanto risulta essere *quantum resistant*. L'esecuzione di diversi test può contribuire a guidare lo sviluppo futuro dei protocolli di sicurezza di rete, assicurando che siano pronti per l'era post-quantistica. Queste sono le principali ragioni che hanno guidato la realizzazione di questo progetto.

2.2.3 Introduzione alla crittografia Post Quantistica

La crittografia post-quantistica [10] è un ramo della crittografia che si concentra sulla teoria e lo sviluppo di algoritmi critografici resistenti agli attacchi da parte dei computer quantistici. Questo nuovo modello di computer è in grado di poter risolvere determinati problemi computazionali molto più velocemente rispetto ai computer classici grazie all'utilizzo dei *qubit* invece dei *bit* per rappresentare e manipolare le informazioni. A differenza dei bit classici, che possono essere 0 o 1, i qubit possono esistere in uno stato di *sovraposizione*, dove possono essere sia 0 che 1 contemporaneamente ed è proprio grazie a questa proprietà che i computer quantistici possono risolvere alcuni problemi molto più velocemente dei loro modelli classici. La realizzazione pratica di computer quantistici è ancora un'impresa difficile ma lo sviluppo della crittografia post-quantistica rimane comunque in pieno sviluppo, data la potenziale minaccia che i computer quantistici rappresentano per i sistemi crittografici attuali.

In sintesi, la sicurezza delle reti informatiche è un campo in continua evoluzione che richiede l'implementazione di protocolli di sicurezza robusti come VPN e WireGuard, nonché la preparazione per le future minacce della crittografia post-quantistica.

2.3 Linux

Linux è un sistema operativo *open source* noto per la sua flessibilità e versatilità, con numerose distribuzioni disponibili per soddisfare una varietà di esigenze. Linux è composto da un componente essenziale chiamato *Kernel* [11], il quale si interpone tra le risorse hardware ed il sistema operativo in modo da creare un'interfaccia di comunicazione. La filosofia alla base di Linux è quella di fornire un sistema operativo che sia completamente aperto e modificabile dall'utente, promuovendo la condivisione, la collaborazione e la libertà di scelta.

2.3.1 Raspberry Pi

Raspberry Pi è una piccola scheda *On Board* di dimensioni molto ridotte la quale è stata progettata principalmente per uso didattico. Essa permette l'esecuzione di numerose applicazioni e di diversi sistemi operativi grazie al tipo di architettura *ARM* con la quale queste schede sono state ideate [12]. Con il tempo sono state prodotte diverse schede, ognuna delle quali rappresenta l'evoluzione della precedente in termini di prestazioni. Attualmente questi dispositivi sono impiegati in diversi settori tra cui quello delle *telecomunicazioni*, *IoT*, *networking*, grazie al costo relativamente contenuto ed al set di periferiche disponibili.

2.4 Tecnologie utilizzate

2.4.1 Docker

Docker è un software *open source* impiegato per consentire l'esecuzione e la gestione di software ospitati all'interno di *container* autonomi, eseguibili sia in locale che in cloud [13]. Questo strumento viene spesso impiegato quando si vuole avere un container contenente tutto il necessario per l'esecuzione di un'applicazione come: *library*, *codice*, *ambiente di esecuzione* ecc. Nello specifico è stata utilizzata la versione *Compose* la quale offre la possibilità di poter gestire un'applicazione utilizzando più di un unico container così da poter aver la possibilità di estendere il progetto in futuro, in caso di necessità [14].

2.4.2 Celery

Celery è un *task scheduler* che permette di eseguire lavori in modalità *asincrona*, utile quando il compito da svolgere può richiedere diverso tempo e non si ha la possibilità di attendere la sua fine. Celery fornisce anche una *API Python* in modo da poter definire un'attività da svolgere e soprattutto gestire il loro stato. Esso è molto flessibile ed offre la possibilità di poter utilizzare diversi *message broker* come ad esempio *Redis* o *RabbitMQ* [15].

2.4.3 Redis

Redis è un *database* di tipo *chiave-valore* che offre tempi di risposta inferiori al millisecondo per cui viene molto spesso impiegato dato che gode di ottime prestazioni in lettura e scrittura dei dati [?]. Oltre ad essere impiegato come database, Redis ha la possibilità di funzionare anche come *cache* o *message broker* ed è per questo motivo che è stato scelto per essere utilizzato con Celery. Un *message broker* è un'applicazione che svolge il compito di intermediario tra due diverse architetture che comunicano tramite messaggi.

2.4.4 Nmap

Nmap è uno scanner di rete il quale permette di ricostruire la struttura della rete interna. Esso viene ampiamente utilizzato dato che fornisce la possibilità di personalizzare il tipo di scansione da eseguire; alcune delle varie features che caratterizzano Nmap sono:

- Individuare i diversi dispositivi in rete;
- Identificare i servizi in esecuzione sui dispositivi;
- Rilevare le versioni dei servizi in esecuzione;
- Individuare il sistema operativo in esecuzione sui dispositivi.

2.4.5 Flask

Flask è un framework scritto in *Python* il quale permette lo sviluppo backend di un'applicazione web. Ultimamente sta prendendo piede grazie alla sua semplicità ed alla possibilità di scalare facilmente per la costruzione di applicazioni complesse; un'altra caratteristica importante riguarda la community che offre costantemente la possibilità di poter integrare nuove funzionalità.

2.5 Metodologia di testing

Per quanto riguarda i test che sono stati eseguiti con la testbed *TurtleVPN*, essi sono stati ideati, con lo scopo di simulare nel modo più fedele possibile, diversi flussi di dati aventi specifici requisiti *time sensitive* in modo da stabilire le performance dei diversi protocolli VPN presi in esame.

2.5.1 iPerf

Il tool in questione è un particolare strumento di diagnostica di rete Open Source il quale consente di misurare le performance della rete tramite la regolazione di numerosi parametri che consentono di simulare in modo fedele i diversi tipi di flussi di dati che possono scambiarsi due differenti host in rete. Per poter eseguire i test è necessario che i due host installino entrambi il tool in modo da poter stabilire un collegamento e poter proseguire con il trasferimento dei flussi ideati.

CAPITOLO 3

Obiettivi della tesi

Questo capitolo descrive il cuore dello studio: la definizione degli obiettivi permetterà di delineare le funzionalità chiave che il sistema dovrà implementare per rispondere in modo efficace alle esigenze degli utenti finali.

3.1 Obiettivo della tesi

Come è stato accennato nei capitoli precedenti, l’obiettivo principale sul quale è stato fondato tutto lo studio presente all’interno di questa tesi, riguarda principalmente l’analisi e lo studio delle prestazioni relative alla sicurezza di particolari flussi di dati i quali posseggono determinati requisiti *time sensitive* che devono essere rispettati affinché possano fornire la piena funzionalità dei sistemi correlati e soprattutto garantire una determinata qualità dei servizi offerti. I flussi definiti saranno posti in analisi tenendo in considerazione del fatto che su di essi impatterà l’utilizzo di diversi protocolli VPN tra cui *WireGuard*. Con questo studio si intende contribuire attivamente alla ricerca, in modo da stabilire chiaramente il comportamento di determinati flussi di dati in relazione anche al protocollo VPN scelto per garantire il giusto grado di sicurezza.

3.2 Identificazione delle funzionalità del dispositivo

Passiamo ora alla definizione delle diverse funzionalità che il dispositivo in questione dovrà supportare per soddisfare le esigenze degli utenti. Di seguito sono elencate le principali utilità del dispositivo:

- Il dispositivo deve fornire la possibilità di eseguire scansioni sui dispositivi connessi alla stessa rete LAN della testbed;
- La testbed deve fornire la possibilità connettersi ad un server VPN in modo da poter navigare in modo sicuro ed accedere alle risorse della rete da remoto.

3.3 Identificazione requisiti non funzionali e di sicurezza

Ora passiamo alla definizione dei vincoli che descrivono le proprietà e le restrizioni riguardanti il sistema stesso oltreché alle procedure necessarie per evitare la sua compromissione:

- Il dispositivo deve essere costantemente alimentato;
- Il dispositivo deve autenticare gli utenti prima di permettere l’utilizzo del servizio VPN;
- I dati trasmessi tramite canale VPN devono essere crittografati;
- Il dispositivo dovrebbe essere fisicamente resistente ad attacchi.

3.4 Interazione con la API

In questa sezione sono rappresentate le interazioni tra il sistema ed i principali attori coinvolti nel normale utilizzo. Analizziamo i casi d'uso relativi alle funzionalità da realizzare.

3.4.1 Use case modulo scansione

Il caso d'uso in questione riguarda l'utilizzo del modulo adibito alle scansioni degli *host* presenti nella LAN a cui è connesso il dispositivo da parte del network admin o di chi ne fa le veci.

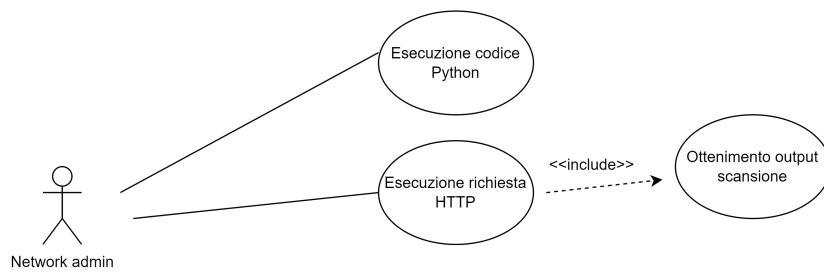


Figura 3.1: Use case modulo scansione.

3.4.2 Use case modulo VPN

Come si evince dall'immagine sottostante, il modulo VPN viene anch'esso gestito da una figura preposta a tale lavoro; mentre i vari client che desiderano connettersi al *server VPN* hanno la possibilità di usufruire del servizio finale cioè entrare a far parte della rete LAN del dispositivo o navigare in modo sicuro.

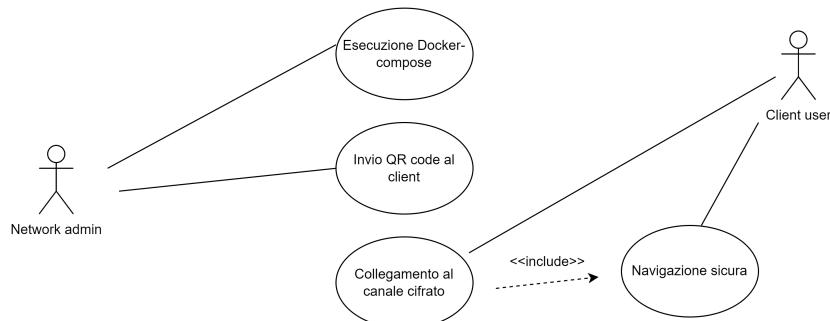


Figura 3.2: Use case modulo VPN.

3.5 Conclusioni

I requisiti raccolti in questa fase saranno successivamente impiegati per la progettazione e l'implementazione del sistema proposto.

CAPITOLO 4

Progettazione del sistema

In questo capitolo, trasformeremo i requisiti definiti nel capitolo precedente, in un progetto ben strutturato. Questo permetterà di costruire un sistema che non solo soddisfi le esigenze degli utenti, ma che sia anche robusto, scalabile e manutenibile nel tempo.

4.1 Progettazione dei test e scelta della metodologia

Nel capitolo precedente sono stati definiti gli obiettivi che il presente studio intende raggiungere, oltre alle funzionalità che il sistema dovrà offrire a chi intende utilizzarlo. In questo capitolo saranno progettate le fasi utili al raggiungimento degli obiettivi prefissati partendo dalla definizione della *metodologia* impiegata per l'ottenimento dei risultati sperati.

4.1.1 Metodologia

Per quanto riguarda i diversi esperimenti condotti con il sistema realizzato, è stato definito un *flusso* di esecuzione univoco per tutti in modo da avere un metodo sperimentale da seguire, il quale aiuta a garantire che i risultati ottenuti siano affidabili e non siano dovuti al caso. Come rappresentato nella figura 4.1, ogni singolo esperimento si compone di diverse fasi:

- Definizione del flusso di dati da testare;
- Invio del flusso tra client e server;
- Ottenimento output;
- Ripetizione del test 10 volte;
- Generazione dei grafici utilizzando gli output di ogni esecuzione.

Una volta definito il set di esperimenti ed i relativi flussi di dati aventi particolari requisiti di *QoS*, esso sarà utilizzato in accoppiata con i protocolli *WireGuard standard*, *WireGuard post quantum*, *OpenVPN* così da poter tirare fuori degli output sulla quale fare le dovute considerazioni.

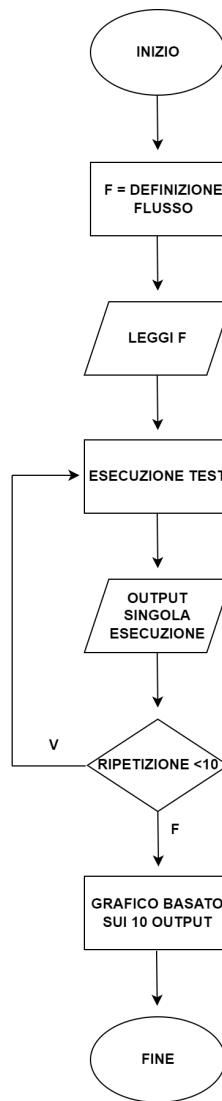


Figura 4.1: Flusso di esecuzione degli esperimenti.

4.2 Architettura del sistema

L’architettura del sistema è fondamentale per comprendere come esso funzioni nel suo insieme e come le sue parti interagiscono tra loro. Nel contesto di questo progetto, l’*architettura fisica* è organizzata come rappresentato nella figura 4.2; il dispositivo realizzato sarà collocato all’interno di una qualsiasi rete LAN e permetterà, ai diversi client che vogliono accedere alla medesima LAN, di poterlo fare tramite la connessione ad un canale VPN realizzato con WireGuard.

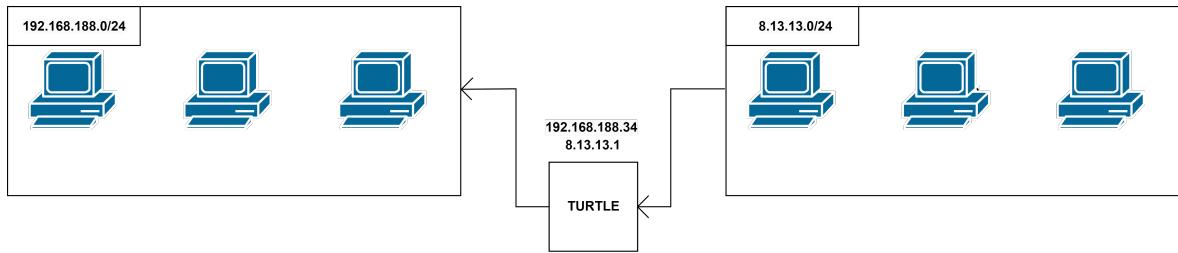


Figura 4.2: Schema TurtleVPN.

4.3 Progettazione hardware

La scelta dei componenti *hardware* è un passaggio fondamentale in qualsiasi progetto e ancor di più in questo; essa influenzera le prestazioni di tutto il sistema per cui è molto importante scegliere le componenti adeguate anche in base alle performance che si vogliono ottenere, soprattutto per garantire una buona *stabilità* e *connettività* agli utenti finali. Alla base del sistema da realizzare è stata impiegata la scheda *Raspberry Pi 4 model B* [16].

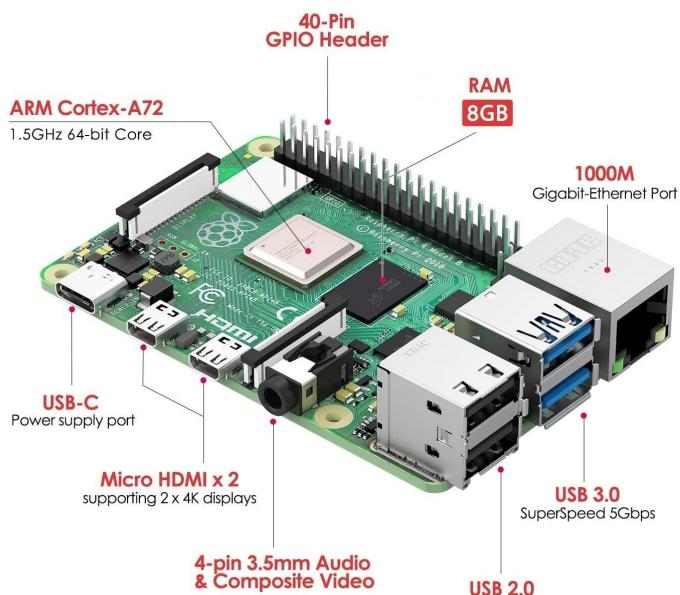


Figura 4.3: Raspberry Pi 4 model B.

