



UNIVERSITÀ DEGLI STUDI DI ROMA

TOR VERGATA

---

## DDos Mitigation System Simulation

---

Chiara IURATO  
chiara.iurat@gmail.com

Alessandro CORTESE  
ale.cortese99@gmail.com

# Contents

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Scopi e Obiettivi</b>	<b>3</b>
<b>3</b>	<b>Modello Concettuale</b>	<b>4</b>
3.1	Eventi del Sistema . . . . .	5
3.1.1	Sistema . . . . .	5
3.1.2	Mitigation Server . . . . .	5
3.1.3	Web Server . . . . .	6
3.1.4	Spike Server . . . . .	6
3.2	Stati del Sistema . . . . .	6
<b>4</b>	<b>Modello delle Specifiche</b>	<b>7</b>
4.1	Stime dei costi . . . . .	7
4.2	Modellazione dei Centri . . . . .	8
<b>5</b>	<b>Modello Computazionale</b>	<b>9</b>
5.1	Gestione degli Eventi . . . . .	10
<b>6</b>	<b>Verifica</b>	<b>16</b>
6.1	Parametri del modello . . . . .	16
6.2	Spazio degli stati . . . . .	16
6.3	Transizioni del processo . . . . .	17
6.4	Risoluzione della Catena di Markov . . . . .	17
6.5	Risultati Ottenuti . . . . .	18
<b>7</b>	<b>Validazione</b>	<b>19</b>
7.1	Mitigation Center . . . . .	19
7.2	Web Server . . . . .	21
7.3	Spike Server . . . . .	23
7.4	Numero di Spike Server allocati . . . . .	24
<b>8</b>	<b>Progettazione degli esperimenti</b>	<b>26</b>
8.1	Intervallo di Confidenza . . . . .	26
8.2	Analisi del Transitorio . . . . .	26
8.3	Simulazione Orizzonte Finito . . . . .	30
8.4	Simulazione Orizzonte Infinito . . . . .	37
8.4.1	Autocorrelazione . . . . .	41
<b>9</b>	<b>Modello Migliorativo</b>	<b>42</b>
9.1	Considerazione e Limitazione del Modello di Base . . . . .	42
9.2	Modello Concettuale . . . . .	43
9.3	Eventi del Sistema . . . . .	43
9.3.1	Machine Learning Analysis . . . . .	43
9.4	Stato del Sistema . . . . .	43
9.5	Modello delle Specifiche . . . . .	44
9.6	Modello Computazionale . . . . .	45
9.6.1	Gestione degli Eventi . . . . .	45
9.7	Verifica . . . . .	48
9.7.1	Parametri del modello . . . . .	48
9.7.2	Spazio degli Stati . . . . .	48

9.7.3	Transazione del processo . . . . .	48
9.7.4	Risoluzione della Catena di Markov . . . . .	49
9.7.5	Risultati Ottenuti . . . . .	49
9.8	Validazione . . . . .	50
9.8.1	Mitigation Center . . . . .	50
9.8.2	Machine Learning Analysis . . . . .	52
9.8.3	Web Server . . . . .	54
9.8.4	Primo Spike Server Allocato . . . . .	56
9.8.5	Numero di Spike Server allocati . . . . .	58
9.9	Progettazione degli Esperimenti . . . . .	59
9.9.1	Analisi del Transitorio . . . . .	59
9.9.2	Simulazione ad Orizzonte Finito . . . . .	64
9.9.3	Simulazione ad Orizzonte Infinito . . . . .	72
9.9.4	Autocorrelazione . . . . .	77
<b>10</b>	<b>Conclusioni</b>	<b>78</b>

# 1 Introduzione

Questo documento ha l'obiettivo di descrivere lo studio e la valutazione delle prestazioni di un sistema di mitigazione per attacchi Distributed Denial of Service (DDoS).

Un attacco DDoS ha come obiettivo quello di sovraccaricare un servizio online, rendendolo inaccessibile agli utenti legittimi. Questo può generare gravi ripercussioni in termini economici, di reputazione e di continuità operativa. Mentre esistono meccanismi di filtraggio a livello di rete (es. firewall, rate limiting), gli attacchi DDoS a livello applicativo risultano particolarmente complessi da identificare, poiché spesso simulano il comportamento di utenti reali. Il filtraggio in questi casi diventa oneroso dal punto di vista computazionale e può introdurre falsi positivi, ostacolando anche gli utenti legittimi.

Nel contesto attuale, caratterizzato da un'intensificazione dell'uso di servizi digitali, i sistemi di mitigazione DDoS giocano un ruolo fondamentale nel garantire la qualità del servizio e la sicurezza delle infrastrutture aziendali. Uno degli aspetti più critici riguarda la gestione delle fluttuazioni di carico, che possono manifestarsi:

- nel lungo periodo, con intensità medio-bassa e frequenza ridotta, dovute ai trend fisiologici di crescita del traffico;
- nel breve periodo, improvvise e ad alta intensità, come nel caso di picchi di traffico o di attacchi DDoS.

Il problema del *right-sizing*, ovvero individuare il numero minimo di risorse necessario per rispettare gli obiettivi di prestazione, è quindi particolarmente complesso: un eccesso di risorse porta a costi inutili, mentre una carenza determina violazioni di QoS. Per affrontare questa sfida, i moderni sistemi cloud adottano meccanismi di *autoscaling*, che adattano dinamicamente il numero di server disponibili in funzione del carico. Le tecniche di scaling orizzontale risultano efficaci per variazioni graduali, ma si rivelano poco adatte a picchi repentina, poiché l'attivazione di nuove risorse richiede tempo e può generare oscillazioni instabili nella configurazione.

Un caso di studio analogo è presentato nel libro di testo *Performance Engineering: Learning Through Applications Using JMT* (Sezione 6.2, *Autoscaling Load Fluctuations*) [10], dove viene introdotto un **autoscaler gerarchico** capace di gestire i picchi improvvisi di traffico.

In questa relazione il caso di studio del libro viene esteso a uno scenario ibrido, in cui:

- la logica di mitigazione è gestita localmente (on-premise);
- i server applicativi risiedono su infrastruttura cloud con modello pay-per-use.

Pertanto, l'obiettivo dello studio è quello di modellare e simulare il sistema, valutando l'impatto del meccanismo di autoscaling sia sulle prestazioni (latenza, throughput, utilizzo delle risorse), sia sui costi associati all'infrastruttura. Lo scopo è identificare una configurazione ottimale in grado di garantire un livello adeguato di servizio agli utenti, mantenendo al contempo sotto controllo i costi operativi.

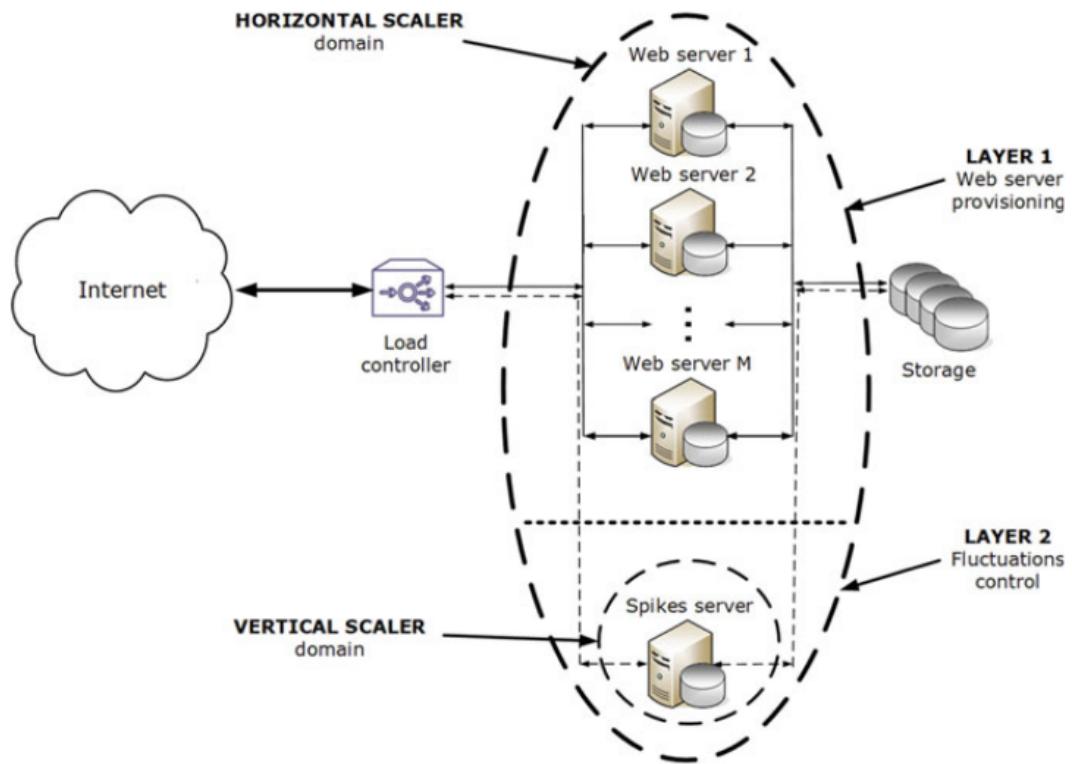


Figure 1: Autoscaler Gerarchico per la gestione dei picchi di traffico

## 2 Scopi e Obiettivi

Lo scopo di questo studio è modellare e analizzare un sistema di mitigazione per attacchi DDoS, includendo l'effetto dell'allocazione dinamica di server su infrastruttura cloud. In particolare, si intende valutare le prestazioni del sistema in termini di costi e qualità del servizio, con l'obiettivo di garantire un tempo medio di risposta inferiore a 5 secondi per le richieste lecite anche in presenza di un attacco.

Gli obiettivi specifici della simulazione sono:

- **Minimizzare il numero di server allocati dinamicamente** per la gestione delle richieste, al fine di contenere i costi legati all'autoscaling;
- **Rispettare i requisiti di QoS** in termini di tempo medio di risposta delle richieste considerate lecite.

Il rispetto di tali obiettivi consente di ottenere molteplici benefici: una riduzione dei costi operativi legati al cloud computing, un miglioramento dell'affidabilità percepita del servizio e un rafforzamento della credibilità dell'azienda nei confronti degli utenti finali anche in caso di attacchi DDoS.

### 3 Modello Concettuale

In questa fase dell’analisi, le caratteristiche del sistema vengono definite a un livello astratto. L’obiettivo principale è esaminare gli stati e i possibili eventi all’interno del sistema, valutando come questi siano correlati.

Il sistema considerato è composto da tre tipologie principali di centri di servizio, ciascuno con un ruolo specifico nella gestione del traffico e nella mitigazione dell’attacco.