4.3.1 Architettura del dispositivo Raspberry Pi

Il sistema ideato sarà implementato a partire dalla scheda one board *Raspberry Pi 4 model B* che offre la possibilità di poter installare diversi sistemi operativi *Linux based*. Tale scheda ha delle prestazioni molto simili ad un normale computer di fascia entry-level permettendo così di adempiere in modo adeguato ai compiti stabiliti.

- Processore quad-core a 64 bit
- 8GB di RAM
- Decodifica hardware video fino a 4Kp60
- Dual-band 2.4 /5.0 GHz LAN wireless
- Gigabit Ethernet
- Bluetooth 5.0
- USB 3.0

4.3.2 Scelta del sistema operativo

Per quanto riguarda la scelta del sistema operativo, ne esistono di svariati tra cui *Raspbian*, *Ubuntu MATE*, *Arch Linux* ma per questo progetto è stato utilizzato *Ubuntu Server* versione 22.04.3 LTS il quale offre sicuramente ottima stabilità e costanza negli aggiornamenti per lungo tempo.

4.4 Progettazione software

Passiamo ora alla progettazione delle funzionalità stabilite. Esse saranno organizzate in diversi *moduli* ognuno dei quali sarà completamente indipendente e funzionante in autonomia. Prima di procedere è necessaria eseguire l'installazione del sistema operativo sulla scheda in questione.

4.4.1 Installazione del sistema operativo

L'installazione del sistema operativo sulla scheda Raspberry Pi 4 è stata eseguita utilizzando l'ottimo tool *Pi imager* il quale offre la possibilità di poter scaricare un sistema operativo compatibile con una delle schede Raspberry e di automatizzare il processo di installazione di esso.

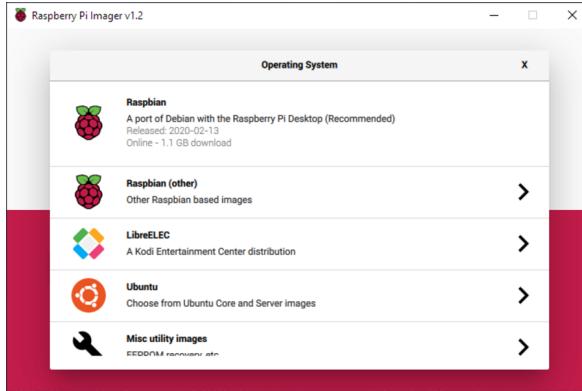


Figura 4.4: Raspberry Pi imager.

4.4.2 Integrazione di Nmap per le scansioni di rete

Il primo modulo è stato progettato con lo scopo di fornire all’utente finale la possibilità di effettuare diverse tipologie di scansione all’interno della rete in modo da poter acquisire facilmente importanti informazioni sugli *host* presenti in essa. Per soddisfare tale compito è stato scelto di utilizzare il noto network scanner *Nmap* [17]. Per quanto riguarda la realizzazione del modulo in sé è stato scelto di impiegare il linguaggio di programmazione *Python* [18] il quale al giorno d’oggi rappresenta sicuramente uno dei linguaggi più versatili, avente un gran numero di *library* a disposizione.

Gestione delle richieste HTTP

Le diverse scansioni realizzate tramite l’utilizzo di *Nmap* e *Python* saranno poi rese accessibili ai diversi utilizzatori grazie all’impiego del framework *Flask* [19] il quale rappresenta un ottima scelta per quanto riguarda la programmazione web *backend*. Con tale framework saranno implementate diverse funzioni che possono gestire le richieste HTTP in arrivo dai client.

4.4.3 Integrazione dei servizi VPN

Il secondo modulo è stato progettato con l’obiettivo di fornire agli utenti la possibilità di connettersi ad un *server VPN* il quale instaura un canale cifrato tra il client che usufruisce del servizio e la rete LAN a cui è connesso il dispositivo Raspberry Pi 4, in modo che tale client entri a far parte di quella rete e possa avere accesso alle *risorse* presenti all’interno di essa. Per assolvere a questo compito è stato scelto di utilizzare il protocollo *WireGuard VPN* versione *standard* [20] il quale risulta essere molto semplice da configurare ma allo stesso tempo offre ottime prestazioni, superiori anche al protocollo *OpenVPN* [21].

CAPITOLO 5

Realizzazione del sistema proposto

In questo capitolo, le idee e i concetti, definiti precedentemente, saranno realizzati sotto forma di codice funzionante. Attraverso l'implementazione, il sistema prenderà forma, diventando un prodotto software pronto per essere impiegato nella risoluzione dei compiti preposti.

Come ampiamente esplicitato nei capitoli precedenti, l’obiettivo principale del lavoro svolto è quello di poter analizzare il comportamento di determinati flussi di dati con particolari requisiti di *QoS* in relazione all’utilizzo di diversi protocolli VPN. Vediamo ora nel dettaglio come è stato svolto questo compito.

5.0.1 Utilizzo dei protocolli VPN

Per quanto riguarda l’utilizzo dei protocolli VPN, come è stato presentato nei capitoli precedenti, è stato scelto di utilizzare tre diversi protocolli affinché siano ben chiare le performance ottenibili tramite il loro impiego. I flussi di test precedentemente definiti sono stati scambiati tra client e server i quali sono stati messi in comunicazione tramite l’attivazione di un singolo protocollo VPN per volta in modo da stabilire un canale di comunicazione *cifrato* tra loro. Tutti i test eseguiti sono stati portati avanti utilizzando i protocolli VPN installati *localmente* al sistema realizzato così da evitare l’utilizzo di *container* che avrebbero potuto compromettere i risultati finali dei test.

5.1 Sviluppo e integrazione modulo scansione

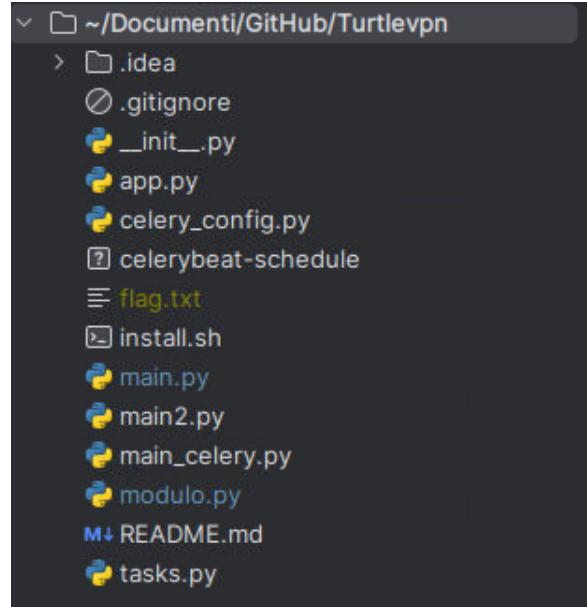
Passiamo ora all’implementazione vera e propria del modulo adibito all’esecuzione delle scansioni dei dispositivi presenti in rete. Come è stato accennato nel capitolo precedente, il presente modulo è stato realizzato impiegando *Python* come linguaggio di programmazione proprio perché offre la possibilità di poter integrare numerose librerie utili allo sviluppo del codice tra cui *Nmap*, *Flask* e *Celery*. Nella figura 5.1 è possibile notare la struttura del progetto realizzato ed i file che lo compongono.

Passiamo ora ad analizzare i principali file che implementano le funzionalità del sistema in questione.

5.1.1 modulo.py

Questo file contiene tutte le scansioni implementate tramite l’utilizzo della libreria *Nmap*. ogni scansione genera un file di output *testuale*, così da poter consultare i risultati anche in un secondo momento dopo la scansione; in particolare sono state implementate diverse scansioni affinché sia possibile cogliere diversi aspetti riguardanti gli host in rete:

- *scans (indirizzo)*: elenca i diversi host presenti in rete;
- *scans_time (indirizzo, timer)*: scansione periodica della rete;

**Figura 5.1:** File del progetto.

- *scan_port (indirizzo, porte)*: scansione di determinate porte;
- *scan_service (indirizzo)*: scansione dei servizi presenti su un indirizzo;
- *scan_os (indirizzo)*: scansione del sistema operativo degli host;
- *scan_vuln (indirizzo)*: scansione delle vulnerabilità degli host.

Nella figura 5.2 è riportato un esempio di scansione, in particolare è l'implementazione della scansione *scan_port*. Per completezza è possibile consultare le altre scansioni all'interno del progetto GitHub [22].

```
def scan_port(indirizzo, porte): # SCANSIONE PER UN CERTO IP E PORTE
    nmap = nmap3.Nmap()
    found = re.fullmatch(pattern: r"\[0-9]{1,3}\.\[0-9]{1,3}\.\[0-9]{1,3}\.\[0-9]{1,3}", indirizzo) # regex per controllo ip
    found2 = re.fullmatch(pattern: r"\[0-9]{1,5}", porte) # regex controllo singola porta
    found3 = re.fullmatch(pattern: r"\[0-9]{1,5}\-\[0-9]{1,5}", porte) # regex controllo intervallo porte
    found4 = re.fullmatch(pattern: r"\[0-9]{1,5}\.*", porte) # regex controllo elenco porte
    if found and (found2 or found3 or found4): # se l'ip è nella forma corretta
        print("IP E PORTE SINTATTICAMENTE CORRETTI")
        print("ESEGUO LA SCANSIONE DELL'IP: ", indirizzo)
        result = nmap.scan_top_ports(indirizzo, args="-p" + porte) # scansione le porte prese in input
        print(json.dumps(result, indent=2), file=open("output_scan_port" + str(datetime.now()) + ".txt", "w+"))
        return 1
    else:
        print("RITENTA LA SCANSIONE CON UN IP CORRETTO")
        return 0
```

Figura 5.2: Esempio scansione: scan_port.

5.1.2 tasks.py

In questo file sono stati definiti i diversi *task* eseguibili tramite l’impiego di *Celery*, uno scheduler Open Source che permette di programmare l’esecuzione dei task ed anche di eseguirli in modo *asincrono* così da non bloccare l’esecuzione e l’interattività di una pagina web. Nella figura seguente è riportata la definizione di un task per Celery; ovviamente tutti gli altri task sono presenti nel progetto *GitHub* [22].

```
@app.task
def run_scans(ip_address):
    modulo.scans(ip_address)
```

Figura 5.3: Task per Celery.

5.1.3 app.py

In questo file sono state definite le varie *richieste HTTP* in modo da poter utilizzare i task, definiti precedentemente, anche richiamandoli da una possibile pagina web. Le diverse richieste sono state realizzate tramite l’utilizzo di *Flask*, un particolare framework per lo sviluppo web backend in Python.

```
@app.route('/scan', methods=['POST'])
def scan():
    ip_address = request.json.get('ip_address')

    if not ip_address:
        return jsonify({'errore': 'IP richiesta'}), 400

    result = run_scans.delay(ip_address)
    return jsonify({'task_id': result.task_id}), 202
```

Figura 5.4: Task per Flask.

5.2 Sviluppo e integrazione modulo VPN

In questa sezione è stato sviluppato il modulo inerente all’impiego del protocollo *WireGuard* versione *standard* per fornire agli utenti la possibilità di accedere alle risorse presenti nella rete a cui è connesso il sistema *TurtleVPN* oltre che a fornire la possibilità di poter navigare in rete in modo sicuro.

5.2.1 WireGuard e Docker

A tale scopo è stato utilizzato il gestore di container *Docker* il quale permette di poter gestire in modo agevole i diversi container in cui sono stati istanziati gli applicativi di interesse. In questo caso la versione di Docker utilizzata è la *Compose* [14] che a differenza della versione standard permette di poter gestire più container allo stesso tempo, di conseguenza offre la possibilità di poter estendere o implementare nuove funzionalità legate all'utilizzo del protocollo WireGuard. Nella figura 5.5 è possibile osservare la configurazione del file *docker-compose.yaml* il quale ha il compito di istanziare i servizi definiti all'interno di esso in base alla configurazione scelta.

```
orazio@orazio-bt3:/opt/wireguard-server$ cat docker-compose.yaml
version: "2.1"
services:
  wireguard:
    image: local_wireguard
    container_name: wireguard
    cap_add:
      - NET_ADMIN
      - SYS_MODULE #optional
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/Berlin
      - SERVERURL=wireguard-vpn-turtle.duckdns.org
      - SERVERPORT=51820
      - PEERS=1 #optional
      - PEERDNS=1.1.1.1,1.0.0.1 #optional
      - INTERNAL_SUBNET=8.13.13.0 #optional
      - ALLOWEDIPS=0.0.0.0/0,::/0 #optional
      - PERSISTENTKEEPALIVE_PEERS= #optional
      - LOG_CONFS=true #optional
    volumes:
      - /opt/wireguard-server/config:/config
      - /lib/modules:/lib/modules #optional
    ports:
      - 51820:51820/udp
    sysctls:
      - net.ipv4.conf.all.src_valid_mark=1
    restart: always
```

Figura 5.5: File docker-compose.yaml.

docker-compose.yaml

Alcuni dettagli presenti nel file di configurazione:

- L'immagine di WireGuard è presente localmente al sistema;
- L'indirizzo *reale* del server è stato impostato in modo da fare sempre riferimento all'indirizzo *pubblico* della rete a cui è connesso TurtleVPN in modo che i client possano sempre navigare utilizzando quell'indirizzo. Per raggiungere questo obiettivo è stato impiegato il servizio *Duck DNS* [23] il quale permette di assegnare ad una stringa l'indirizzo di rete pubblico e di gestire in maniera automatica l'indirizzo di rete fornito dal *provider* anche nel caso in cui cambi dopo un certo lasso di tempo;

- La connessione al server WireGuard avviene tramite la porta *51820 UDP*;
- *8.13.13.0/24* rappresenta la *subnet* assegnata ai dispositivi connessi a TurtleVPN utilizzando un canale sicuro instaurato tramite WireGuard;

5.2.2 WireGuard client

Relativamente ai dispositivi client che intendono connettersi al server VPN è necessario che essi eseguano l'*import* del file di configurazione che è stato generato dall'applicativo server WireGuard in esecuzione nel sistema; per i dispositivi mobili è possibile anche effettuare la scansione del *QR-Code* così da poter eseguire l'importazione della configurazione in modo agevole all'interno dell'applicazione client. Nella figura 5.6 è mostrata la configurazione per un client.

```
orazio@orazio-bt3:/opt/wireguard-server/config/peer1$ cat peer1.conf
[Interface]
Address = 8.13.13.2
PrivateKey = KI/pYA3A3Rhhc8/tibAPIdtErKb/cwErGJvmt418o1s=
ListenPort = 51820
DNS = 1.1.1.1,1.0.0.1

[Peer]
PublicKey = pYhURzbyS0owBoHLGMyyGvUi6GejtLQezHyFRrfrMBA=
PresharedKey = xk1lf+CaaFgGs/sOXGwwgLTevEKdJ2jXr/MGzpCjxCs=
Endpoint = wireguard-vpn-turtle.duckdns.org:51820
AllowedIPs = 0.0.0.0/0,::/0
```

Figura 5.6: Configurazione WireGuard client.

CAPITOLO 6

Testing e valutazione delle prestazioni del sistema

In questa sezione, il sistema sarà messo alla prova, misurando le sue prestazioni in termini di velocità, efficienza e affidabilità. Questo permetterà di individuare eventuali aree di miglioramento e di ottimizzare il sistema per garantire la migliore esperienza possibile agli utenti finali.

6.1 Progettazione dei test

Come è stato spiegato nel capitolo *Progettazione*, alla base di ogni esperimento è stato definito un preciso *flusso di dati* avente dei requisiti di *QoS*. Sono stati presi in considerazione i flussi che al giorno d’oggi sono maggiormente utilizzati nell’ambito delle comunicazioni e di fornitura di determinati servizi web based. Sotto sono riportati i flussi che sono stati presi in considerazione per gli esperimenti da eseguire; ogni flusso ha il compito di simulare uno specifico servizio nel modo più fedele possibile:

- Flusso UDP per misurare il *throughput* massimo;
- Flusso TCP per misurare il *throughput* massimo;
- Flusso UDP per simulare il protocollo *VoIP*;
- Flusso per simulare uno *streaming video*;
- Flusso per simulare un *trasferimento massivo*.

6.2 Implementazione dei flussi con iPerf

Nel paragrafo precedente sono stati progettati i diversi flussi di dati da utilizzare per i vari test da eseguire; per la realizzazione vera e propria si è scelto di utilizzare *iPerf* il quale permette di poter simulare un flusso di dati in maniera molto precisa tramite l’utilizzo di numerosi *flag* e di poter gestire lo scambio di dati tra un *server* ed un *client* i quali eseguono entrambi il tool. Sotto è riportata la definizione esatta dei comandi per simulare i flussi:

- `.\iperf3 -c 10.66.66.1 -p 2020 -u -b 0 -n 512M -get-server-output;`

Questo comando simula l’invio di 512 MB tramite il protocollo di rete UDP, utilizzando tutta la banda disponibile.

- `.\iperf3 -c 10.66.66.1 -p 2020 -P 10 -b 0 -w 100k -get-server-output;`

Questo flusso simula l’invio di pacchetti TCP utilizzando dieci connessioni parallele verso il server ed la massima banda disponibile.

- `.\iperf3 -c 10.66.66.1 -u -p 2020 -S 0x28 -l 78 -b 100K -get-server-output;`

Questo flusso simula l’invio di pacchetti UDP appartenenti ad un flusso di dati di tipo VoIP.

- `.\iperf3 -c 10.66.66.1 -p 2020 -S 32 -M 1460B -get-server-output;`

Questo flusso simula l'invio di pacchetti TCP appartenenti ad un flusso di streaming video.

- `.\.\iperf3 -c 10.66.66.1 -p 2020 -S 10 -M 1460B -b 0 -get-server-output;`

Questo flusso simula l'invio di pacchetti TCP appartenenti ad un flusso bulk data cioè di trasferimento di massa.

Dettagli sui flag utilizzati

Al fine di comprendere a pieno come sono stati simulati i diversi flussi di dati scambiati negli esperimenti tra client e server, è importante capire il significato dei *flag* utilizzati:

- *-c*: Indirizzo dell'host che esegue iPerf server;
- *-p*: Porta in ascolto di iPerf server;
- *-u*: Flusso di tipo UDP;
- *-b*: Banda disponibile per il trasferimento. 0 indica che la banda è massima;
- *-n*: Quantità di byte da inviare durante il test;
- *-P*: Numero di connessioni parallele del client;
- *-S*: Assegna ai pacchetti una determinata priorità per simulare i pacchetti di un preciso protocollo applicativo;
- *-M*: Grandezza massima del singolo pacchetto TCP;
- *-get-server-output*: L'output del test viene visualizzato anche sul client.

6.3 Definizione metriche di valutazione

Arrivati a questo punto del lavoro è molto importante capire sulla base di quali *parametri* effettuare la valutazione del sistema realizzato in modo da poter stabilire con certezza l'efficacia di esso. A tal proposito, è stato scelto di rendere le valutazioni degli esperimenti del tutto indipendenti tra loro; in particolare i diversi esperimenti sono stati valutati in base ai valori assunti principalmente dai due seguenti parametri:

- *Bitrate*: Indica la quantità di informazioni che sono state trasferite in un'unica unità di tempo;

- *Trasferimento*: Indica la quantità di informazioni trasferite durante l'esecuzione dell'intero test.

I parametri sopracitati sono di fondamentale importanza affinché si possa stabilire se le *condizioni di rete*, derivanti anche dall'utilizzo o meno di uno dei protocolli VPN, siano adatte per consentire il corretto impiego dei flussi stabiliti.

6.4 Esecuzione degli esperimenti e analisi dei risultati

Il passo successivo è quello di proseguire con l'esecuzione degli esperimenti prefissati in modo da poter raccogliere i dati di output e poterli utilizzare per la generazione di *grafici* da utilizzare come strumento di ausilio all'analisi degli output stessi. Al fine di evitare ripetizioni superflue è stato evitato di trattare ogni singola esecuzione di ogni test per cui nel paragrafo successivo è riportato un singolo campione di esecuzione che rispecchia la struttura di tutte le altre indipendentemente dal protocollo VPN utilizzato. I dati di output di tutte le esecuzioni dei test sono presenti nel progetto *GitHub* [22].

6.4.1 Esempio di esecuzione test

Nel seguente sottoparagrafo è riportato un esempio di una singola esecuzione del test *numero 1* tra il sistema TurtleVPN ed un client, avente sistema operativo *Windows 11*, il quale invia il flusso di dati in modo da poter ricevere i risultati di output. Nella figura 6.1 è rappresentata l'esecuzione del comando per fare in modo che TurtleVPN possa ricevere il flusso di dati prestabilito che sarà inviato dal client, come in figura 6.2, il quale riceverà anche l'output dell'esecuzione. Ogni esperimento è stato eseguito allo stesso modo indipendentemente dal protocollo VPN impiegato. Da come si nota dalla figura 6.2, l'output del test è suddiviso in intervalli i quali sono numericamente differenti in base al tipo di flusso simulato.

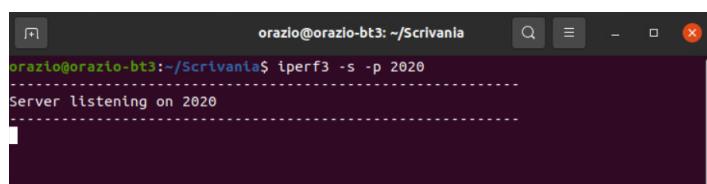
A screenshot of a terminal window titled "orazio@orazio-bt3: ~/Scrivania". The command "iperf3 -s -p 2020" is being run. The output shows "Server listening on 2020" followed by a series of dashed lines indicating the start of data transmission intervals.

Figura 6.1: Comando iPerf server.

```
PS C:\Users\Orazio\Desktop\pc bsq\iperf-3.1.3-win64> .\iperf3 -c 10.66.66.1 -p 2020 -u -b 0 -n 512M --get-server-output
Accepted connection from 10.66.66.2, port 63818
[ 5] local 10.66.66.1 port 2020 connected to 10.66.66.2 port 55385
[ ID] Interval           Transfer     Bitrate      Jitter    Lost/Total Datagrams
[ 5]  0.00-1.00   sec   67.1 Mbytes   563 Mbits/sec  0.065 ms  3366/11952 (28%)
[ 5]  1.00-2.00   sec   47.2 Mbytes   396 Mbits/sec  0.513 ms  7463/13504 (55%)
[ 5]  2.00-2.12   sec   0.00 Bytes   0.00 bits/sec  0.513 ms  0/0 (0%)
[ ID] Interval           Transfer     Bitrate      Jitter    Lost/Total Datagrams
[ 5]  0.00-2.12   sec   114 Mbytes   452 Mbits/sec  0.513 ms  10829/25456 (43%)  receiver
```

Figura 6.2: Comando iPerf client.

Sulla base degli output ottenuti dalle varie esecuzioni di ogni test, saranno estrapolate le considerazioni sui protocolli VPN, grazie anche ai grafici generati a partire proprio dai dati di output dei test.

6.5 Analisi dei risultati

Analizziamo ora i risultati ottenuti dai test; in particolare i grafici riguardano i parametri *bitrate* e *transfer* ottenuti in output da ogni esecuzione. Per ogni esperimento sono quindi stati creati *due* grafici per ognuna delle *dieci* esecuzioni del test in modo da avere una visione chiara dell’andamento dell’esperimento, inoltre per ogni esperimento sono stati generati anche due grafici ad istogramma rappresentanti *bitrate* e *transfer* medio. Visto che ogni esperimento è stato eseguito con diversi protocolli VPN, si eviterà di mostrare tutti i grafici affinché si possa cogliere il senso di ogni esperimento; il set di grafici completo è presente nel progetto *GitHub* [22].

6.5.1 Primo esperimento

L'esperimento in questione riguarda l'invio di 512 MB tramite l'utilizzo del protocollo UDP, sfruttando tutta la banda di rete disponibile.

Grafici esperimento senza VPN

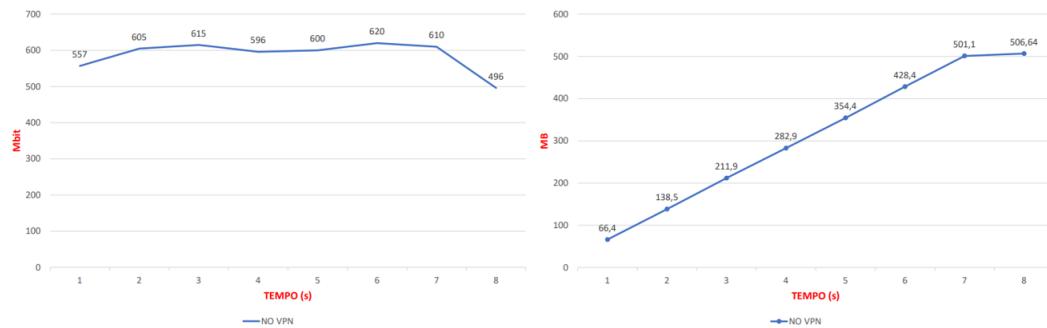


Figura 6.3: Grafici 1° esecuzione.

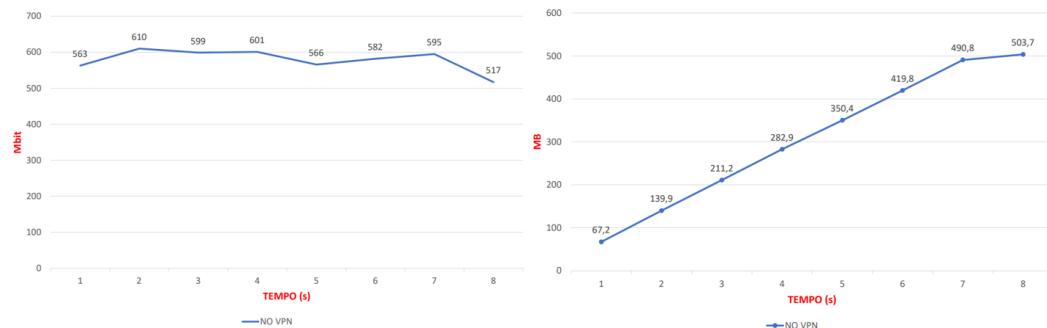
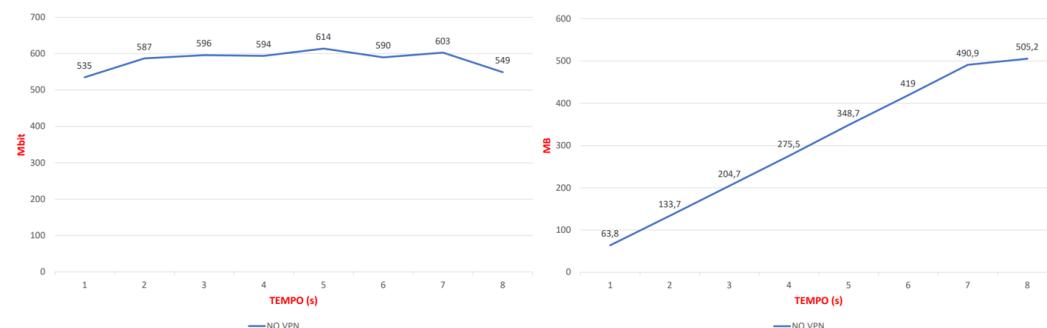
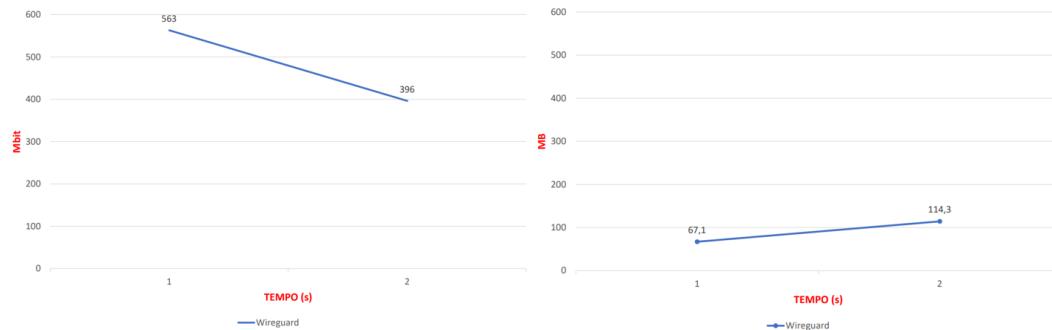
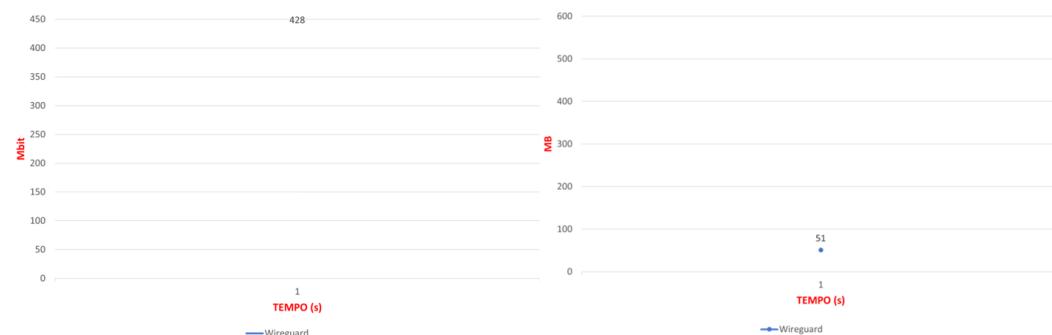
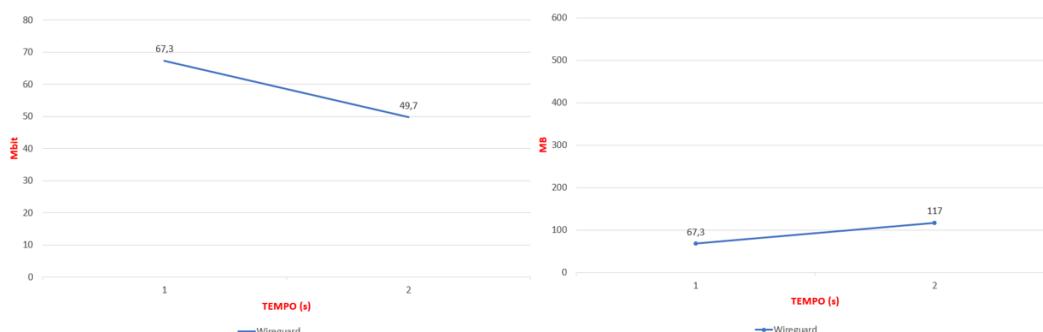


Figura 6.4: Grafici 5° esecuzione.



Grafici esperimento con WireGuard standard**Figura 6.6:** Grafici 1° esecuzione.**Figura 6.7:** Grafici 5° esecuzione.**Figura 6.8:** Grafici 10° esecuzione.

Grafici esperimento con WireGuard PQ

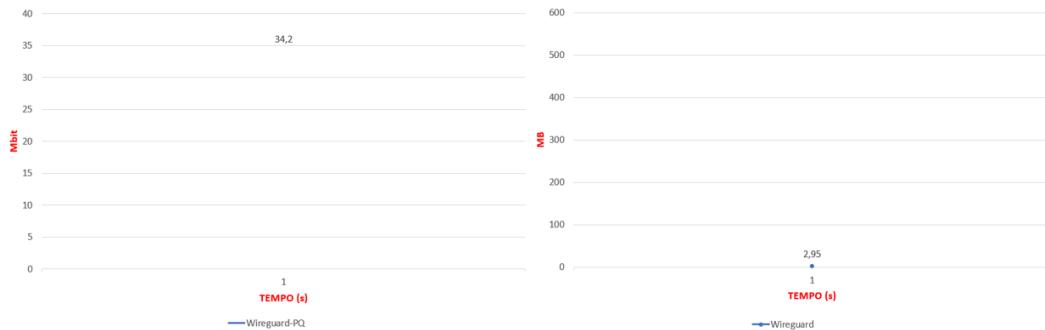


Figura 6.9: Grafici 1° esecuzione.



Figura 6.10: Grafici 5° esecuzione.