- **Mitigation Server**

Questo centro rappresenta il componente responsabile dell’analisi e del filtraggio del traffico in ingresso. Applica euristiche per determinare se una richiesta HTTP sia lecita o faccia parte di un attacco DDoS. Le richieste identificate come malevole vengono scartate, impedendo che raggiungano i server applicativi e preservando così le risorse per gli utenti legittimi.

In alcuni casi, l’analisi potrebbe risultare inconcludente, ad esempio per mancanza di informazioni sufficienti o per comportamenti ambigui del client. In tali situazioni, il sistema implementa un meccanismo di *retry*, che consente di sottoporre una stessa richiesta a più tentativi di classificazione.

Bisogna sottolineare che tali sistemi potrebbero classificare in modo erroneo delle richieste: una richiesta che fa parte di un attacco DDoS potrebbe essere classificata come lecita e viceversa.

In base a come il centro applica questa classificazione si possono avere delle complicazioni. In particolare, se una richiesta lecita dovesse essere classificata come illecita allora questa verrà scartata, con conseguente ritardo e disservizio percepito dell’utente finale del sistema, il che si può classificare come una perdita di credibilità per l’azienda che fornisce il servizio stesso.

Al contrario, se una richiesta illecita viene classificata come lecita allora deve essere processata, questo determina un’utilizzazione delle risorse a disposizione, a discapito delle effettive richieste lecite degli utenti finali, ma anche maggiori costi di gestione per le istanze allocate nel cloud.

- **Web Server**

È il centro incaricato dell’effettiva elaborazione delle richieste considerate lecite. In condizioni normali, viene utilizzato un solo server per processare il traffico in ingresso. Tuttavia, quando il numero di richieste supera una soglia prestabilita, viene attivato un meccanismo di autoscaling che permette l’allocazione di server aggiuntivi per mantenere la qualità del servizio.

- **Spike Server**

Questi centri rappresentano le istanze aggiuntive allocate dinamicamente in risposta a un picco di traffico. Sono attivati dal meccanismo di autoscaling e hanno il compito di gestire il carico eccedente, supportando il Web Server nel garantire le prestazioni richieste dal sistema.

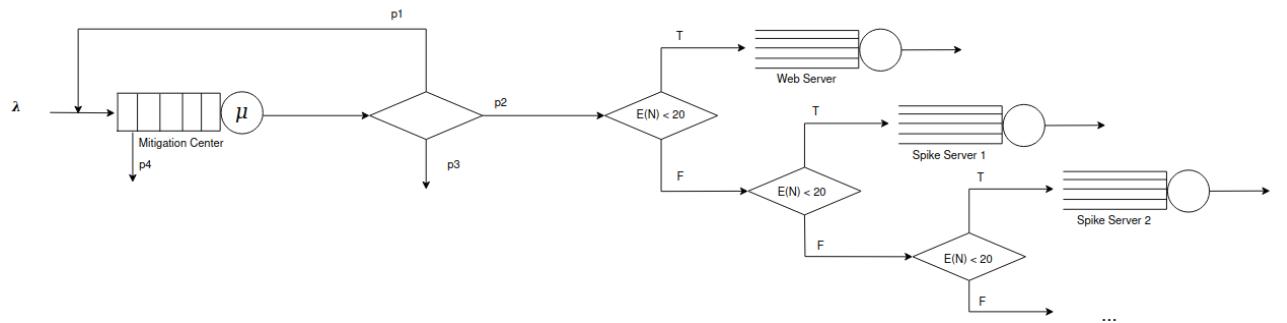
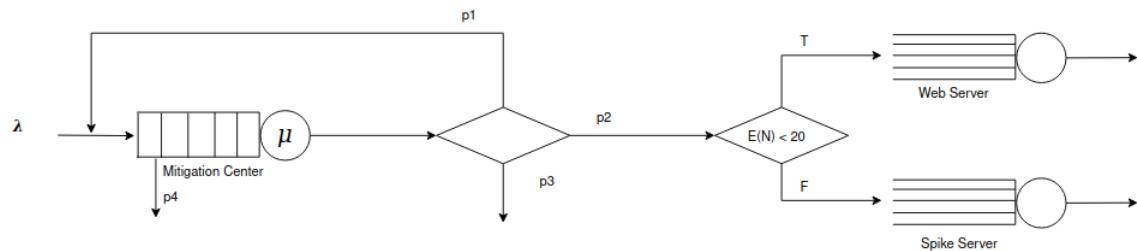
In particolare, il primo Spike Server viene allocato al superamento di una certa soglia da parte del Web Server, il secondo Spike Server viene allocato al superamento di una certa soglia del primo Spike Server. Il processo di allocazione è dinamico grazie al meccanismo di autoscaling utilizzato.

Per essere consistenti con l’esempio preso dal libro di testo, la soglia utilizzata per l’allocazione di nuovi Spike Server per la gestione dei picchi di traffico è basata sulla popolazione attuale dei Server che utilizzano la gestione dei servizi in modalità Processor Sharing.

In particolare, se il Web Server sta attualmente servendo 20 richieste e arriva una nuova da processare, allora viene allocato in maniera dinamica uno Spike Server in grado di gestire questa nuova richiesta. La stessa cosa accade considerando anche gli Spike Server: se il Web Server che gli Spike Server allocati stanno attualmente servendo 20 richieste e arriva una nuova richiesta al sistema allora verrà allocato un nuovo Spike Server in grado di soddisfare la richiesta.

Bisogna considerare che nel sistema considerato, in particola modo per la gestione corretta dell'autoscaling e dell'istanziazione delle nuove istanze di Spike Server da utilizzare per la gestione dei picchi di carico, non viene affrontata la problematica del *cold start* delle nuove istanze ma viene assunto che tale tempo di avvio sia trascurabile ai fini dello studio.

Il modello concettuale appena descritto può essere rappresentato come segue:



### 3.1 Eventi del Sistema

Nel contesto della simulazione, si utilizza il termine *job* come sinonimo di richiesta. Il job rappresenta l'entità da processare nel sistema e può essere sia lecito sia illecito. L'insieme dei centri che compongono l'infrastruttura costituisce il *sistema* nel suo complesso.

#### 3.1.1 Sistema

- L'arrivo di un job al Mitigation Server corrisponde a un evento di **arrivo nel sistema**.
- L'evento di completamento di un job può avvenire solamente all'uscita dal Web Server o dagli Spike Server allocati.

#### 3.1.2 Mitigation Server

- L'arrivo di una richiesta al Mitigation Server è considerato un evento di arrivo nel sistema.

- È considerato evento di arrivo anche il caso in cui una richiesta venga reinviata al Mitigation Server per una nuova analisi (retry), in seguito a una classificazione inconcludente. Questo evento, però, **non coincide** con un nuovo arrivo nel sistema.
- Il completamento dell’analisi da parte del Mitigation Server può portare a due possibili esiti:
  - la richiesta è correttamente identificata come *illecita* e viene scartata. In questo caso, l’evento non coincide con un completamento del sistema;
  - la richiesta è classificata come *lecita* e viene instradata verso uno dei centri di elaborazione (Web Server o Spike Server).

### 3.1.3 Web Server

- Un evento di arrivo al Web Server si verifica quando una richiesta classificata come lecita dal Mitigation Server viene assegnata a questo centro per l’effettiva esecuzione del servizio richiesto. Tale evento **non rappresenta un nuovo arrivo nel sistema**.
- Il completamento della richiesta da parte del Web Server rappresenta un evento di **completamento del sistema**.

### 3.1.4 Spike Server

- Un evento di arrivo ad uno Spike Server si verifica quando una richiesta lecita non può essere gestita dal Web Server a causa del superamento di una soglia di carico. In questo caso, il sistema attiva dinamicamente uno Spike Server per assorbire il carico eccedente. Anche questo evento **non rappresenta un arrivo nel sistema**.
- Il completamento dell’elaborazione da parte dello Spike Server rappresenta un evento di **completamento del sistema**.

## 3.2 Stati del Sistema

Lo stato del sistema è rappresentato dall’unione degli stati dei singoli centri.

Per quanto riguarda il centro di mitigazione, ovvero il centro che gestisce le code con una politica FIFO, lo stato è definito dello stato delle coda e dallo stato del servente: lo stato di una coda è rappresentato dalla presenza di richieste accodate, lo stato del servente è rappresentato dall’essere occupato o meno.

Per quanto riguarda i centri che gestiscono le richieste con modalità processor sharing, ovvero il Web Server e i vari Spike Server allocati dinamicamente, lo stato è rappresentato dal numero di richieste attualmente servite.

Lo stato del sistema è descritto da una tupla formata nel seguente modo  $(i, w, s_1, \dots, s_n)$  dove:

- $i \in \{0, 1, 2, 3, 4, \dots, 1025\}$ : numero di job nel centro di mitigazione. Il numero job in questo centro è costituito dalla somma delle richieste poste in coda e dalla richiesta che viene processata.
- $w \in \{0, 1, \dots, 20\}$ : numero di job nel Web server.
- $s_i \in \{0, 1, \dots, 20\}$ : numero di job nel Spike server. A livello concettuale non è possibile definire a priori il numero esatto di Spike Server da allocare a causa delle fluttuazioni del carico da processare.

## 4 Modello delle Specifiche

In questa sezione vengono presentati i parametri principali utilizzati per la modellazione del sistema. Le scelte effettuate derivano da due fonti principali: il caso di studio del libro di testo [10] e studi in letteratura relativi agli attacchi DDoS. Il paper "*A Simulation Model for the Analysis of DDoS Amplification Attacks*" [4], mostra come il traffico in ingresso durante un attacco DDoS possa raggiungere picchi di un fattore moltiplicativo pari a  $\times 40$ .

«*Open Resolver DNS, which are recursive DNS resolvers that accept requests from anyone, are frequently used as amplifiers to perform DDoS amplification attacks. These servers allows recursive ANY queries that make it possible to generate a large response by sending a single packet of 64 bytes. The BAF for this amplification attack is 40x*»

Per il sistema in esame, si è quindi assunto un tasso di ingresso pari a quello specificato dal libro di testo con un fattore moltiplicativo di 40.

Questa assunzione non rappresenta una media di tutti gli attacchi DDoS, ma è un esempio specifico di un possibile scenario d'attacco particolarmente aggressivo. Il valore è stato scelto per testare la robustezza del sistema in condizioni di stress elevato, considerando che il volume di traffico in questi attacchi varia notevolmente a seconda della tipologia e della tecnica impiegata.

Per quanto riguarda la scelta del valore della capienza della coda del centro di mitigazione si fa riferimento ai valori reali trovati in contesti enterprise di firewalling. In dispositivi come il *Cisco ASA*, la coda ha una dimensione predefinita di 1024 pacchetti, ma può essere configurata fino ai limiti di memoria disponibili [11].

Il tempo medio di servizio adottato per il Mitigation Center è pari a 1100 microsecondi, valore derivato dalla documentazione ufficiale Cisco [12] relativa alla performance del modulo IPS dei firewall Secure. In particolare, Cisco dichiara che la maggior parte delle operazioni di ispezione profonda dei pacchetti (Deep Packet Inspection) vengono completate in meno di 1100  $\mu$ s. Questo tempo di servizio rappresenta quindi una stima realistica e conservativa per simulare la capacità del sistema di mitigazione di elaborare pacchetti in modo rapido ma non istantaneo.

Al termine dell'analisi svolta dal Mitigation Center, ogni richiesta può avere tre possibili esiti:

- può essere **reinviata** al centro di mitigazione per una seconda verifica tramite un meccanismo di *feedback*. Nel modello questa probabilità è fissata a  $P_{feedback} = p_1 = 2\%$ , in linea con scenari reali in cui una piccola quota di traffico richiede ulteriori controlli a causa di incertezze nella classificazione [1] [2];
- può essere scartata poiché identificata come malevola. I sistemi di sicurezza come gli IDS/IPS presentano tipicamente un tasso di scarto del traffico, inclusi i falsi positivi, compreso tra l'1% e il 3%. Per la nostra modellazione, abbiamo scelto il valore  $p_3 = 1\%$ . Questo valore, apparentemente basso, riflette il comportamento di un firewall generico che non è progettato per analizzare il volume di traffico, ma si basa principalmente su regole predefinite (come blocklist per indirizzi IP, firme note, euristiche base). [8];
- infine, se la richiesta non viene reinviata né scartata, essa viene accettata e inoltrata verso i server applicativi (Web o Spike). Questa ha probabilità  $p_2 = 97\%$ .

Va infine considerato che, indipendentemente da questi esiti, una richiesta può anche essere scartata immediatamente all'arrivo qualora la coda del Mitigation Center sia piena.

### 4.1 Stime dei costi

Per stimare i costi del sistema, in particolare quelli legati alle istanze di Web Server e Spike Server allocate dinamicamente, abbiamo ipotizzato di modellare ciascun server come una macchina

virtuale attivata tramite il servizio Amazon EC2 [3], sfruttando il meccanismo di Auto Scaling offerto da AWS.

Le tariffe orarie delle diverse tipologie di istanze sono state raccolte dalla documentazione ufficiale AWS e riportate nella tabella precedente. Nello studio si è considerato un intervallo di configurazioni con almeno 4 e fino a 32 vCPU, così da poter gestire in modo realistico i volumi di traffico attesi. Tutte le configurazioni riportate si riferiscono a macchine general purpose con sistema operativo Linux.

Per la valutazione economica è stata scelta, a titolo esemplificativo, la macchina **r6a.xlarge**, utilizzata sia per il Web Server che per gli Spike Server attivati dinamicamente. Questa scelta ha carattere puramente indicativo ed è motivata dall'obiettivo di stimare i costi di utilizzo. Inoltre, il servizio di Auto Scaling non comporta costi aggiuntivi [9], quindi l'utente paga esclusivamente le risorse effettivamente allocate per gestire le richieste.

È importante sottolineare che l'analisi è stata condotta su una delle istanze più economiche tra quelle considerate; di conseguenza, l'impatto economico reale potrebbe risultare significativamente superiore qualora si optasse per macchine più potenti.

Nome istanza	Tariffa oraria (USD)	vCPU	Memoria	Storage	Prestazioni di rete	Costo orario €	Costo mensile €
x8g.xlarge	0,3908	4	64 GiB	Solo EBS	Fino a 12,5 Gigabit	0,34	244,8
r6a.xlarge	0,2268	4	32 GiB	Solo EBS	Fino a 12500 Megabit	0,19	136,8
is4gen.xlarge	0,5763	4	24 GiB	1 x 3750 SSD	Fino a 25 Gigabit	0,50	360
i7i.2xlarge	0,755	8	64 GiB	1 x 1875 NVMe SSD	Fino a 12 Gigabit	0,65	468
r6i.2xlarge	0,504	8	64 GiB	Solo EBS	Fino a 12500 Megabit	0,44	316,8
m6idn.2xlarge	0,63648	8	32 GiB	1 x SSD NVMe 474	Fino a 40.000 Megabit	0,55	395
i3en.3xlarge	1,356	12	96 GiB	1 x 7.500 SSD NVMe	Fino a 25 Gigabit	1,17	824,4
z1d.3xlarge	1,116	12	96 GiB	1 x 450 SSD NVMe	Fino a 25 Gigabit	0,97	698,4
i7ie.3xlarge	1,5594	12	96 GiB	1 x 7.500 SSD NVMe	Fino a 25 Gigabit	1,35	972
r5n.4xlarge	1,192	16	128 GiB	Solo EBS	Fino a 25 Gigabit	1,03	741,6
r5n.4xlarge	1,5632	16	256 GiB	Solo EBS	Fino a 15 Gigabit	1,35	972
x2iedn.4xlarge	3,3345	16	512 GiB	1 x 475 SSD NVMe	Fino a 25 Gigabit	2,88	2073,6
i7ie.6xlarge	3,1188	24	192 GiB	2 x 7.500 SSD NVMe	Fino a 25 Gigabit	2,70	1944
inf1.6xlarge	1,18	24	48 GiB	Solo EBS	25 Gigabit	1,02	734,4
hpc7a.12xlarge	7,20	24	768 GiB	Solo EBS	300 Gigabit	6,23	4485,6
g6e.8xlarge	4,52856	32	256 GiB	1 x 900 GB NVMe SSD	25 Gigabit	3,92	2822,4
x2iedn.8xlarge	6,669	32	1024 GiB	1 x 950 SSD NVMe	25 Gigabit	5,77	4154,4
p3.8xlarge	12,24	32	244 GiB	Solo EBS	10 Gigabit	10,59	7624,8

## 4.2 Modellazione dei Centri

### Mitigation Center

È stato modellato come un singolo server con coda FIFO finita. La scelta della distribuzione esponenziale è motivata dalla sua semplicità e dalla capacità di rappresentare un'elevata variabilità nei tempi di processamento, in assenza di dati empirici.

### Web Server e Spike Servers

Sono stati modellati come centri PS (Processor Sharing), in cui la capacità del server viene suddivisa equamente tra tutti i job attualmente in servizio. Questo modello è particolarmente indicato per rappresentare sistemi in cui più richieste vengono elaborate “in parallelo” a livello logico, come avviene nei processori time-shared o nei server web multitasking. Nel PS, ogni job riceve una frazione del tempo di servizio proporzionale alla capacità totale divisa per il numero di job presenti. Se il numero di job in servizio è  $n$  e il tempo trascorso dall'ultimo evento (arrivo o completamento) è  $dt$ , allora il tempo di servizio effettivo assegnato a ciascun job è:

$$\frac{dt}{n}$$

La formula riflette il fatto che tutti i job avanzano simultaneamente verso il completamento: più job sono presenti, più lentamente ciascuno procede. In altre parole, se il server ha capacità C normalizzata a 1, ogni job riceve  $1/n$  della capacità totale.

Ogni volta che si verifica un evento vengono eseguite le seguenti operazioni:

1. si calcola  $dt$ , ovvero il tempo trascorso dall'ultimo evento;
2. si riduce il remaining time di ciascun job di  $dt/n$  e si aggiorna il tempo di completamento;
3. se il remaining time di un job raggiunge zero, questo completa il servizio ed esce dal sistema;
4. il numero di job in servizio viene aggiornato e il processo continua fino al prossimo evento.

Dal punto di vista operativo, il server mantiene l'informazione sul tempo residuo di ogni job e lo aggiorna a ogni evento, riducendolo della quota corrispondente al servizio ricevuto. L'aggiunta o la rimozione di un job modifica immediatamente la velocità con cui tutti i job avanzano verso il completamento, simulando così l'effetto di un servizio condiviso in parallelo.

Parametro	Valore
Distribuzione degli arrivi	Iperesponenziale ( $\lambda_1 = 16.176$ , $\lambda_2 = 517.156$ , $p = 0.03033$ )
Capienza coda Mitigation	1024 job
Tempo medio servizio Mitigation	$E[S] = 1100 \mu s \Rightarrow \mu = 909 \text{ job/s}$
Prob. feedback	$p_1 = 0.02$
Prob. scarto	$p_3 = 0.01$
Prob. accettazione	$p_2 = 0.97$
Distribuzione servizio Web/Spike	Iperesponenziale ( $\lambda_1 = 0.3791$ , $\lambda_2 = 12.1208$ , $p = 0.03033$ )
Threshold N	20 job

Table 1: Parametri principali del modello delle specifiche

## 5 Modello Computazionale

Il modello computazionale del sistema è stato realizzato utilizzando la **Next-Evet Simulation**. Come linguaggio di programmazione è stato utilizzato Python per una serie di vantaggi: la gestione implicita della memoria, un elevato livello di flessibilità e l'orientamento della alla programmazione object oriented; in questo modo è stato possibile utilizzare molte librerie che facilitano la programmazione, sia per il codice relativo al simulatore che per la realizzazione dei grafici.

Per l'organizzazione del codice si è cercato di eseguire una divisione secondo il pattern di progettazione MVC, in cui si è cercato di dividere le responsabilità di model, view e controller cercando di avere il sorgente il più modulare possibile in modo che eventuali modifiche future non impattino in modo cruciali l'intero codice sorgente.