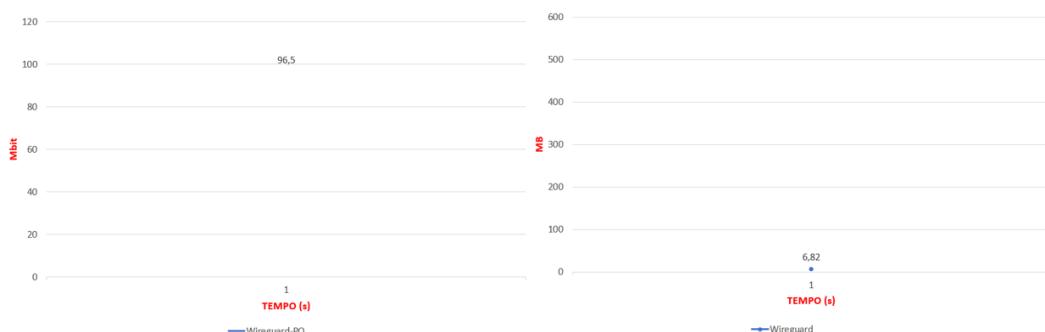


Figura 6.11: Grafici 10° esecuzione.

Grafici esperimento con OpenVPN

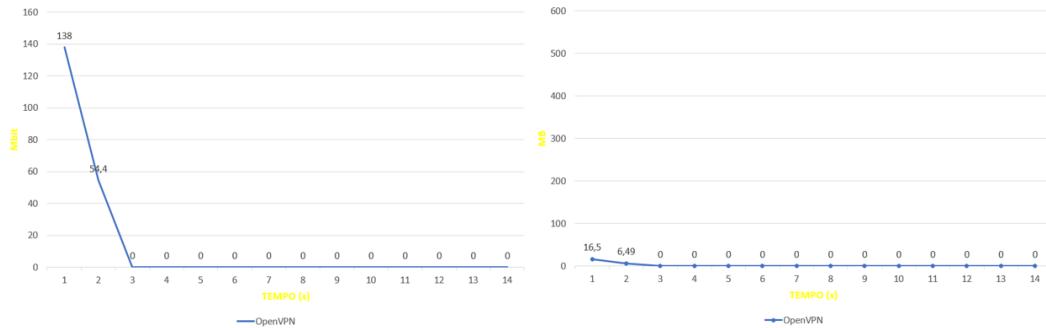


Figura 6.12: Grafici 1° esecuzione.

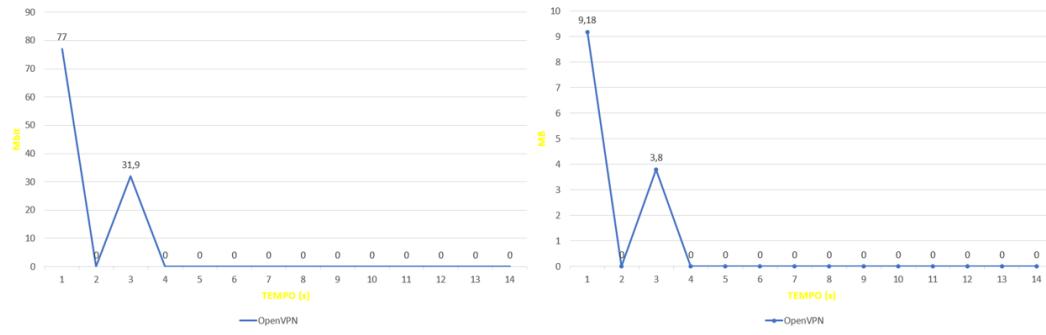


Figura 6.13: Grafici 5° esecuzione.

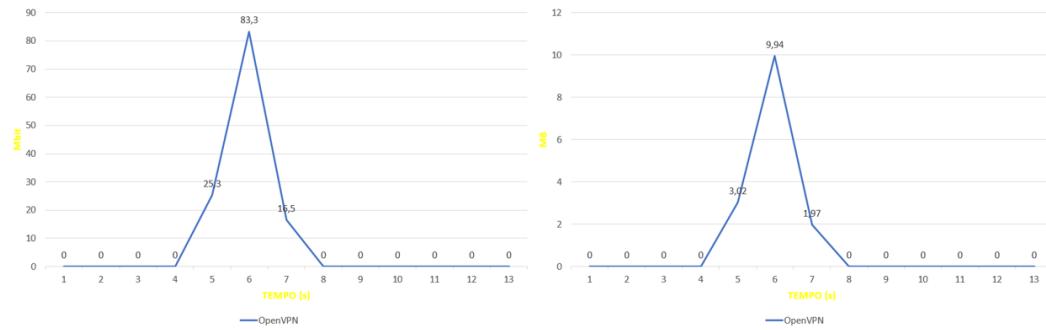


Figura 6.14: Grafici 10° esecuzione.

Grafici riassuntivi

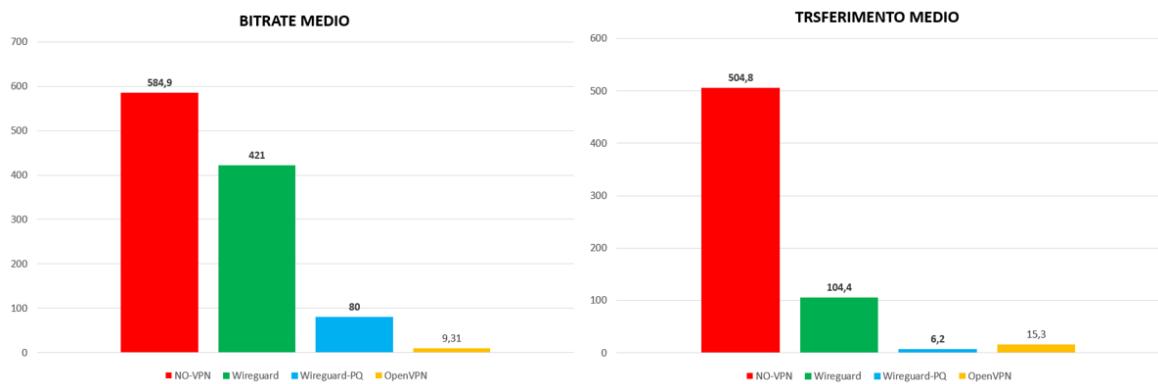


Figura 6.15: Grafici riassuntivi.

Esito primo esperimento

Dai grafici si può notare che ovviamente l'esperimento restituisce i risultati migliori quando nessun protocollo VPN è attivo; mentre tra le diverse VPN testate quella che si comporta meglio è sicuramente *WireGuard standard* il quale offre prestazioni migliori almeno in questo esperimento.

6.5.2 Secondo esperimento

Il secondo esperimento riguarda l'invio di numerosi pacchetti tramite l'utilizzo del protocollo TCP e di dieci connessioni parallele verso il server iPerf in modo da poter stabilire principalmente il throughput massimo raggiungibile.

Grafici esperimento senza VPN

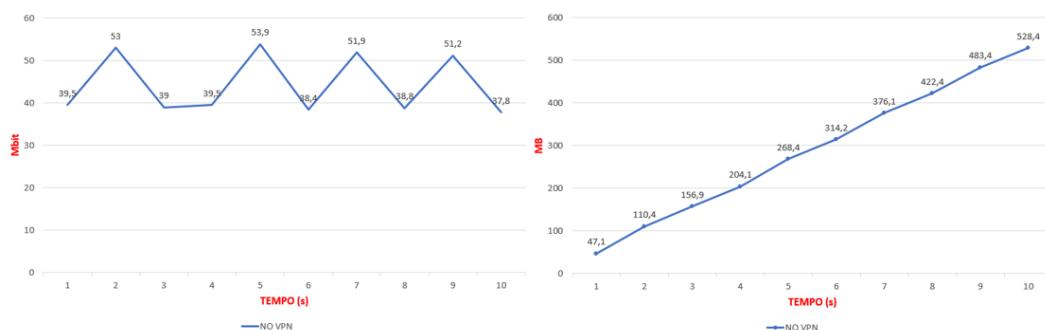


Figura 6.16: Grafici 1° esecuzione.

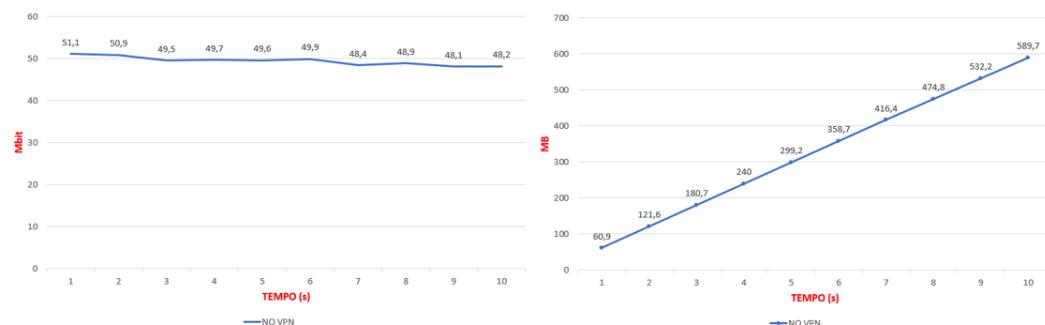


Figura 6.17: Grafici 5° esecuzione.

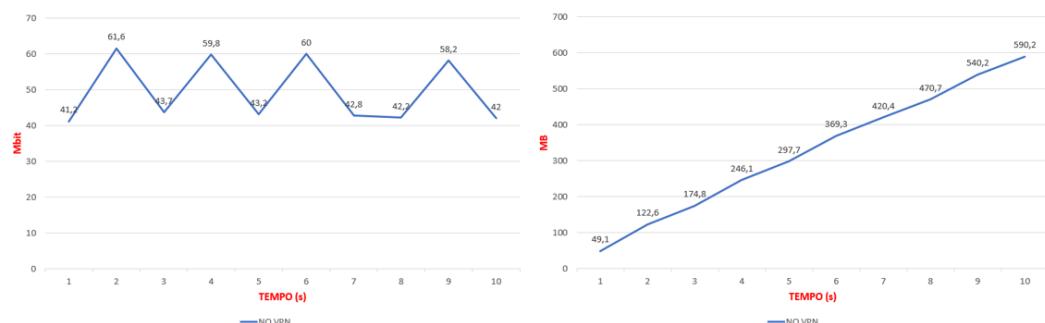
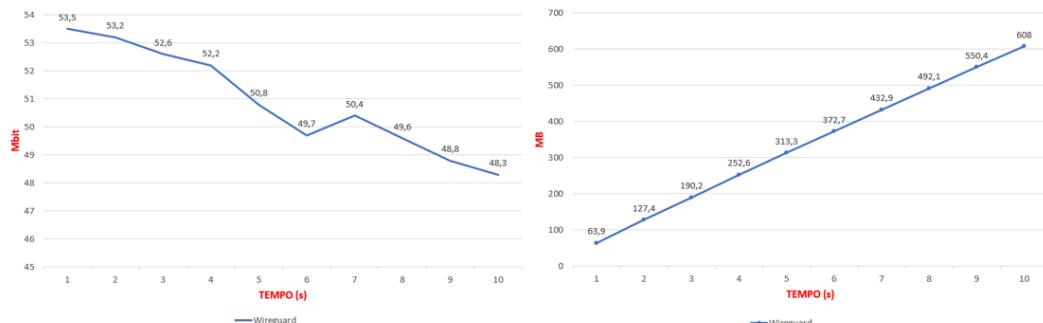
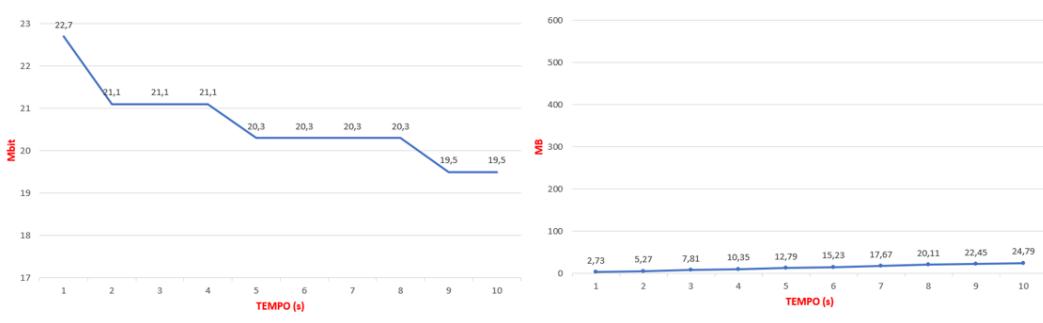
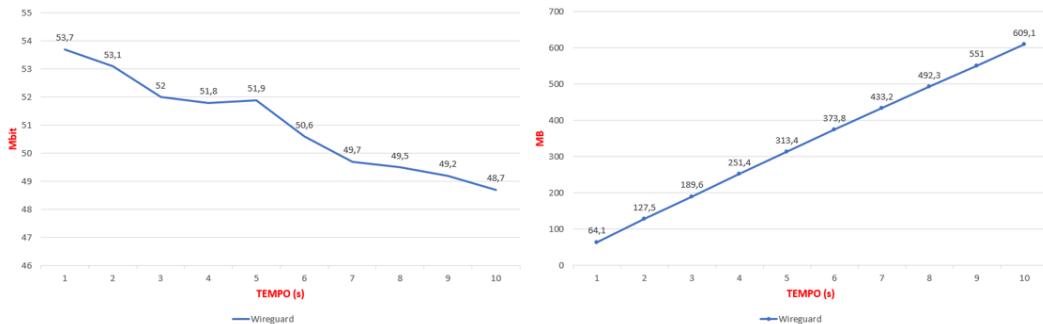


Figura 6.18: Grafici 10° esecuzione.

Grafici esperimento con WireGuard standard



Grafici esperimento con WireGuard PQ

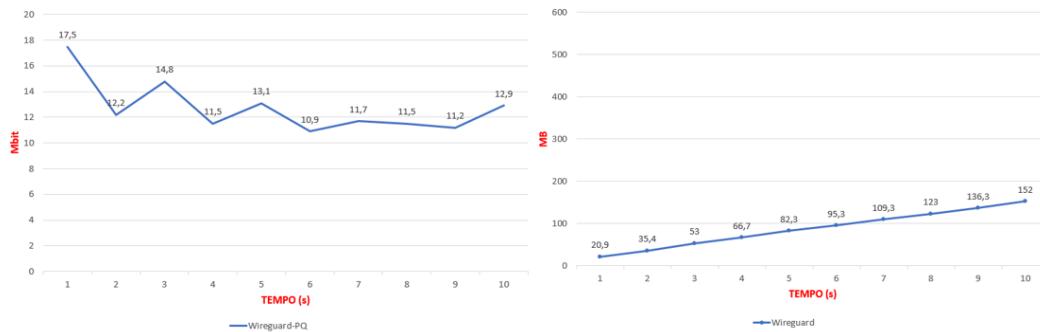


Figura 6.22: Grafici 1° esecuzione.

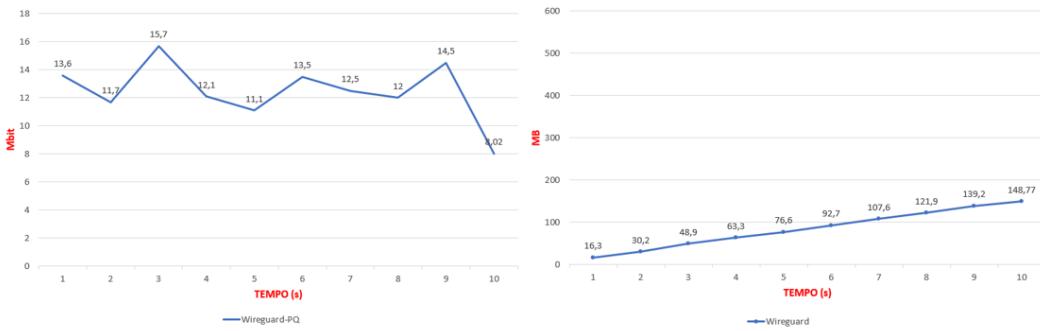


Figura 6.23: Grafici 5° esecuzione.

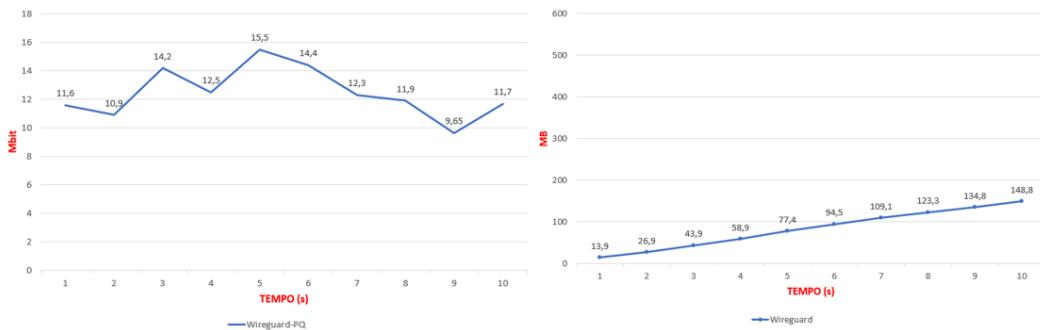


Figura 6.24: Grafici 10° esecuzione.