Il codice sviluppato è all'interno della directory `src` al cui interno sono presenti:

- file `main.py`: il programma principale da eseguire. Quando viene eseguito viene stampato a schermo diverse modalità di esecuzione della simulazione che verrà eseguita in base al valore passato dall'utente tramite input;
- directory `controller`: la cartella contiene il file `simulation.py` in cui è presente il codice utilizzato per i vari tipi di simulazione, codice definito all'interno della classe `DDoSSystem`, che rappresenta una delle porzioni del core per l'esecuzione dei vari tipi di simulazione;
- directory `engineering`: la cartella contiene dei file a supporto utilizzati per l'esecuzione delle varie simulazioni. Per esempio, nel file `constants.py` vengono definite le varie costanti che poi verranno utilizzate, come i paramenti utilizzati per le distribuzioni, per i seeds

utilizzati, le varie costanti rappresentanti le probabilità utilizzate e le varie condizioni per fermare la simulazione. Un altro file all'interno della directory è il file `distributions.py`, utilizzato per selezionare il tipo di distribuzione, e con quali parametri, utilizzare per la simulazione. Il file `statistics.py` contiene delle funzioni a supporto per l'esecuzione di alcuni tipi di simulazione, come per esempio per l'applicazione del metodo *Batch Means* per le simulazioni ad orizzonte infinito;

- directory `library`: all'interno sono presenti i file `rngs.py`, `rvgs.py` e `rvms.py`, ovvero i codici forniti durante il corso e che abbiamo utilizzato per semplicità; sono stati utilizzati per il generatore di numeri casuali, per il multi-stream e per il calcolo di funzioni di densità;
- directory `model`: all'interno della cartella sono altre porzioni del core delle simulazioni, ovvero i file `job.py`, `mitigation_center.py` e `processor_sharing_server.py` vengono modellati rispettivamente le richieste in arrivo al sistema, il centro di mitigazione delle richieste e un centro con la politica di gestione Processor Sharing utilizzato per il Web Server e gli Spike Server allocati. All'interno del file `mitigation_manager.py` sono presenti le ultime due parti fondamentali per il core delle simulazioni: il meccanismo di autoscaling e l'algoritmo di instradamento delle richieste in arrivo al sistema;
- directory `plot`: sono presenti i file `.csv` contenenti i risultati raccolte nelle varie esecuzioni delle simulazioni, i codici per la generazione dei grafici e le varie cartelle contenenti grafici generati;
- all'interno della directory `utils` vengono definiti altri file a supporto delle esecuzione delle simulazioni. All'interno della cartella è presente il file `resolve_markov_chain.py`, ovvero il codice utilizzato per risolvere la catena di Markov rappresentante il sistema, codice utilizzato nella parte di verifica.

## 5.1 Gestione degli Eventi

La simulazione adotta l'approccio del Next-Event Simulation, che consiste nel far avanzare il tempo processando gli eventi successivi in ordine cronologico. In altre parole, si individua l'evento più imminente, che sia un arrivo o un completamento, e si esegue l'azione associata ad esso, determinando così il cambiamento di stato del sistema e l'aggiornamento dei tempi.

A livello di codice, la gestione degli eventi avviene all'interno dei metodi `DDoSSystem.arrival_process()`, `MitigationManager.handle_job()` e `MitigationManager._mitigation_process()` in cui viene gestita la logica degli arrivi al sistema, in particolare al centro di mitigazione, e l'instradamento delle richieste verso i centri destinati all'effettiva esecuzione delle richieste in ingresso. All'interno della funzione dell'instradamento viene anche considerata la logica degli scarti e del meccanismo di feedback dopo una prima fase di analisi.

Per i server che gestiscono le richieste in modalità Processor Sharing, la gestione degli eventi avviene nei metodi `arrival()`, `schedule_completion()`, `completion_event()`, `schedule_next_completion()`, `update()`. Dove ad ogni aggiornamento vengono aggiornati i tempi di completamento e viene impostato il prossimo evento di completamento.

Per la gestione delle richieste da parte del centro di mitigazione, la gestione degli arrivi viene gestita all'interno dei metodi `update()` e `server_process()`.

Indipendentemente dal tipo di simulazione, che sia finita o infinita, la gestione degli arrivi avviene principalmente attraverso i metodi e le funzioni precedentemente elencate.

Di seguito sono mostrati i codici utilizzati.

Funzione `DDoSSystem.arrival_process()`.

```

1 def arrival_process(self, mode):
2     if mode == "verification":
3         arrivals = N_ARRIVALS_VERIFICATION

```

```

4         interarrival_mode = "verification"
5         stop_on_arrivals = True
6     elif mode in ("transitory", "finite simulation"):
7         interarrival_mode = "standard"
8         stop_on_arrivals = False
9     else: # "standard"
10        arrivals = N_ARRIVALS
11        interarrival_mode = "standard"
12        stop_on_arrivals = True
13
14    if not stop_on_arrivals:
15        while True:
16            interarrival_time = get_interarrival_time(interarrival_mode, self.
17                arrival_p, self.arrival_l1, self.arrival_l2)
18            yield self.env.timeout(interarrival_time)
19
20            self.metrics["total_arrivals"] += 1
21            arrival_time = self.env.now
22            is_legal = random() < P_LECITO
23            if is_legal:
24                self.metrics["legal_arrivals"] += 1
25            else:
26                self.metrics["illegal_arrivals"] += 1
27
28            job = Job(self.metrics["total_arrivals"], arrival_time, None, is_legal)
29            self.mitigation_manager.handle_job(job)
30        else:
31            while self.metrics["total_arrivals"] < arrivals:
32                interarrival_time = get_interarrival_time(interarrival_mode, self.
33                    arrival_p, self.arrival_l1, self.arrival_l2)
34                yield self.env.timeout(interarrival_time)
35
36                self.metrics["total_arrivals"] += 1
37                arrival_time = self.env.now
38                is_legal = random() < P_LECITO
39                if is_legal:
40                    self.metrics["legal_arrivals"] += 1
41                else:
42                    self.metrics["illegal_arrivals"] += 1
43
44            job = Job(self.metrics["total_arrivals"], arrival_time, None, is_legal)
45            self.mitigation_manager.handle_job(job)

```

Listing 1: Funzione arrival\_process

Funzione MitigationManager.handle\_job().

```

1 def handle_job(self, job):
2     if not self.center.has_capacity():
3         if "discarded_detail" not in self.metrics:
4             self.metrics["discarded_detail"] = []
5         self.metrics["discarded_detail"].append({
6             "time": self.env.now,
7             "job_id": job.id,
8             "is_legal": job.is_legal
9         })
10        self.metrics["discarded_mitigation"] = self.metrics.get("
11            discarded_mitigation", 0) + 1
12
13    return

```

```

12     if job.is_legal:
13         self.metrics["processed_legal"] = self.metrics.get("processed_legal", 0) +
14             1
15     else:
16         self.metrics["processed_illegal"] = self.metrics.get("processed_illegal",
17             0) + 1
18
19     self._mitigation_process(job)

```

Listing 2: Metodo `handle_job`

Funzione `MitigationManager._mitigation_process()`.

```

1 def _mitigation_process(self, job):
2     try:
3         self.center.arrival(job)
4     except Exception:
5         return
6     now = self.env.now
7
8     # Classificazione: false positive => job droppto
9     selectStream(RNG_STREAM_FALSE_POSITIVE)
10    if random() < P_FALSE_POSITIVE:
11        self.metrics["false_positives"] = self.metrics.get("false_positives", 0) +
12            1
13    if job.is_legal:
14        self.metrics["false_positives_legal"] = self.metrics.get("false_positives_legal", 0) + 1
15    return
16
17    # Feedback verso mitigation center
18    selectStream(RNG_STREAM_FEEDBACK)
19    if random() < P_FEEDBACK_VERIFICATION:
20        job.arrival = now
21        self.handle_job(job)
22    return
23    selectStream(RNG_STREAM_SERVICE_TIMES)
24    service_time = get_service_time(self.mode)
25    job.remaining = service_time
26    job.original_service = service_time
27    job.last_updated = now
28
29    job.arrival = now
30
31    # Routing verso Web e Spike
32    if len(self.web_server.jobs) < MAX_WEB_CAPACITY:
33        self.web_server.arrival(job)
34    return
35
36    for server in self.spike_servers:
37        if len(server.jobs) < MAX_SPIKE_CAPACITY:
38            server.arrival(job)
39    return
40    new_id = len(self.spike_servers)
41    new_server = ProcessorSharingServer(self.env, f"Spike-{new_id}")
42    self.spike_servers.append(new_server)
43
44    new_server.arrival(job)

```

Listing 3: Metodo `_mitigation_process`

Metodo ProcessorSharing.arrival().

```
1 def arrival(self, job):
2     now = self.env.now
3     self.update(now)
4
5     if len(self.jobs) == 0:
6         self.busy_periods.append([now, None])
7
8     self.jobs.append(job)
9     self.schedule_completion()
```

Listing 4: Metodo `arrival`

Metodo ProcessorSharing.schedule\_completion().

```
1 def schedule_completion(self):
2     if not self.jobs:
3         return
4     next_job = min(self.jobs, key=lambda j: j.remaining)
5     n = len(self.jobs)
6     delay = max(next_job.remaining * n, 0.0)
7
8     if self.proc and self.proc.is_alive and self.proc != self.env.active_process:
9         self.proc.interrupt()
10
11    if self.env.active_process != self.proc:
12        self.proc = self.env.process(self.completion_event(next_job, delay))
```

Listing 5: Metodo `schedule_completion`

Metodo ProcessorSharing.completion\_event().

```
1 def completion_event(self, job, delay):
2     try:
3         yield self.env.timeout(delay)
4     except simpy.Interrupt:
5         return
6
7     now = self.env.now
8     self.update(now)
9
10    if job in self.jobs:
11        self.jobs.remove(job)
12        response_time = now - job.arrival
13        global_rt = now - getattr(job, "sys_arrival", job.arrival)
14
15        self.completed_jobs.append(response_time)
16        self.global_completed_jobs.append(global_rt)
17        self.completion_times.append(now)
18        self.total_completions += 1
19
20        if hasattr(self, 'observer') and self.observer:
21            self.observer.notify_completion(self.name, response_time)
22
23        if job.is_legal:
24            self.legal_completions += 1
25        else:
26            self.illegal_completions += 1
27
28    if len(self.jobs) == 0:
```

```

29     if self.busy_periods and self.busy_periods[-1][1] is None:
30         self.busy_periods[-1][1] = now
31
32     if self.jobs:
33         self.env.process(self.schedule_next_completion())

```

Listing 6: Metodo `completion_event`

Metodo `ProcessorSharing.schedule_next_completion()`.

```

1 def schedule_next_completion(self):
2     yield self.env.timeout(0)
3     self.schedule_completion()

```

Listing 7: Metodo `schedule_next_completion`

Metodo `ProcessorSharing.update()`.

```

1 def update(self, now):
2     dt = now - self.last_time
3     if dt <= 0:
4         return
5
6     n = len(self.jobs)
7     self.area += n * dt
8
9     if n > 0:
10        self.busy_time += dt
11
12    self.last_time = now
13    self._record_busy_point(now)
14
15    if n > 0:
16        service_per_job = dt / n
17        for job in self.jobs:
18            job.remaining = max(job.remaining - service_per_job, 0.0)
19            job.last_updated = now

```

Listing 8: Metodo `update`

Metodo `MitigationCenter.server_process()`.

```

1 def server_process(self):
2     while True:
3         job = yield self.queue.get()
4
5
6         self.busy = True
7         self.last_start = self.env.now
8
9         selectStream(RNG_STREAM_MITIGATION_SERVICE)
10        yield self.env.timeout(Exponential(MITIGATION_MEAN))
11
12
13        now = self.env.now
14        self.busy_time += now - self.last_start
15        self.busy_periods.append((self.last_start, now))
16        self.completed_jobs.append(now - job.arrival)
17        self.completion_times.append(now)
18
19        self.total_completions += 1

```

```

20     if job.is_legal:
21         self.legal_completions += 1
22     else:
23         self.illegal_completions += 1
24
25     if self.metrics is not None:
26         self.metrics["mitigation_completions"] = self.metrics.get("mitigation_completions", 0) + 1
27
28     self.busy = False
29     self.last_start = None

```

Listing 9: Metodo `server_process`

Metodo `MitigationCenter.update()`.

```

1 def update(self, now):
2     if self.busy and self.last_start is not None:
3         self.busy_time += now - self.last_start
4         self.busy_periods.append((self.last_start, now))
5         self.last_start = now

```

Listing 10: Metodo `update`

## 6 Verifica

Per eseguire la verifica dei risultati l'approccio usato è stato modellare il sistema come una catena di Markov. Di seguito vengono mostrate le assunzioni utilizzate:

- Si simula il comportamento del sistema in condizioni di traffico normali, in assenza di attacchi DDoS.
- La capacità della coda del Mitigation Center è stata ridotta a 4.
- Per semplicità, le distribuzioni dei tempi di servizio sono di tipo esponenziale anziché iperesponenziale; questo proprio per rendere semplificato il calcolo di tutte le stime.
- La probabilità di feedback è stata settata a 0.
- L'autoscaling è stato disattivato, impostando il numero massimo di Spike pari a 1. Questo permette di mantenere il modello con un numero limitato di stati.

### 6.1 Parametri del modello

- $\lambda$ : tasso di arrivo medio dei job.

$$\begin{aligned}\mathbb{E}[r] &= p \cdot \frac{1}{\lambda_1} + (1 - p) \cdot \frac{1}{\lambda_2} \\ &= 0.03033 \cdot \frac{1}{0.4044} + 0.96967 \cdot \frac{1}{12.9289} \\ &= 0.03033 \cdot 2.473 + 0.96967 \cdot 0.07735 \\ &\approx 0.075 + 0.075 \approx 0.15 \text{ s} \\ \lambda &= \frac{1}{\mathbb{E}[r]} = 6.666666 \text{ job/s}\end{aligned}$$

- $\mu_{mit}$ : tasso di servizio del centro di mitigazione pari a 909 job/s.
- $p_{discard}$ : probabilità che un pacchetto venga classificato come illecito pari all'1 % .
- $p_{forward}$ : probabilità che un pacchetto venga classificato come lecito pari a 99 %.
- $\mu_w = \mu_s$  : tasso medio di servizio del Web server e Spike Server.

$$\begin{aligned}\mathbb{E}[S] &= p \cdot \frac{1}{\mu_1} + (1 - p) \cdot \frac{1}{\mu_2} \\ &= 0.03033 \cdot \frac{1}{0.3791} + 0.96967 \cdot \frac{1}{12.1208} \\ &= 0.03033 \cdot 2.638 + 0.96967 \cdot 0.0825 \\ &\approx 0.08 + 0.08 \approx 0.16 \text{ s} \\ \mu &= \frac{1}{\mathbb{E}[S]} = 6.25 \text{ job/s}\end{aligned}$$

### 6.2 Spazio degli stati

Lo stato del sistema è descritto dalla terna  $(i, w, s)$  dove:

- $i \in \{0, 1, 2, 3, 4\}$ : numero di job nel centro di mitigazione.
- $w \in \{0, 1, \dots, 20\}$ : numero di job nel Web server.

- $s \in \{0, 1, \dots, 20\}$ : numero di job nel Spike server.

Lo spazio degli stati  $\mathcal{S}$  è quindi dato da:

$$\mathcal{S} = \{(i, w, s) \mid 0 \leq i \leq 4, 0 \leq w \leq 20, 0 \leq s \leq 20\}$$

Il numero totale di stati è  $5 \times 21 \times 21 = 2205$ .

### 6.3 Transizioni del processo

Il processo è a tempo continuo e presenta le seguenti transizioni:

- **Arrivo dall'esterno**: se  $i < 4$ , transizione  $(i, w, s) \rightarrow (i + 1, w, s)$  con tasso  $\lambda$ .
- **Completamento nel centro di mitigazione** (solo se  $i > 0$ ):
  - Con probabilità  $p_{\text{discard}}$ : pacchetto scartato,  $(i, w, s) \rightarrow (i - 1, w, s)$  con tasso  $\mu \cdot p_{\text{discard}}$ .
  - Con probabilità  $p_{\text{forward}}$ :
    - \* Se  $w < 20$ :  $(i, w, s) \rightarrow (i - 1, w + 1, s)$  (Web accetta).
    - \* Se  $w = 20$  e  $s < 20$ :  $(i, w, s) \rightarrow (i - 1, w, s + 1)$  (invia a Spike).
    - \* Altrimenti viene scartato.
- **Completamento Web server** (se  $w > 0$ ): transizione  $(i, w, s) \rightarrow (i, w - 1, s)$  con tasso  $\mu_w$ .
- **Completamento Spike server** (se  $s > 0$ ): transizione  $(i, w, s) \rightarrow (i, w, s - 1)$  con tasso  $\mu_s$ .

### 6.4 Risoluzione della Catena di Markov

Generalizziamo il nostro esempio considerando la tupla  $(i, w, s)$ , con  $0 \leq i \leq K_m$ ,  $0 \leq w \leq K_w$ ,  $0 \leq s \leq K_s$ , le equazioni di bilanciamento hanno la forma:

$$\begin{aligned} \pi_{i,w,s} \cdot [\mathbf{1}_{\{i < K_m\}} \lambda + \mathbf{1}_{\{i > 0\}} \mu_m + \mathbf{1}_{\{w > 0\}} \mu_w + \mathbf{1}_{\{s > 0\}} \mu_s] = \\ \mathbf{1}_{\{i > 0\}} \cdot \pi_{i-1,w,s} \cdot \lambda \\ + \mathbf{1}_{\{i < K_m\}} \cdot \pi_{i+1,w,s} \cdot P_{\text{discard}} \mu_m \\ + \mathbf{1}_{\{i < K_m, w < K_w\}} \cdot \pi_{i+1,w-1,s} \cdot P_{\text{forward}} \mu_m \\ + \mathbf{1}_{\{i < K_m, w \geq K_w, s < K_s\}} \cdot \pi_{i+1,w,s-1} \cdot P_{\text{forward}} \mu_m \\ + \mathbf{1}_{\{w < K_w\}} \cdot \pi_{i,w+1,s} \cdot \mu_w \\ + \mathbf{1}_{\{s < K_s\}} \cdot \pi_{i,w,s+1} \cdot \mu_s \end{aligned}$$

La condizione di normalizzazione è:

$$\sum_{i=0}^{K_m} \sum_{w=0}^{K_w} \sum_{s=0}^{K_s} \pi_{i,w,s} = 1$$

Per risolvere il sistema è stato sviluppato uno script Python che implementa in forma matriciale le equazioni di bilanciamento dei flussi, sostituendo le funzioni indicatrici con controlli condizionali nel calcolo dei tassi di transizione. L'obiettivo è calcolare il sistema stazionario

$$\pi Q = 0$$

soggetto alla condizione di normalizzazione sopra scritta. In questo modo si ottengono le metriche a regime come utilizzazioni, popolazioni medie, throughput, tempi di risposta e probabilità di routing.

## 6.5 Risultati Ottenuti

Nella prossima figura vengono mostrati i risultati ottenuti tramite il risolutore della catena di Markov.

```
=====
CONFRONTO RISULTATI: MODELLO ANALITICO
=====

UTILIZZAZIONI:
Mitigation - Analitico: 0.007334
Web Server - Analitico: 0.973864
Spike Server - Analitico: 0.082242

THROUGHPUT (job/s):
Mitigation - Analitico: 6.666723
Web Server - Analitico: 6.086653
Spike Server - Analitico: 0.514013

TEMPI DI RISPOSTA (s):
Mitigation - Analitico: 0.001108
Web Server - Analitico: 1.964790
Spike Server - Analitico: 0.339016
```

Figure 2: Risultati ottenuti con il risolutore della Catena di Markov

Di seguito sono mostrati i risultati ottenuti con la verifica del simulatore a parità di distribuzioni utilizzate. La simulazione è stata eseguita ad orizzonte infinito utilizzando il metodo del Batch Means, dove il seed utilizzato è 123456789.

```
-- Web Server --
Response Time: : 1.911714 ± 0.073154 (95% CI)
Utilization: : 0.970170 ± 0.004679 (95% CI)
Throughput: : 6.078646 ± 0.037017 (95% CI)

-- Spike Server --
Spike-0:
Response Time: : 0.330175 ± 0.023988 (95% CI)
Utilization: : 0.076698 ± 0.007726 (95% CI)
Throughput: : 0.486146 ± 0.045012 (95% CI)

-- Mitigation Center --
Response Time: : 0.001112 ± 0.000009 (95% CI)
Utilization: : 0.007328 ± 0.000070 (95% CI)
Throughput: : 6.637292 ± 0.054570 (95% CI)
```

Figure 3: Risultati ottenuti dalla simulazione

Dai risultati ottenuti dal simulatore, si nota come i valori analitici siano tutti compresi all'interno degli intervalli ottenuti con il simulatore. Oltre queste metriche è stato possibile trovare le seguenti probabilità:

- Probabilità di trovare il Mitigation Center pieno circa pari a 0. Infatti:

$$p_{\text{loss}} = \frac{\left(\frac{\lambda}{\mu}\right)^C}{\sum_{i=0}^C \left(\frac{\lambda}{\mu}\right)^i} \approx 0.$$

- Probabilità che un job sia mandato al Web Server 0.912961.
- Probabilità che un job sia mandato allo Spike Server pari a 0.077031.
- Probabilità che un job venga rifiutato sia dal Web che Spike Server pari a 0.000008.

## 7 Validazione

Nella fase di validazione si vuole verificare che il modello implementato sia consistente rispetto al modello analizzato. Tuttavia, non avendo accesso gratuito a tracce di attacchi relativi a sistemi affini a quello modellato, è stato controllato che al variare del tempo di interarrivo delle richieste si ottengano delle variazioni delle statistiche del sistema concorde a quella attesa per un sistema reale.