Grafici esperimento con OpenVPN

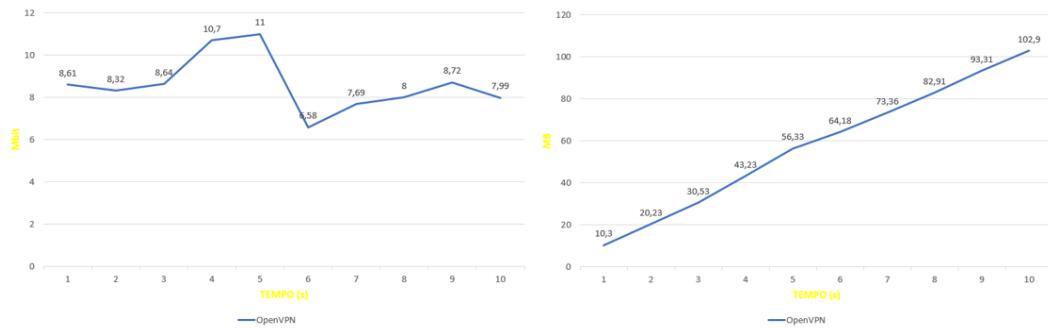


Figura 6.25: Grafici 1° esecuzione.

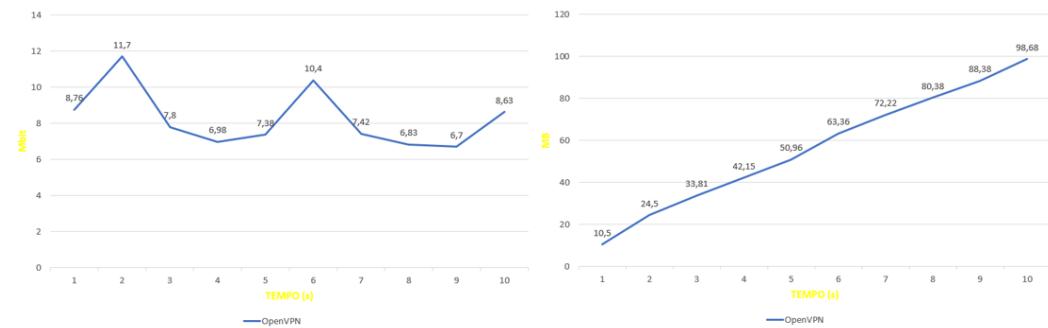


Figura 6.26: Grafici 5° esecuzione.

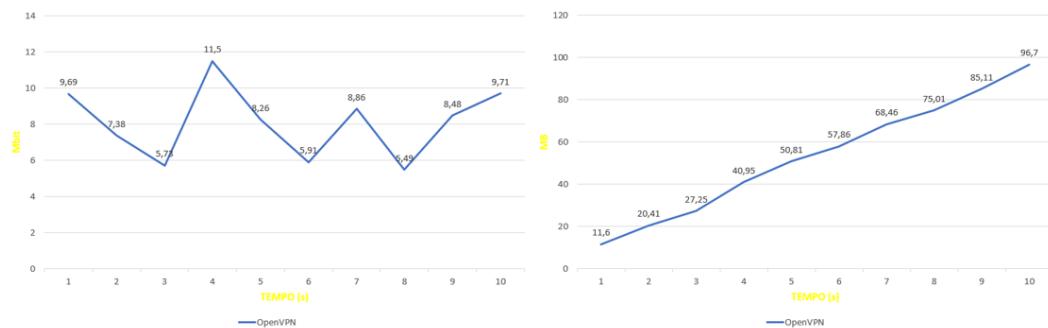


Figura 6.27: Grafici 10° esecuzione.

Grafici riassuntivi

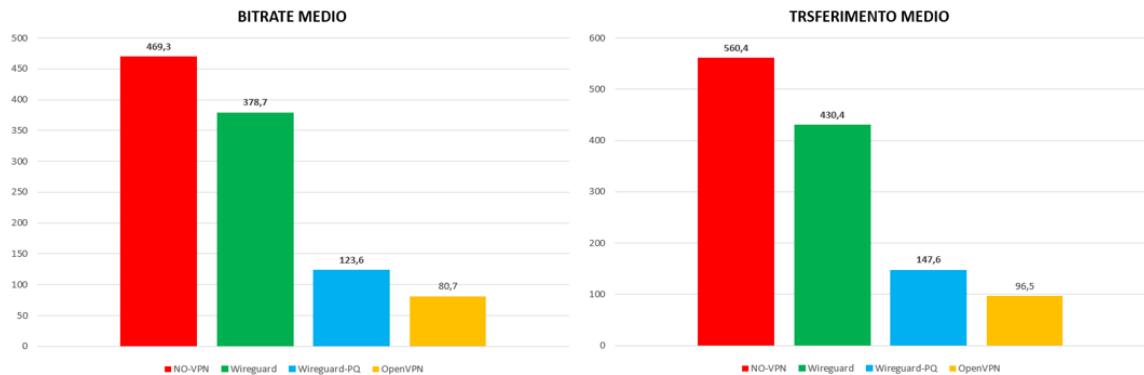


Figura 6.28: Grafici riassuntivi.

Esito secondo esperimento

Come nel primo esperimento, anche in questo caso è ovvio che l'esito sia a favore dell'esecuzione senza impiego di protocolli VPN anche se il divario con essi è abbastanza trascurabile, soprattutto se si considera l'output del test con l'utilizzo di *WireGuard standard*.

6.5.3 Terzo esperimento

Questo esperimento riguarda la simulazione di invio di un flusso di dati di livello *applicativo*, in particolare del protocollo VoIP.

Grafici esperimento senza VPN

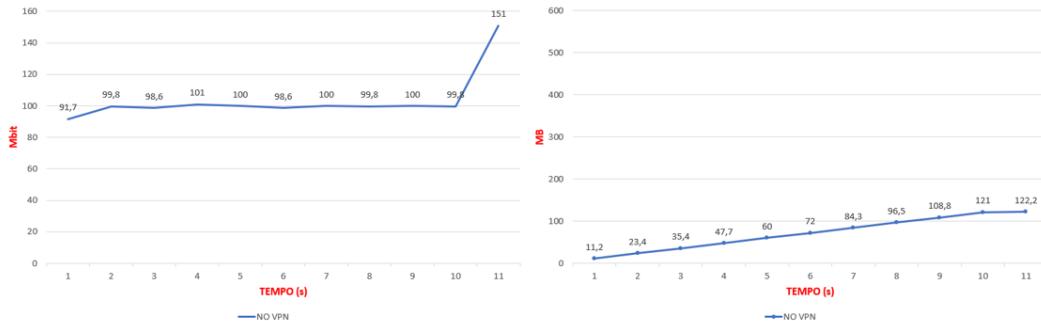


Figura 6.29: Grafici 1° esecuzione.

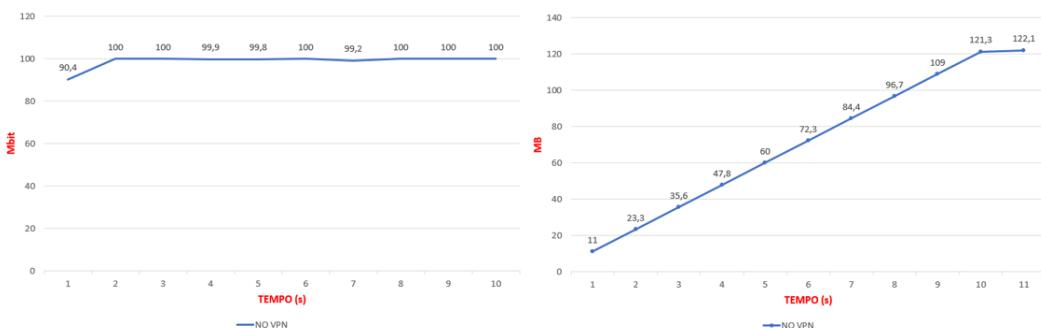


Figura 6.30: Grafici 5° esecuzione.

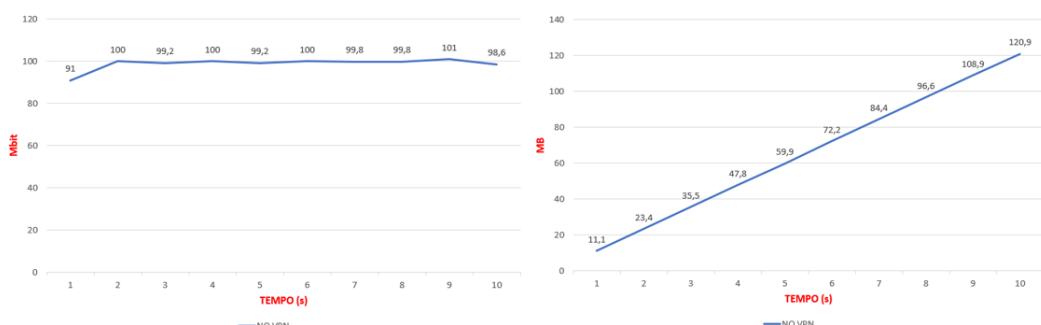
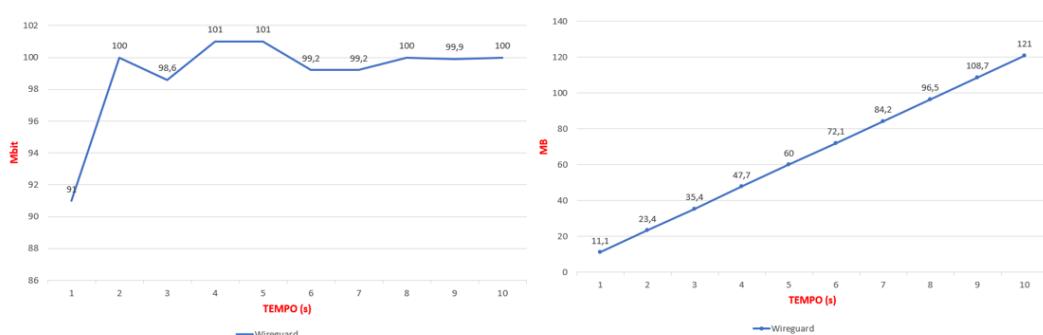
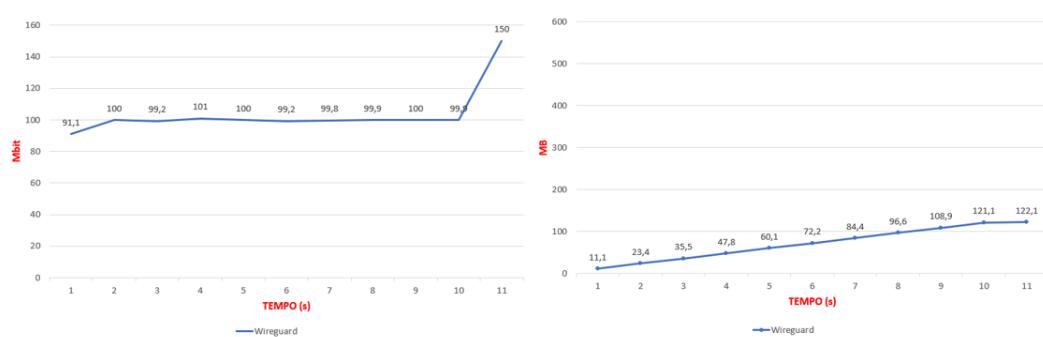
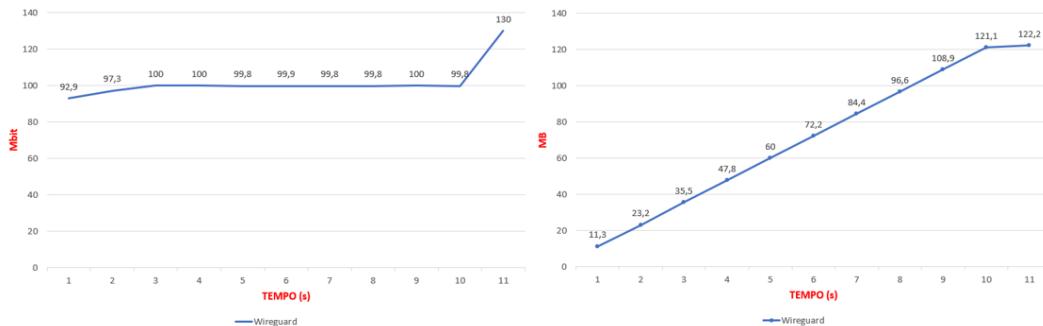


Figura 6.31: Grafici 10° esecuzione.

Grafici esperimento con WireGuard standard



Grafici esperimento con WireGuard PQ

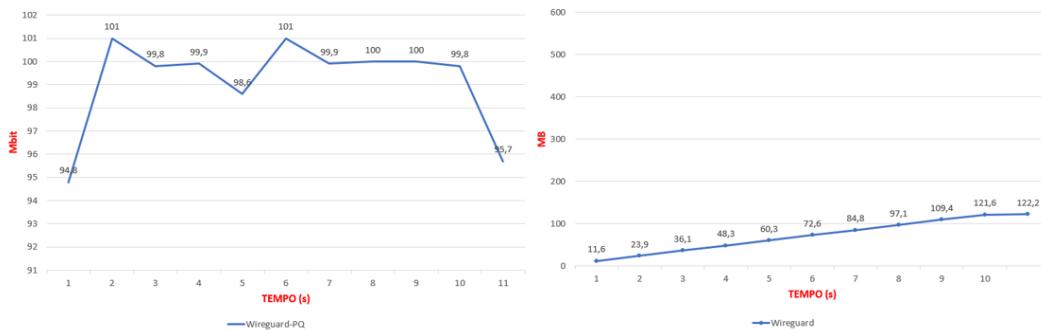


Figura 6.35: Grafici 1° esecuzione.

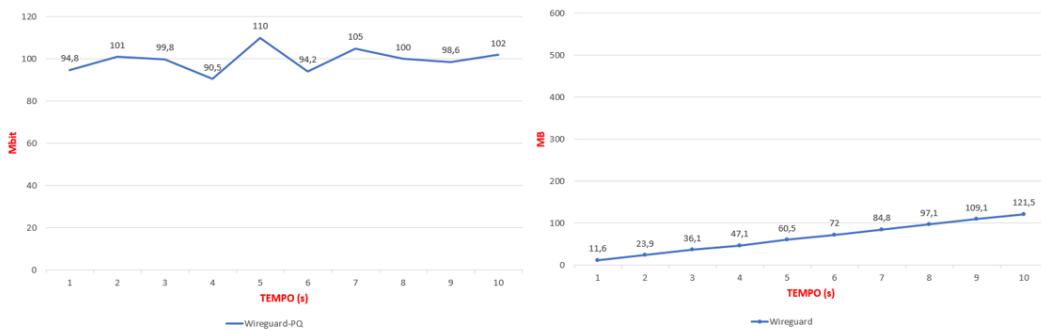


Figura 6.36: Grafici 5° esecuzione.

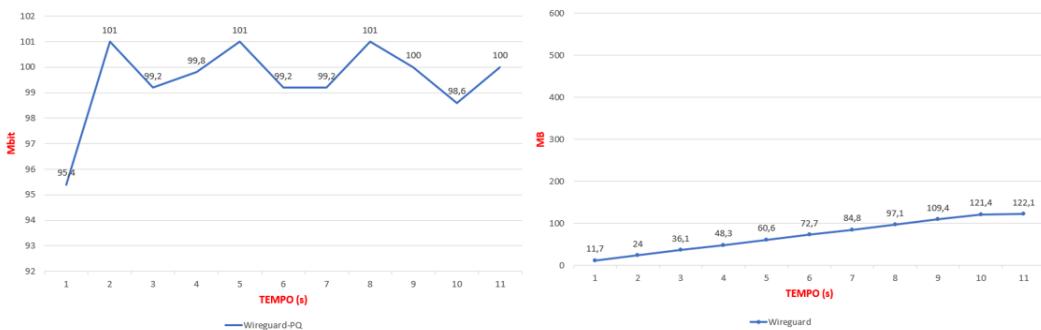


Figura 6.37: Grafici 10° esecuzione.