In particolare, diminuendo il tempo di interarrivo tra le richieste e di conseguenza aumentando i valori dei  $\lambda_1$  e  $\lambda_2$ , si nota come le metriche di interesse tendano ad aumentare. Per la validazione abbiamo considerato le metriche relative al Mitigation Center, il Web Server e il primo Spike Server allocato, in particolare abbiamo considerato:

- **tempo medio di risposta;**
- **utilizzazione;**
- **throughput;**

Un'altra metrica che abbiamo considerato è il **numero di Spike Server allocati** durante l'attacco, questo perché ci si aspetta che con l'aumentare dell'intensità di traffico questo numero tenda ad aumentare.

Per l'esecuzione della validazione sono state eseguite delle simulazioni ad orizzonte infinito utilizzando il metodo Batch Means, dove il seed utilizzato è 123456789, mentre l'intervallo di confidenza è del 95%.

Per quanto riguarda i tassi di arrivo utilizzati siamo partiti dai parametri di arrivo definiti nell'esempio del libro di testo [10], per poi moltiplicare il valore di  $\lambda_1$  e  $\lambda_2$  di un fattore moltiplicativo a aumentandolo ogni volta fino ad arrivare al valore 40, ovvero il carico effettivo del sistema utilizzato per simulare il workload in caso di attacco, come indicato in [4].

In particolare, i fattori moltiplicativi utilizzati sono:  $\times 2, \times 5, \times 10, \times 40$ .

### 7.1 Mitigation Center

Nella prossima figura viene rappresentato il tempo di risposta medio del Mitigation Center al variare dei valori del tempo di interarrivo delle richieste.

Il comportamento ottenuto è un comportamento atteso: più richieste arrivano al sistema e più il tempo di risposta medio tenderà ad aumentare.

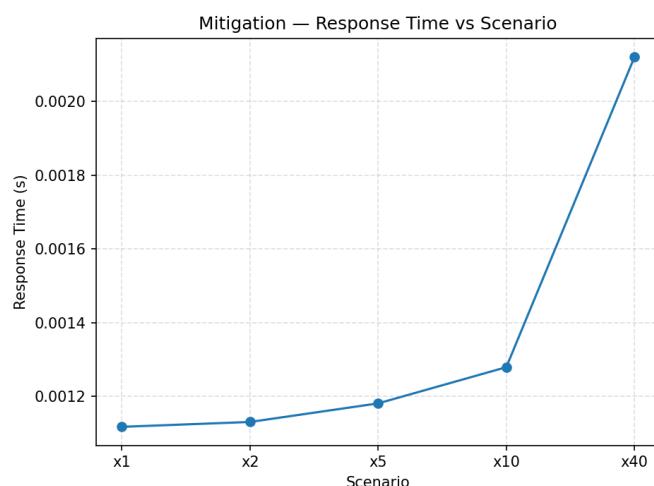


Figure 4: Tempo Medio di Risposta per il Mitigation Center al variare del tasso di arrivo

Nella prossima figura vengono mostrati i valori del throughput del centro di mitigazione ottenuti al variare del tempo di interarrivo. Anche in questo caso il comportamento ottenuto è consistente: maggiore è il numero di richieste che arrivano al sistema e maggiore sarà il throughput misurato.

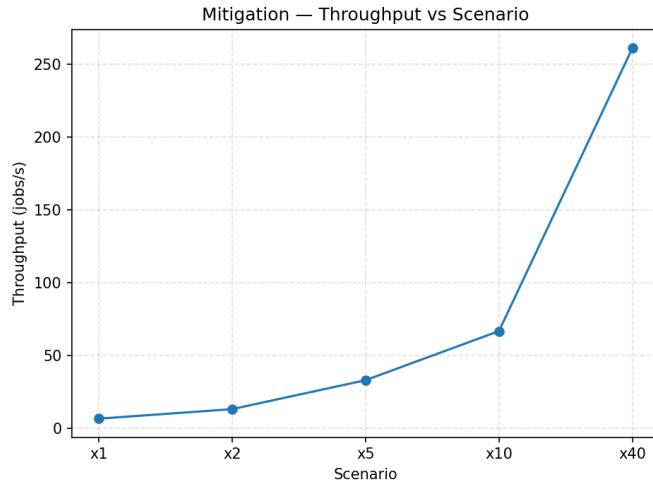


Figure 5: Throughput per il Mitigation Center al variare del tasso di arrivo

Nella prossima figura viene mostrato i valori ottenuti relativi all'utilizzazione del centro al variare del tempo di interarrivo. I risultati ottenuti sono coerenti con il comportamento atteso del centro: all'aumentare del numero delle richieste il valore dell'utilizzazione del centro tenderà ad aumentare.

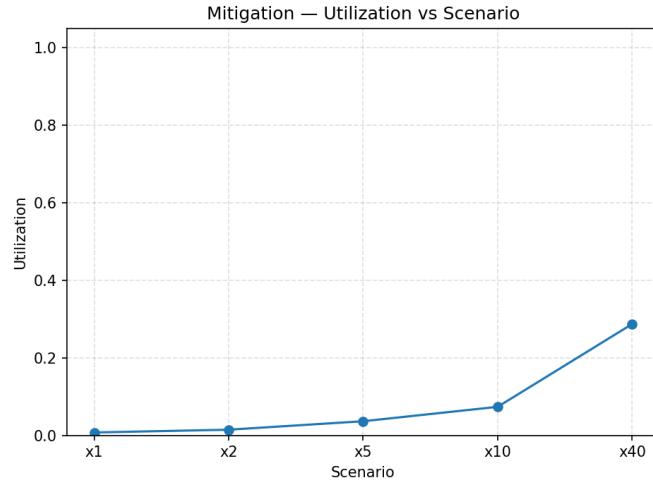


Figure 6: Utilizzazione per il Mitigation Center al variare del tasso di arrivo

I comportamenti osservati corrispondono ai comportamenti attesi e di conseguenza validano il sistema.

Di seguito vengono rappresentati i valori in forma tabellare dei risultati ottenuti.

Scenario	Tempo Medio di Risposta	Throughput	Utilizzazione
x1	0.0011175562330061487	6.665690104166666	0.007346427526366432
x2	0.0011307251115986884	13.195882161458332	0.014513671458381405
x5	0.0011811240220418228	33.06878306878306	0.03636782778156748
x10	0.0012792721904985717	66.77029569892473	0.07355543112966628
x40	0.002120375165444526	261.0086111111112	0.2873597416911712

Table 2: Tabella Riassuntiva delle metriche ottenute per il Mitigation Center

## 7.2 Web Server

Considerando la simulazione e l'utilizzo dei tassi di arrivo, sono stati ottenuti i seguenti risultati relativi al Web Server.

Come si può osservare dalla prossima figura, con l'aumentare del tasso di arrivo il tempo di risposta medio aumenta di conseguenza.

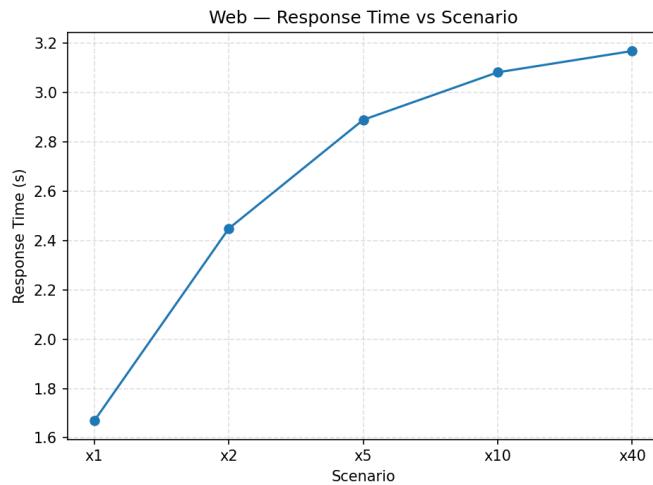


Figure 7: Tempo Medio di Risposta del Web Server al variare del tasso di arrivo

Questo è un risultato che conferma l'andamento atteso e valida il comportamento del web server: con l'aumentare del tempo di interarrivo il numero di richieste da processare aumenta di conseguenza e quindi le risorse del web server continueranno ad essere divise in base alle richieste all'interno del centro, con il conseguente aumento di tempo medio di risposta.

Nella prossima figura è mostrato il valore del throughput del centro al variare del tasso di arrivo utilizzato. Anche questo è un risultato atteso: con l'aumentare del tasso di arrivo verranno sottoposte più richieste al sistema e di conseguenza una percentuale di queste verrà sottoposta al web server con il conseguente aumento throughput dello stesso.

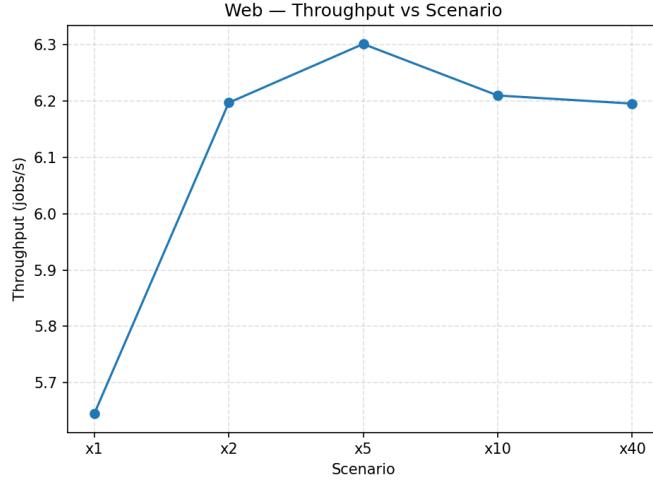


Figure 8: Throughput del Web Server al variare del tasso di arrivo

Un altro aspetto che si nota dal grafico è come il valore del throughput tendi leggermente a diminuire con l'aumentare degli arrivi al sistema. Questo è fortemente influenzato dal meccanismo di scaling: più richieste da processare ci sono, maggiore è la probabilità che vengano instradate verso gli altri Spike Server allocati.

Nella seguente figura viene mostrato il valore dell'utilizzazione del Web Server in base al valore del tasso di interarrivo utilizzato.

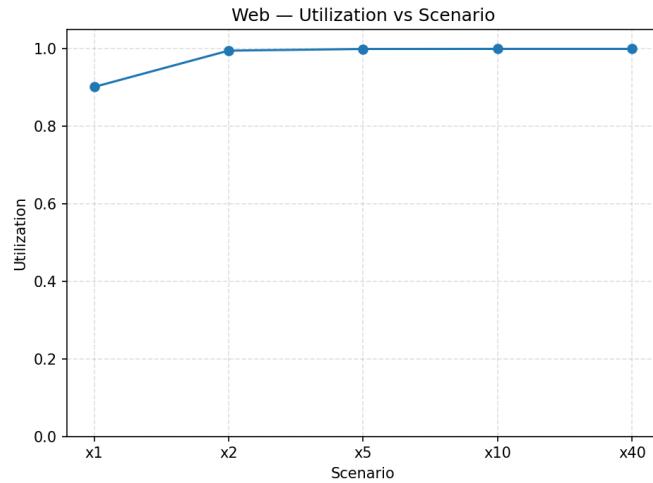


Figure 9: Utilizzazione del Web Server al variare del tasso di arrivo

Si osserva che l'utilizzazione cresce rapidamente fino a saturare il centro e rimanere prossima a 1. Questo risultato, che a prima vista potrebbe sembrare “anomalo”, è in realtà coerente con il modello adottato: il Web Server è progettato per essere sfruttato al massimo, mentre il meccanismo di autoscaling attiva nuovi Spike Server solo quando il Web Server ha raggiunto la sua soglia di saturazione impostata, ovvero in base alle richieste processate parallelamente.

Scenario	Tempo Medio di Risposta	Throughput	Utilizzazione
x1	1.6687503287841263	5.644986979166667	0.9017629398539585
x2	2.448566261567958	6.196614583333333	0.995495539809006
x5	2.889800312750591	6.300694444444445	0.999769933891889
x10	3.0825414859141826	6.209408602150537	0.9999927342510532
x40	3.1689752516466854	6.195000000000001	0.999999023996117

Table 3: Tabella Riassuntiva delle metriche ottenute per il Web Server

### 7.3 Spike Server

Di seguito sono mostrati i dati relativi al primo spike server allocato tramite il meccanismo di autoscaling al variare del tasso di interarrivo. Come si può notare valgono le stesse considerazioni fatte per il Web Server.

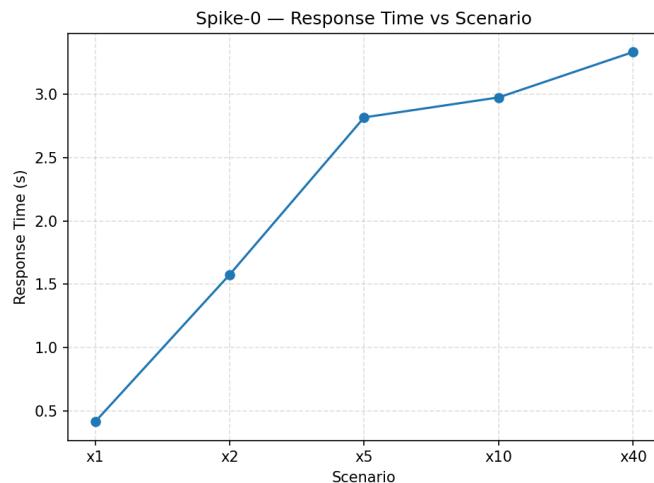


Figure 10: Tempo Medio di Risposta del primo Spike Server al variare del tasso di arrivo

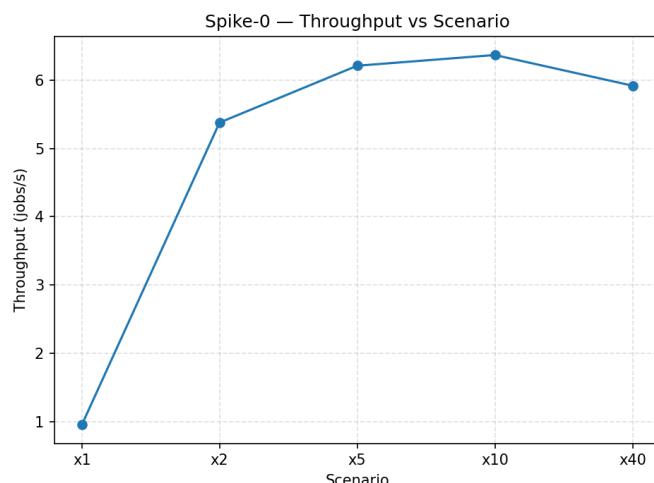


Figure 11: Throughput del primo Spike Server al variare del tasso di arrivo

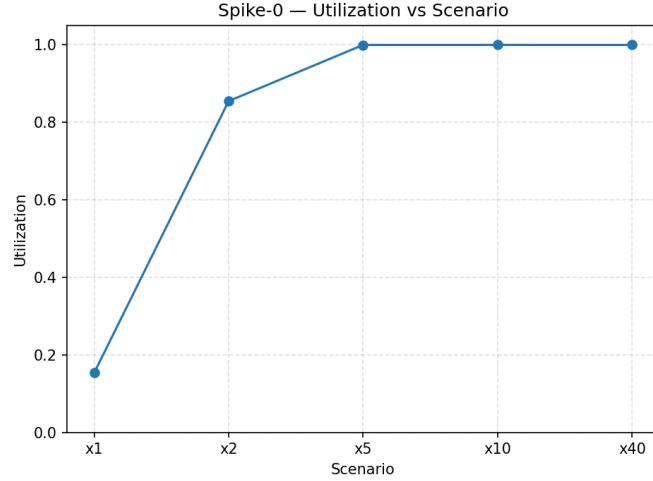


Figure 12: Utilizzazione del primo Spike Server al variare del tasso di arrivo

Scenario	Tempo Medio di Risposta	Throughput	Utilizzazione
x1	0.4161305717788653	0.95390625	0.15314895024205127
x2	1.5765842256113836	5.378645833333334	0.8549141060713916
x5	2.8157115443572165	6.21117724867725	0.9999072775267968
x10	2.9733181214632136	6.368279569892473	0.9999714265512207
x40	3.3312092275629945	5.918888888888889	0.9999931258327968

Table 4: Tabella Riassuntiva delle metriche ottenute per il primo Spike Server

#### 7.4 Numero di Spike Server allocati

Di seguito è riportato il grafico raffigurante il numero di Spike Server allocati al variare del tempo di interarrivo.

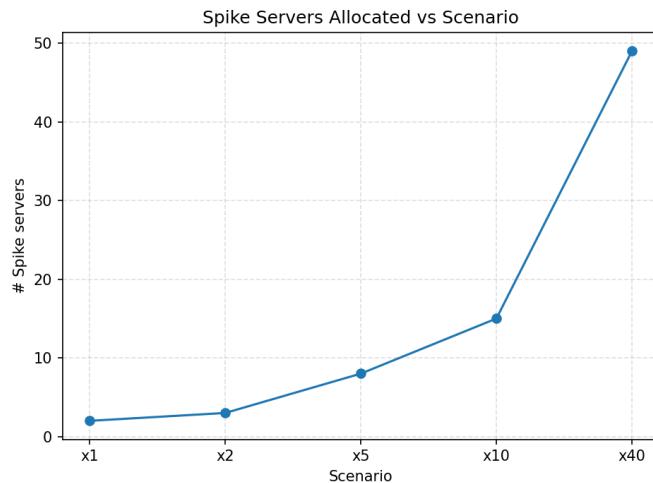


Figure 13: Numero di Spike Server al variare del tasso di arrivo

Il comportamento osservato corrisponde a quello atteso dal sistema, in particolare all'aumentare

del numero di richieste da processare allora il numero di spike server allocati tenderà ad aumentare.

Si nota come l'incremento non è lineare ma quasi esponenziale: con piccoli aumenti di carico si attivano poche istanze, mentre per scenari di attacco il sistema arriva ad allocare quasi 50 server.

Scenario	Spike Server Allocati
<b>x1</b>	2
<b>x2</b>	3
<b>x5</b>	8
<b>x10</b>	15
<b>x40</b>	49

Table 5: Tabella Riassuntiva del numero di Spike Server allocati

## 8 Progettazione degli esperimenti

La fase di progettazione degli esperimenti consiste nei seguenti tre punti:

- Analisi del transitorio: studio della convergenza del sistema allo stato stazionario.
- Simulazione a orizzonte finito: si simula il sistema per N ore tramite il metodo delle **Replications**.
- Simulazione a orizzonte infinito: si simula il sistema per un periodo di tempo molto più lungo rispetto a quello reale, utilizzando il metodo del **Batch Means**.

Nel dettaglio, la configurazione del sistema prende in considerazione lo scenario DDoS con fattore di carico pari a  $40\times$ .

### 8.1 Intervallo di Confidenza

Il calcolo dell'intervallo di confidenza è stato fatto tramite la seguente formula:

$$\text{Intervallo di Confidenza} = t^* \cdot \left( \frac{s}{\sqrt{n-1}} \right)$$

dove  $s$  è la deviazione standard campionaria,  $n$  è la dimensione del campione,  $t^*$  è il valore critico della distribuzione  $t$  di Student. Quest'ultimo valore è stato calcolato tramite la libreria **rvms** come:

$$t^* = \text{idfStudent}(n - 1, 1 - \frac{\alpha}{2}),$$

dove per il parametro  $\alpha$  è stato scelto il valore 0.05, che corrisponde ad un intervallo di confidenza del 95%.

### 8.2 Analisi del Transitorio

Per effettuare questo studio abbiamo eseguito 7 run della simulazione ad orizzonte finito con seed differenti.

Per poter valutare se il sistema raggiungesse un certo valore di convergenza abbiamo impostato la variabile **STOP\_CONDITION\_TRANSITORY = 100'000**

s, ovvero circa quasi un giorno in cui il sistema viene sottoposto ad un attacco, dove la raccolta delle metriche avviene con un intervallo di 100 secondi.

Di seguito vengono mostrati i grafici ottenuti.