Grafici esperimento con OpenVPN

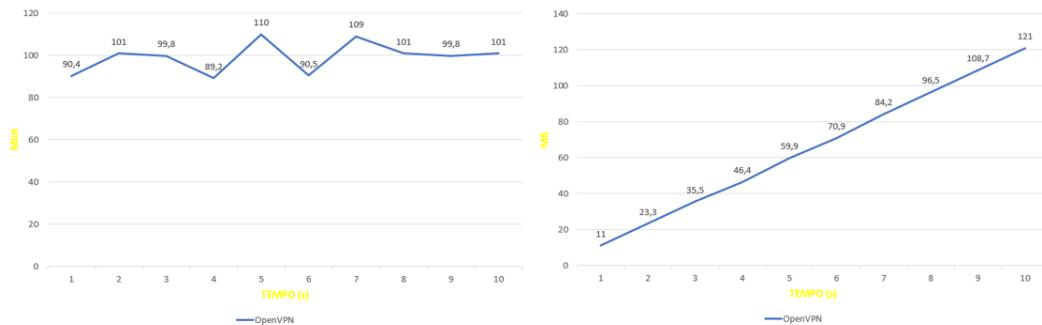


Figura 6.38: Grafici 1° esecuzione.

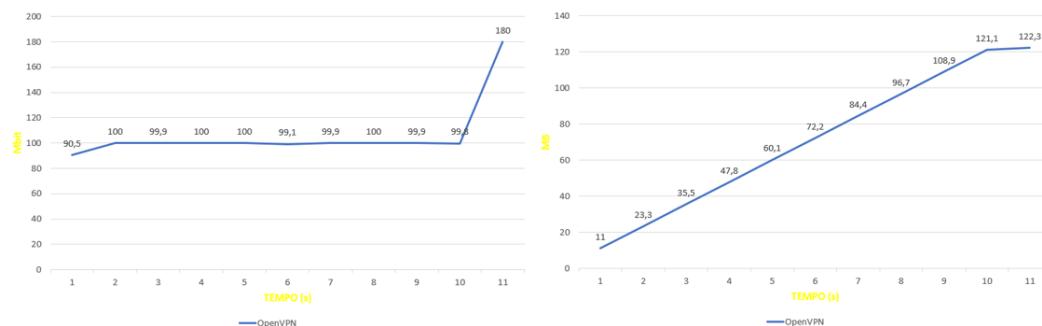


Figura 6.39: Grafici 5° esecuzione.

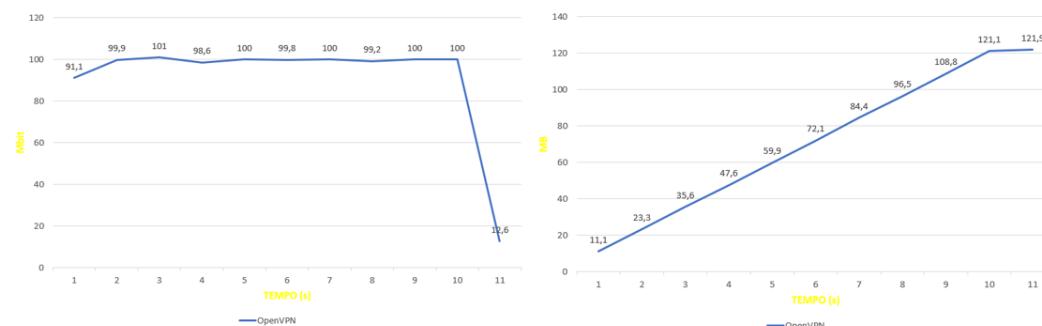


Figura 6.40: Grafici 10° esecuzione.

Grafici riassuntivi

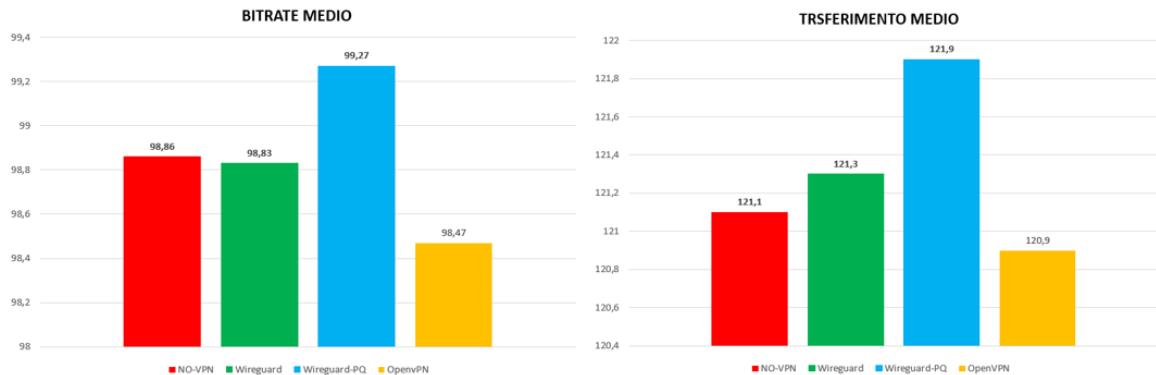


Figura 6.41: Grafici riassuntivi.

Esito terzo esperimento

Sorprendentemente nonostante il protocollo *WireGuard PQ* sia oggettivamente più complesso data la sua natura, riesce, se pur leggermente, a sovrastare gli altri protocolli VPN testati e a superare addirittura l'esito del test senza l'utilizzo di VPN.

6.5.4 Quarto esperimento

Questo esperimento è stato ideato con l'obiettivo di simulare l'invio di pacchetti TCP appartenenti ad un flusso di dati adibito allo streaming video, come ad esempio *Netflix*

Grafici esperimento senza VPN

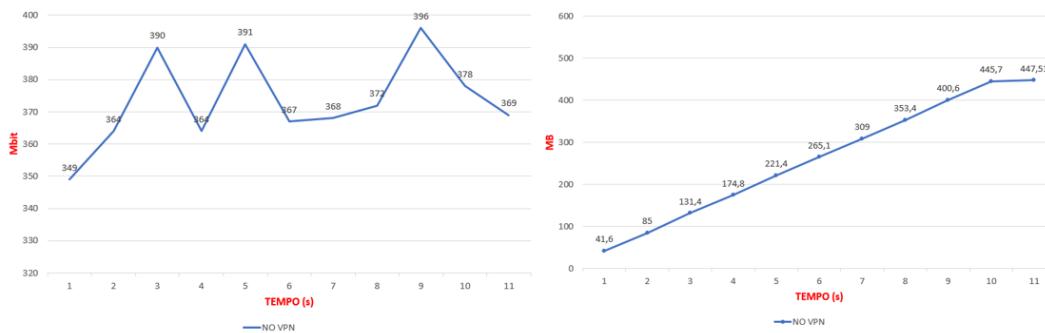


Figura 6.42: Grafici 1° esecuzione.

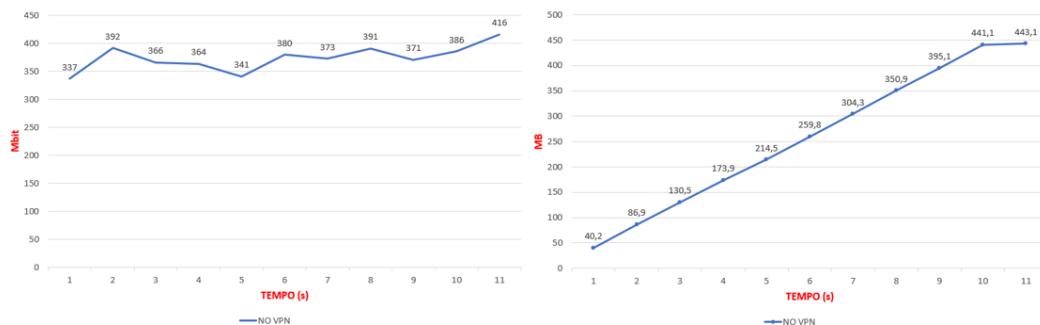


Figura 6.43: Grafici 5° esecuzione.

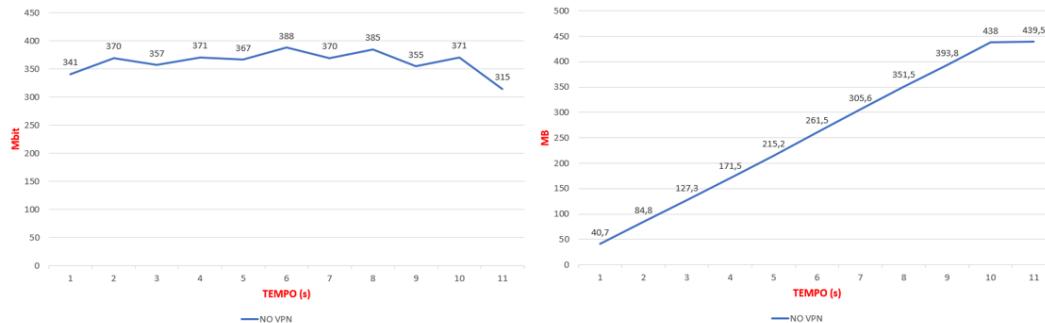


Figura 6.44: Grafici 10° esecuzione.

Grafici esperimento con WireGuard standard

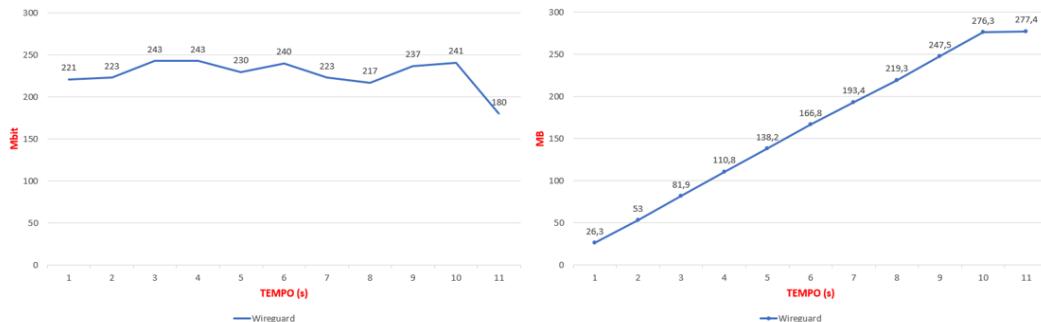


Figura 6.45: Grafici 1° esecuzione.

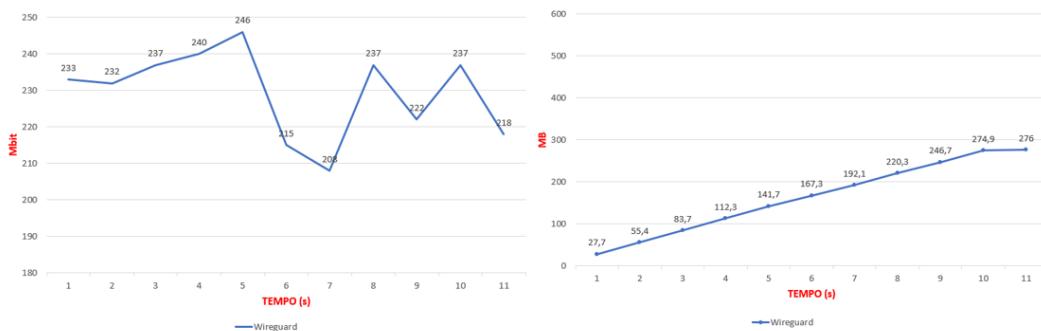


Figura 6.46: Grafici 5° esecuzione.

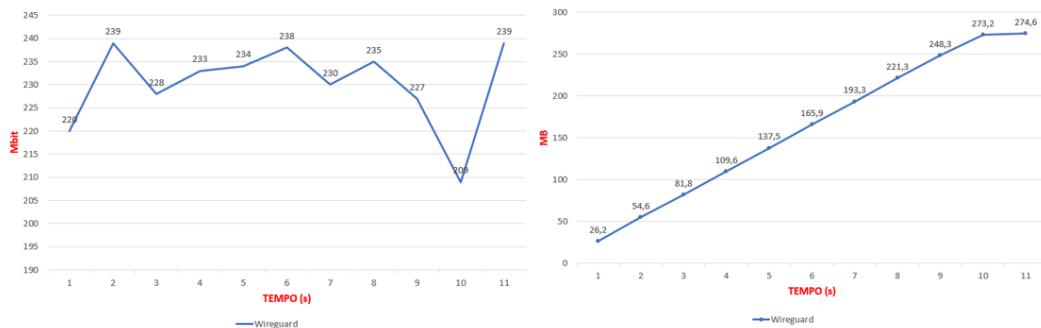
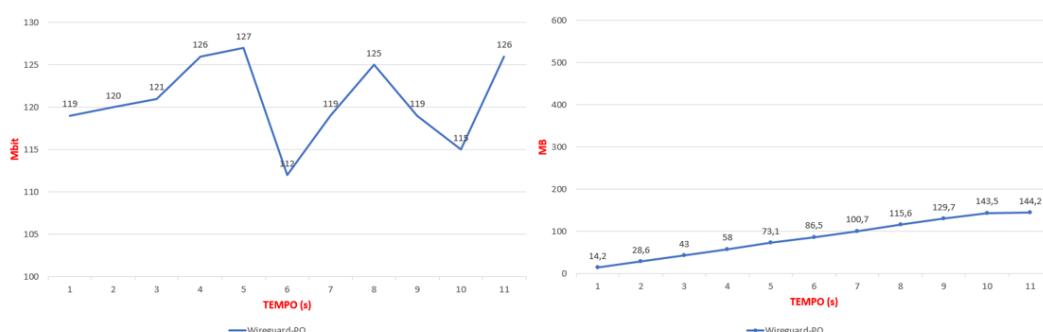
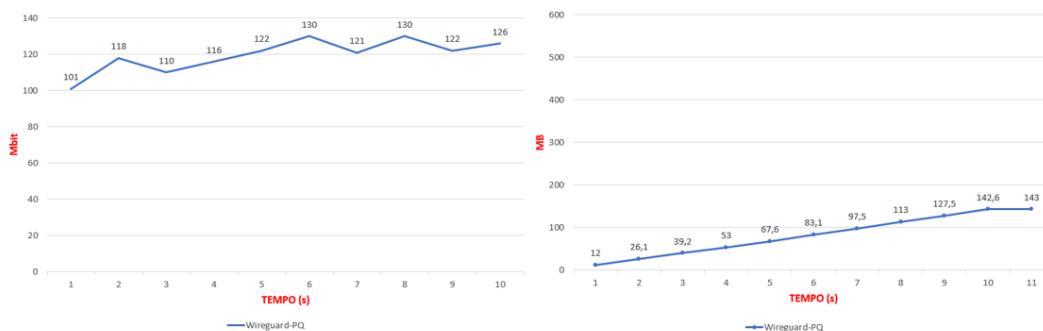
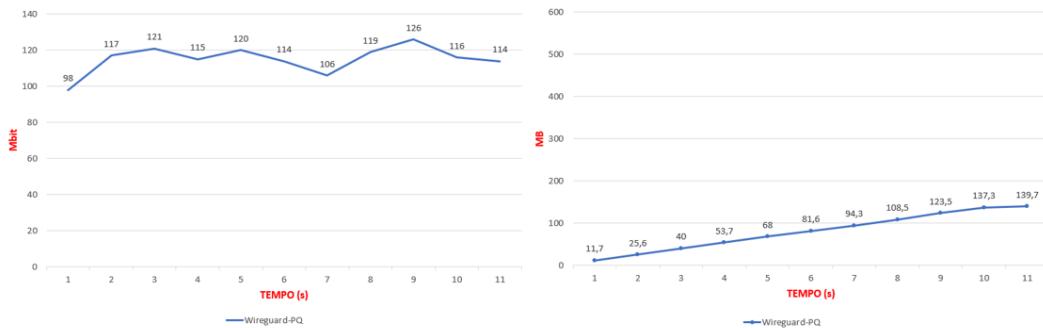


Figura 6.47: Grafici 10° esecuzione.

Grafici esperimento con WireGuard PQ



Grafici esperimento con OpenVPN

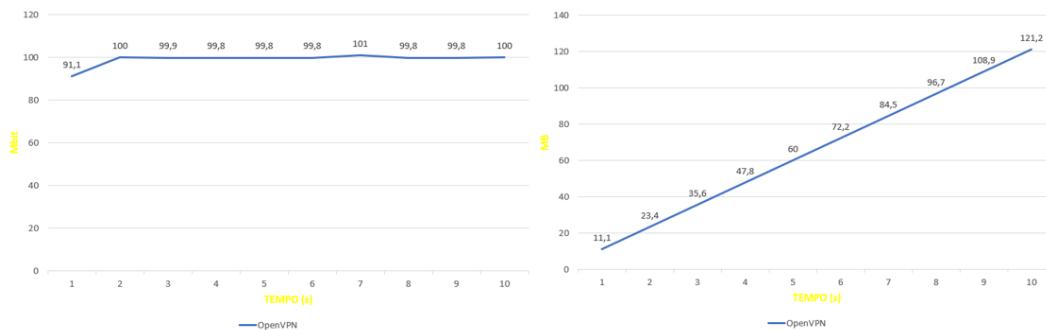


Figura 6.51: Grafici 1° esecuzione.

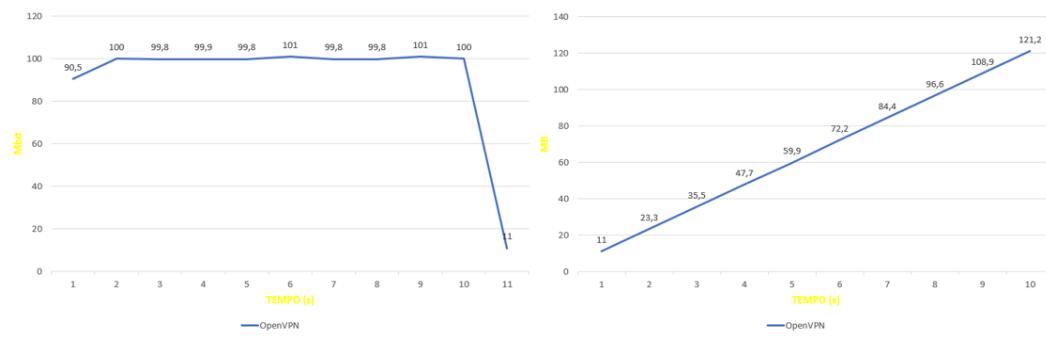


Figura 6.52: Grafici 5° esecuzione.

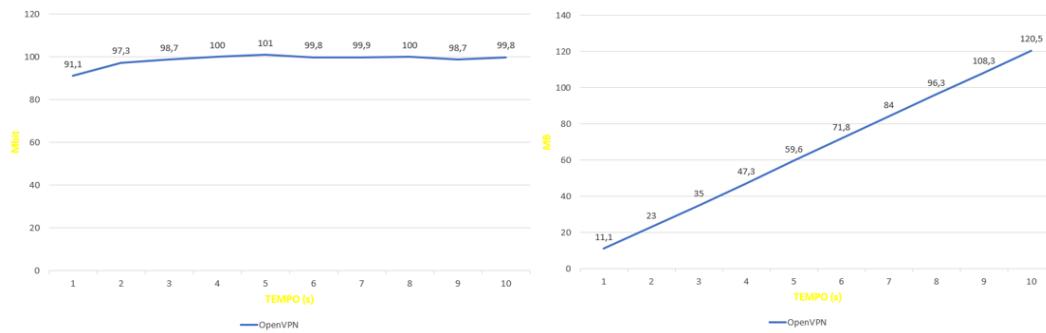


Figura 6.53: Grafici 10° esecuzione.

Grafici riassuntivi

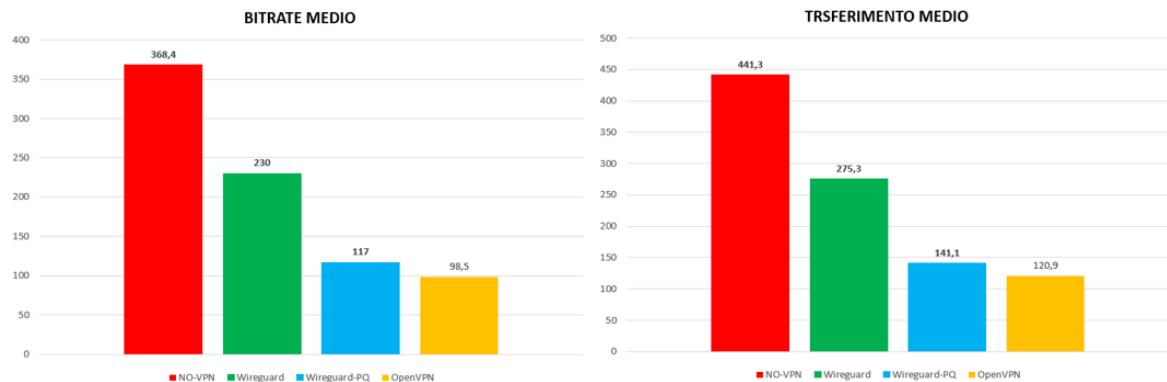


Figura 6.54: Grafici riassuntivi.

Esito quarto esperimento

Per quanto riguarda questo esperimento, il comportamento migliore è stato ottenuto senza l’impiego di protocolli VPN mentre con essi sono stati ottenuti dei risultati che dimostrano chiaramente che la differenza prestazionale è abbastanza ampia, soprattutto se si considerano i protocolli *WireGuard PQ* e *OpenVPN*.

6.5.5 Quinto esperimento

L'ultimo esperimento eseguito riguarda il trasferimento massivo di dati, in particolare i pacchetti appartengono ad un flusso di trasferimento massivo di tipo TCP.

Grafici esperimento senza VPN

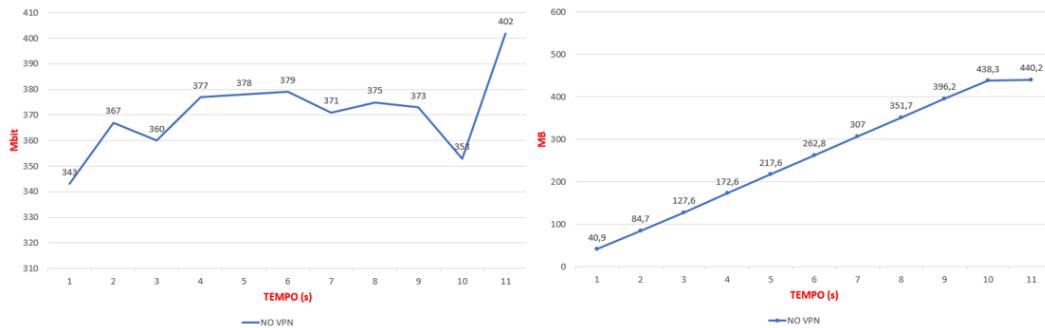


Figura 6.55: Grafici 1° esecuzione.

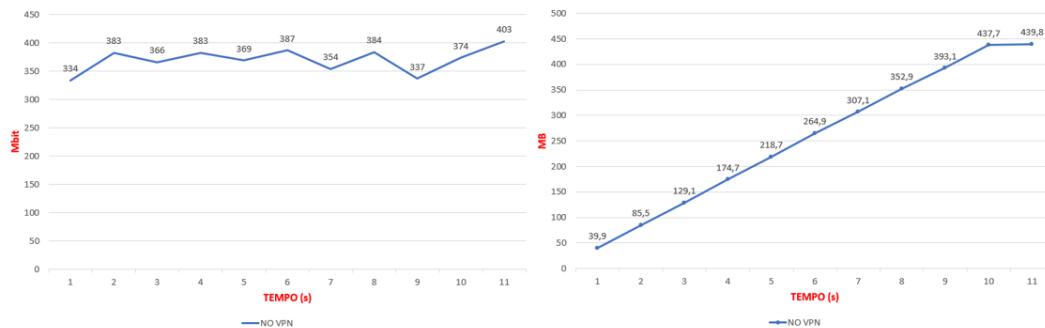


Figura 6.56: Grafici 5° esecuzione.

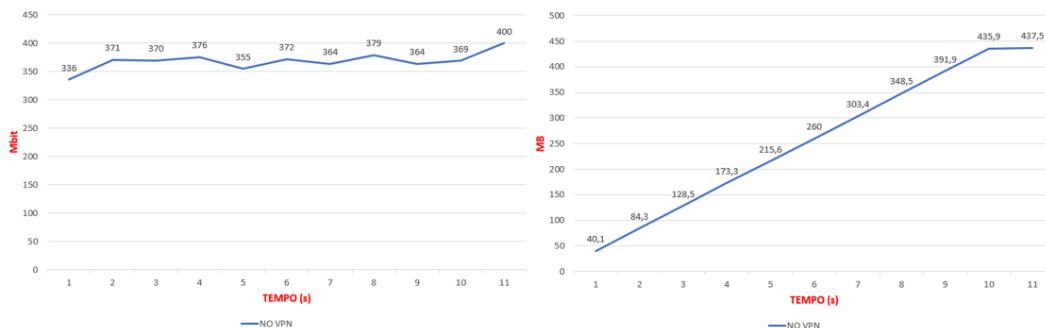
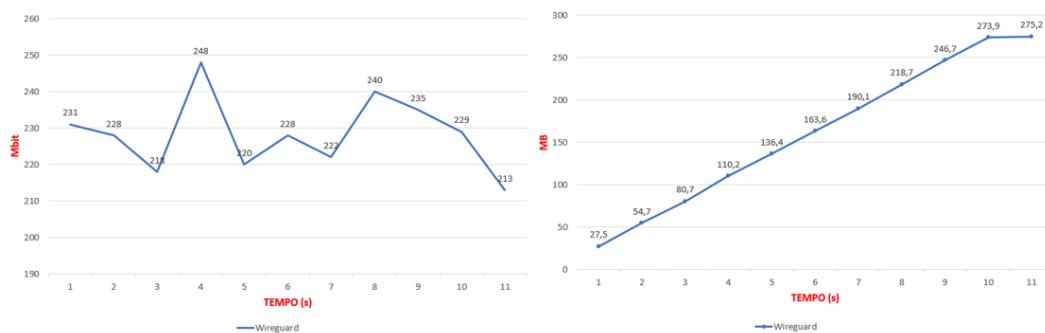
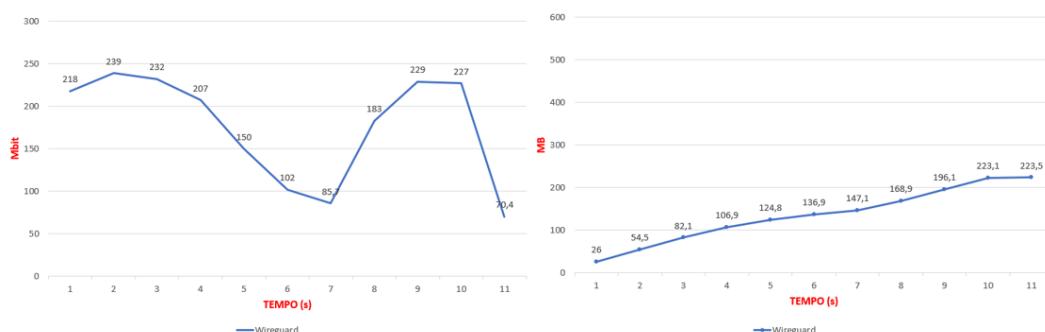
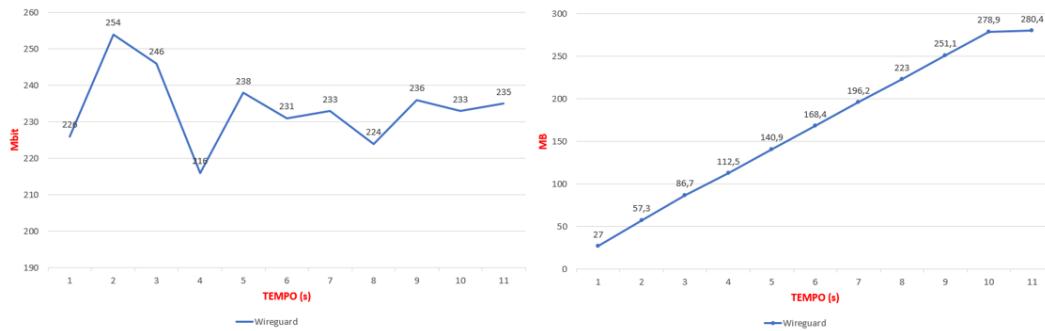


Figura 6.57: Grafici 10° esecuzione.

Grafici esperimento con WireGuard standard



Grafici esperimento con WireGuard PQ

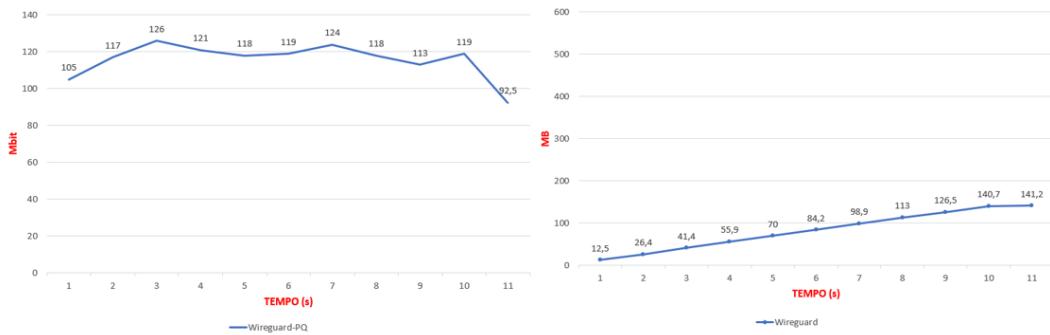


Figura 6.61: Grafici 1° esecuzione.

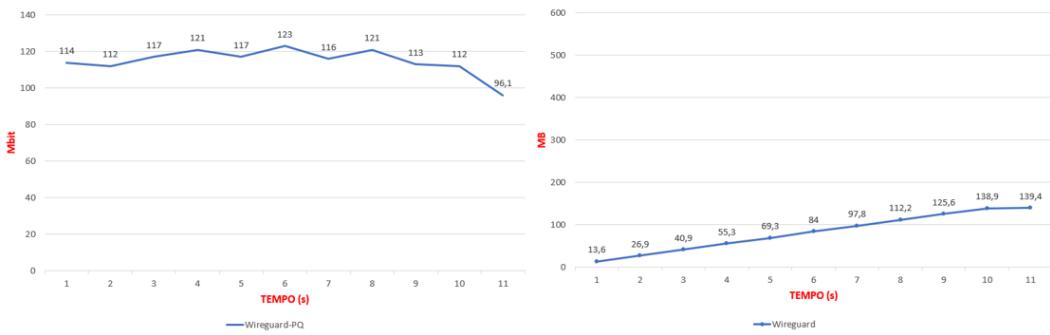


Figura 6.62: Grafici 5° esecuzione.

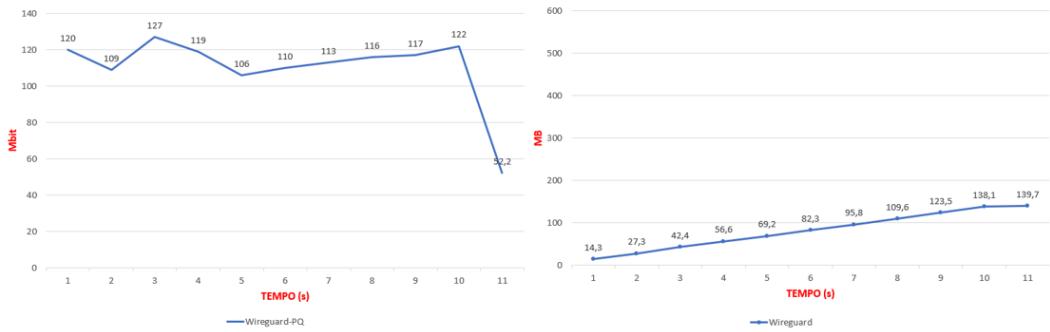
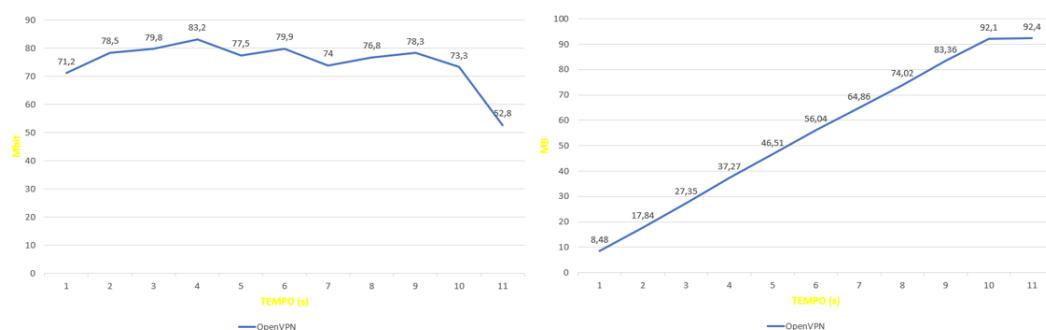
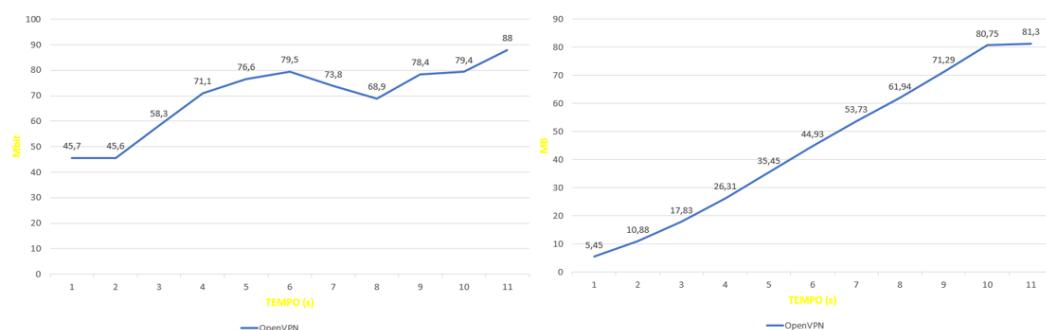
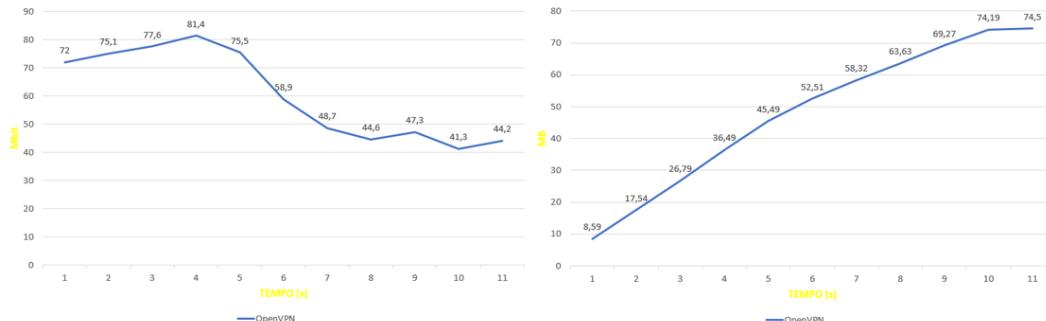


Figura 6.63: Grafici 10° esecuzione.

Grafici esperimento con OpenVPN



Grafici riassuntivi

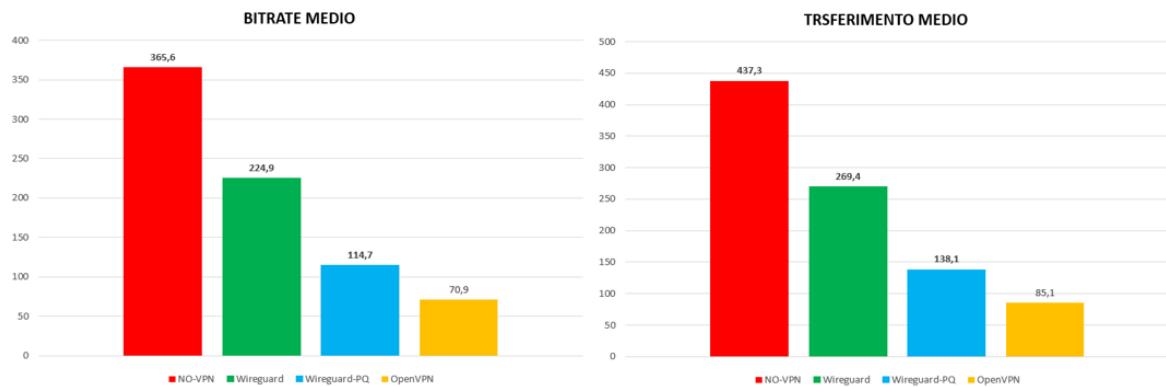


Figura 6.67: Grafici riassuntivi.

Esito quinto esperimento

Anche in questo caso, come per diversi altri esperimenti, è evidente che l'utilizzo di protocolli VPN possa influenzare notevolmente le prestazioni di rete disponibili, spesso causando un decremento notevole come ad esempio nel caso del test eseguito con il protocollo *OpenVPN*.

CAPITOLO 7

Conclusioni e sviluppi Futuri

In questa sezione, saranno riassunti i risultati ottenuti dal sistema realizzato, ed inoltre, saranno discussi i potenziali sviluppi che potrebbero essere intrapresi in futuro.

7.1 Risultati ottenuti

Una volta che il sistema è stato implementato correttamente, tutti i test sono stati portati avanti come spiegato nel capitolo precedente in modo da poter stabilire con sicurezza il *comportamento* di ognuno dei protocolli VPN impiegati affinché sia chiaro quale sia quello più adatto da utilizzare in relazione anche al grado di *sicurezza* che si vuole ottenere. Come si è potuto osservare dai risultati prodotti dalle esecuzioni degli esperimenti, ognuno di essi ha dato un output diverso condizionato principalmente dal fatto che alla base di ogni tentativo eseguito è presente un flusso di dati differente per ognuno di essi.

7.1.1 Valutazione dei risultati

Osservando i grafici e quindi gli output relativi alle esecuzioni degli esperimenti, si può facilmente notare che per ognuno di essi il risultato è distinto, in particolare:

- *Esperimento 1*: WireGuard standard è il protocollo VPN a comportarsi meglio;
- *Esperimento 2*: WireGuard standard è il protocollo VPN a comportarsi meglio;
- *Esperimento 3*: WireGuard PQ è il protocollo VPN a comportarsi meglio;
- *Esperimento 4*: WireGuard standard è il protocollo VPN a comportarsi meglio;
- *Esperimento 5*: Wireguard standard è il protocollo VPN a comportarsi meglio.

Dall’elenco superiore si intuisce che in linea di massima il protocollo *WireGuard* si comporta meglio del protocollo *OpenVPN* con questi precisi esperimenti eseguiti; in particolare la versione *standard* risulta offrire le migliori prestazioni mentre la versione *PQ* ha restituito in output dei risultati che sono perfettamente paragonabili a quelli ottenuti utilizzando OpenVPN, il quale conferma che nonostante l’*overhead* aggiunto a causa degli algoritmi *Post Quantum* impiegati da WireGuard PQ esso risulta comunque prestazionalmente paragonabile ad un protocollo VPN che impiega algoritmi convenzionali.

7.2 Contributi della ricerca

Arrivati a questo punto del percorso affrontato, è importante notare qual’è il *contributo* effettivo che il presente progetto può offrire alla ricerca ed allo sviluppo di soluzioni post quantum inerenti al trasferimento di flussi di dati aventi determinate esigenze in termini di rete. Il lavoro portato avanti ha consentito di realizzare uno strumento che al giorno d’oggi

può essere di grande aiuto soprattutto per quei settori della ricerca che sono strettamente correlati allo sviluppo di nuovi algoritmi post quantum ed al trasferimento sicuro di dati. La testbed realizzata offre la possibilità di poter eseguire numerosi test e di poter integrare anche altri algoritmi in modo da poter effettuare una comparazione dei protocolli VPN ancora più ampia ed esaustiva. Inerentemente ai flussi di dati trasferiti, il tool *Iperf* ha consentito di poter simulare in modo preciso e rapido l'insieme di flussi impiegati durante gli esperimenti, ma offre la possibilità di poterne simulare tanti altri così da approfondire le performance e/o testare altri parametri di rete come il *packet loss*.

7.3 Sviluppi futuri

Analizziamo ora alcune delle possibili strade che potrebbero essere intraprese al fine di apportare delle migliorie al lavoro svolto, sia per quanto riguarda le funzionalità realizzate che lo studio portato avanti.

7.3.1 Ulteriori esperimenti

Come accennato nel paragrafo precedente, in futuro potrebbe essere sicuramente utile progettare ed eseguire diversi altri esperimenti i quali fanno riferimento a differenti flussi di dati aventi requisiti diversi da quelli previsti in questo lavoro. Questo tipo di sviluppo potrebbe fare ulteriore chiarezza sui protocolli VPN in modo da stabilire con certezza quali di essi sono più adatti a determinati scopi.

7.3.2 Infrastruttura di rete

In relazione alle performance ottenute dai protocolli VPN testati tramite gli esperimenti eseguiti, potrebbe essere interessante ripetere questi ultimi fornendo alla testbed delle *connessioni di rete* diverse da quella impiegata in questo lavoro così da poter analizzare i risultati anche in relazione al tipo di connessione che si sta utilizzando per connettere la testbed in rete.

7.3.3 Realizzazione interfaccia web

Le funzionalità proposte agli utenti che intendono farne uso, in particolare il modulo inerente all'esecuzione delle scansioni di rete tramite l'uso della libreria *Nmap*, sono state pensate per essere sfruttate tramite delle *richieste HTTP* eseguite dagli host presenti in rete

LAN; a tale proposito sarebbe efficace sviluppare una *GUI* il quale semplificherebbe di gran lunga l'esecuzione delle richieste, soprattutto ad un pubblico che non ha molta dimestichezza con la generazione delle richieste manualmente o banalmente per avvantaggiare la fruizione delle funzionalità stesse.

Ringraziamenti

Con la realizzazione di quest'ultimo lavoro termina la mia carriera accademica il quale oltre ad avermi permesso di approfondire la mia principale passione e di poterne farne un lavoro, in futuro, mi ha fatto capire fino in fondo cosa significhi avere vicino le giuste persone ed il conseguente impatto che influisce a lungo termine sulla vita quotidiana. Il percorso accademico affrontato è stato avvolto da un contesto non proprio ideale ad esso ed è per tale motivo che intendo ringraziare le giuste persone che hanno contribuito positivamente a migliorarlo.

Ringrazio innanzitutto il *Prof. Arcangelo Castiglione*; ha sempre riposto le giuste attenzioni allo sviluppo del lavoro proposto e alla risoluzione immediata dei problemi riscontrati lungo il percorso. Nutro nei suoi confronti un grande senso di stima in particolare per la dedizione che ripone nel suo lavoro.

Ringrazio *Marco*; oltre a fornire costante supporto sia al progetto che morale, hai creduto sin dall'inizio in me dandomi la possibilità di poter collaborare con te e gli altri colleghi di *BSQ Security*. Grazie a te ho potuto apprendere nuove conoscenze ed un ottimo approccio al mondo lavoro.

Ringrazio *Lorenzo*; ti sei rivelato un sincero amico oltre che affidabile compagno di tanti avvenimenti; ho sempre potuto contare su di te nei momenti in cui avevo bisogno e per questo ti sono riconoscente. Spero che quest'amicizia possa continuare ancora per molto tempo nonostante gli inconvenienti che incontreremo sulle nostre strade.

Ringrazio *Vincenzo*; con te ho trascorso gran parte dell'ultimo anno accademico durante il quale ho imparato a conoserti meglio; con il tempo ho capito che siamo molto più simili di quanto avremmo mai potuto dire qualche anno fa e questo mi ha dato la possibilità di confrontarmi spesso con te su diversi argomenti. Ti ringrazio per le lunghe chiacchierate e per avermi sempre ascoltato seriamente.

Ringrazio *Carmen*; per avermi offerto da sempre il tuo supporto soprattutto nelle situazioni più spinose, grazie a te ho trovato una vera amica capace di dare il giusto sostegno in qualsiasi circostanza e di ascoltare sempre in maniera garbata, talvolta strappando anche una risata.

Ringrazio *Simone*; nonostante quest'ultimo periodo sia stato abbastanza movimentato per entrambi hai da sempre trovato il modo di offrire la giusta attenzione al nostro rapporto e di esserci attivamente per qualsiasi situazione. Confido nel fatto che la nostra amicizia possa proseguire ancora per molto tempo.

Ringrazio i miei *parenti*; per avermi sempre sostenuto senza intralciare le mie scelte ed al contempo dare il sostegno necessario per intraprenderle. Siete stati di fondamentale importanza per il raggiungimento di un obiettivo così importante per me.

Ringrazio *Carmen P.*; sei e sarai il mio punto di riferimento sulla quale posso fare costantemente affidamento in qualsiasi situazione. A te va il mio più sincero ringraziamento.

Infine, ringrazio tutti i miei *amici* e quelle persone che hanno avuto la premura di compiere anche una singola azione positiva nei miei confronti, spronandomi a continuare il mio percorso.

Vi sarò per sempre riconoscente.

Bibliografia

- [1] "Testbed," 2023. <https://en.wikipedia.org/wiki/Testbed>. (Citato a pagina 2)
- [2] "Differenza tra larghezza di banda e velocità nelle telecomunicazioni," 2023. <https://vitolavecchia.altervista.org/differenza-tra-larghezza-di-banda-e-velocita-nelle-telecomunicazioni/>. (Citato a pagina 2)
- [3] M. Y. George Braine, "Local area network (lan) computers in esl and efl writing classes: Promises and realities," p. 2, 1998. <https://jalt-publications.org/files/pdf-article/jj-20.2-art3.pdf>. (Citato a pagina 5)
- [4] M. W. H. Y. Yan Zhang, Nirwan Ansari, "Ieee communications surveys tutorials (volume: 14, issue: 4, fourth quarter 2012," p. 1, 2002. <https://ieeexplore.ieee.org/abstract/document/6042388>. (Citato a pagina 5)
- [5] G. H. Paul Ferguson, "What is a vpn?," p. 1, 1998. <http://sol.te.net.ua/www/nanog/vpn.pdf>. (Citato a pagina 5)
- [6] P. S. F. W. P. R. Z. Andreas Hülsing, Kai-Chun Ning, "Post-quantum wireguard," p. 1, 2021. <https://ieeexplore.ieee.org/document/9519445>. (Citato a pagina 5)
- [7] P. S. F. W. P. R. Z. Andreas Hülsing, Kai-Chun Ning, "Post-quantum wireguard," p. 1, 2021. <https://ieeexplore.ieee.org/document/9519445>. (Citato a pagina 6)
- [8] "Adding quantum resistance to wireguard," 2021. <https://research.kudelskisecurity.com/2021/07/08/adding-quantum-resistance-to-wireguard/>. (Citato a pagina 6)
- [9] E. K. T. L. V. L. J. M. S. P. S. G. S. D. S. Joppe Bos, Léo Ducas, "Crystals – kyber: a cca-secure module-lattice-based kem," p. 1, 2018. <https://ieeexplore.ieee.org/document/8406610>. (Citato a pagina 6)

- [10] C. R. M. R. G. Logan O. Mailloux, Charlton D. Lewis II, "Post-quantum cryptography what advancements in quantum computing mean for it professionals," p. 3, 2016. <https://ieeexplore.ieee.org/document/7579104>. (Citato a pagina 6)
- [11] "Cos'è il kernel di linux?," 2019. <https://www.redhat.com/it/topics/linux/what-is-the-linux-kernel>. (Citato a pagina 7)
- [12] "Architettura arm," 2019. https://it.wikipedia.org/wiki/Architettura_ARM. (Citato a pagina 7)
- [13] "Che cosa sono i container?," 2019. <https://cloud.google.com/learn/what-are-containers?hl=it>. (Citato a pagina 7)
- [14] "Docker compose overview," 2023. <https://docs.docker.com/compose/>. (Citato alle pagine 7 e 24)
- [15] "Cos'è un broker di messaggi?," 2023. <https://www.ibm.com/it-it/topics/message-brokers>. (Citato a pagina 8)
- [16] "Raspberry pi 4 model b," 2023. <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>. (Citato a pagina 17)
- [17] "Nmap," 2023. <https://nmap.org/>. (Citato a pagina 19)
- [18] "Python," 2023. <https://www.python.org/>. (Citato a pagina 19)
- [19] "Flask," 2023. <https://flask.palletsprojects.com/en/3.0.x/>. (Citato a pagina 19)
- [20] "Wireguard," 2023. <https://www.wireguard.com/>. (Citato a pagina 19)
- [21] "Openvpn," 2023. <https://openvpn.net/>. (Citato a pagina 19)
- [22] "Progetto github," 2023. <https://github.com/99orazio/Tesi-Magistrale>. (Citato alle pagine 22, 23, 29 e 30)
- [23] "Duck dns free dynamic dns hosted on aws," 2023. <https://www.duckdns.org/>. (Citato a pagina 24)
- [24] "Raspberry pi imager," 2023. <https://www.raspberrypi.com/software/>.