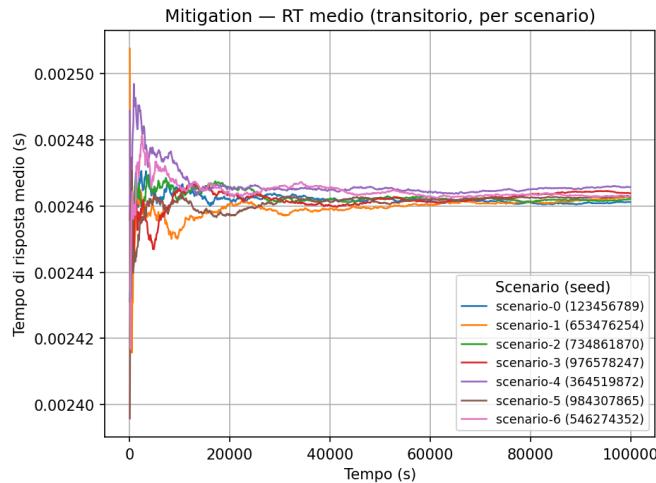


Figure 14: Tempo di Risposta Medio del Centro di Mitigazione

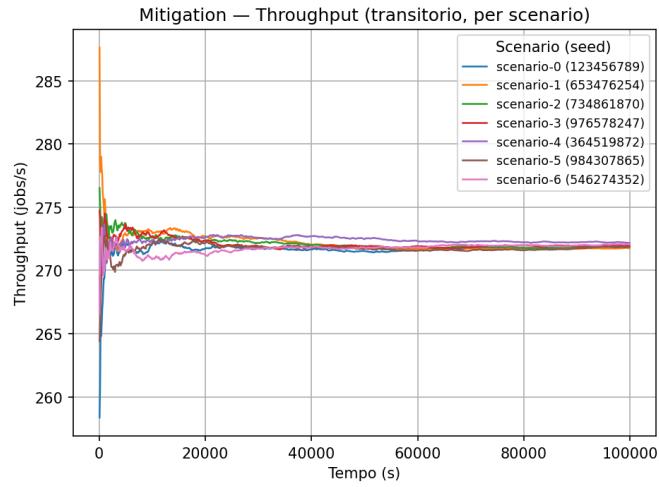


Figure 15: Throughput del Centro di Mitigazione

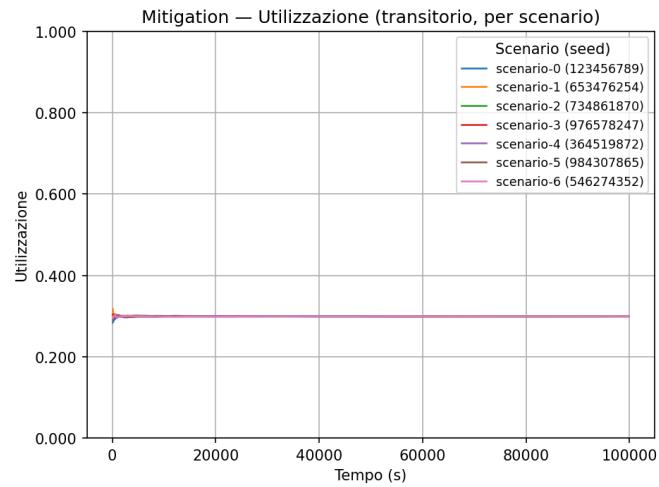


Figure 16: Utilizzazione del Centro di Mitigazione

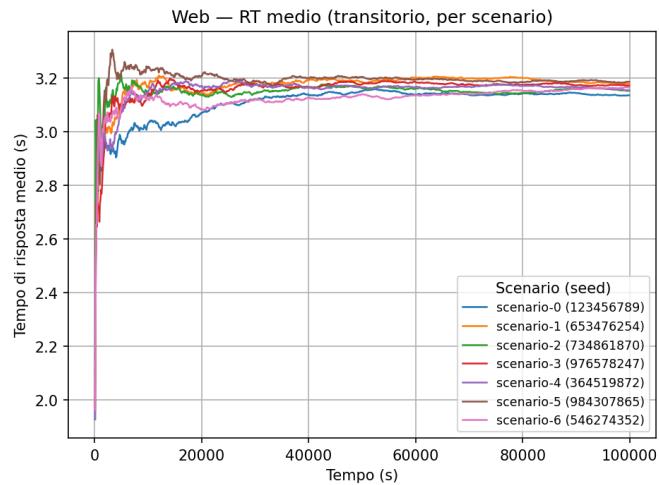


Figure 17: Tempo di Risposta Medio del Web Server

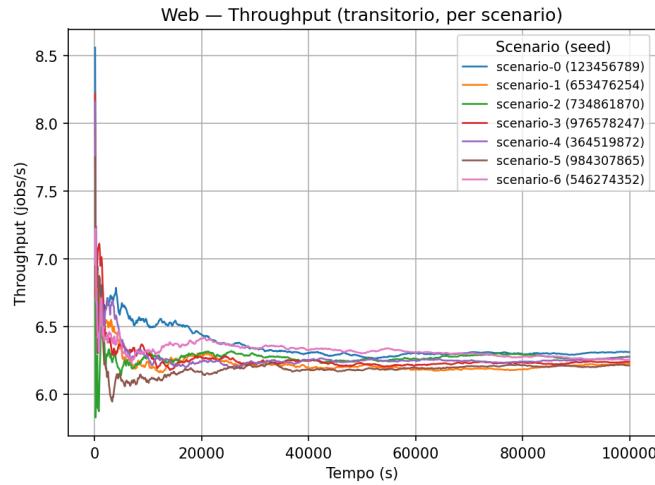


Figure 18: Throughput del Web Server

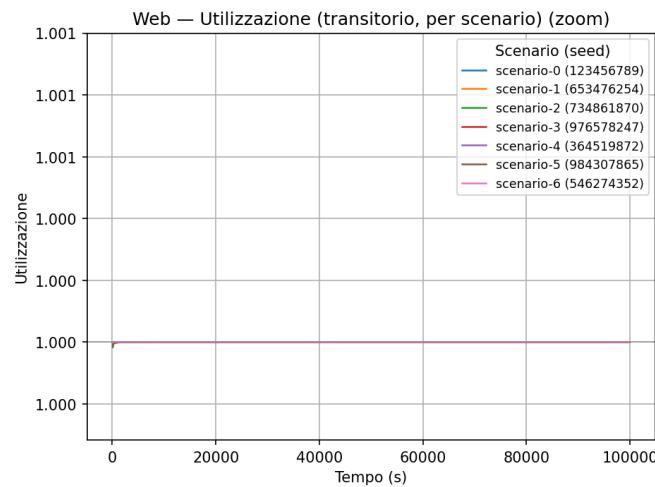


Figure 19: Utilizzazione del Web Server

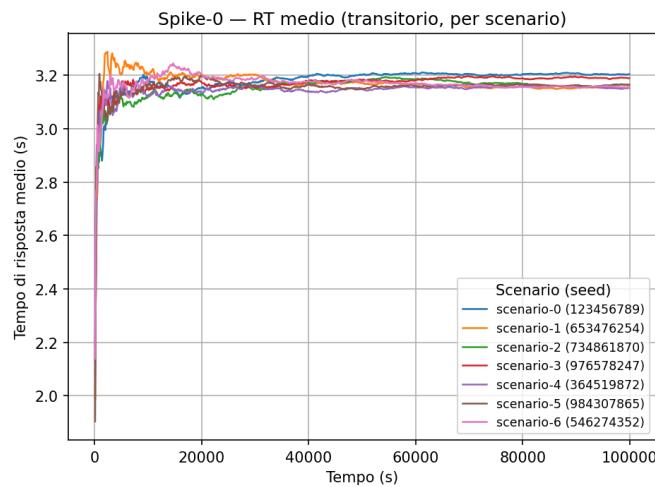


Figure 20: Tempo di Risposta Medio del primo Spike Server allocato

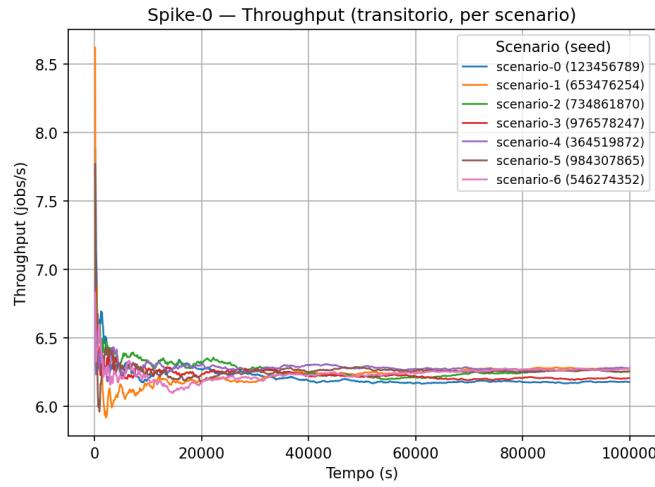


Figure 21: Throughput del primo Spike Server allocato

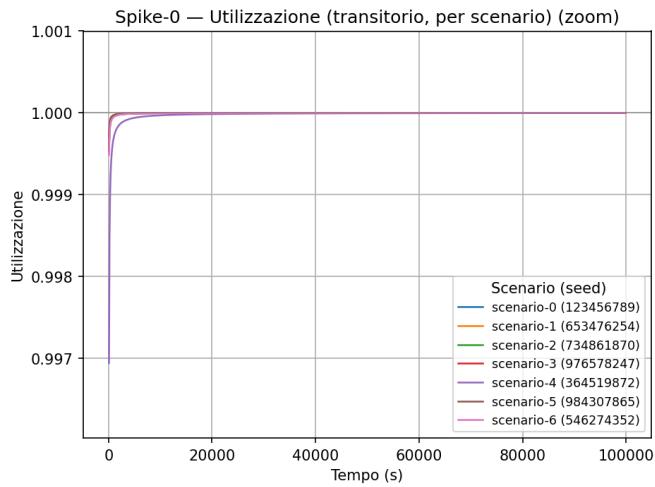


Figure 22: Utilizzazione del primo Spike Server allocato

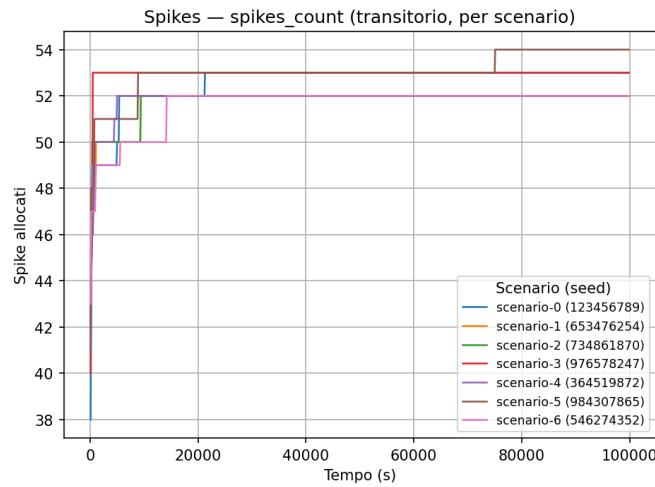


Figure 23: Numero di Spike Server allocati

Nella figura 14 è riportato l'andamento del tempo medio di risposta del Centro di Mitigazione: la curva mostra una rapida convergenza allo stato stazionario, con differenze tra le varie simulazioni inferiori a 0.00005. Il throughput (figura 15) conferma la stabilità del centro con discrepanze lievi. L'utilizzazione (figura 16) evidenzia un comportamento stazionario fin dalle prime fasi.

Per il Web Server (figure 17, 18 e 19) le tre metriche considerate convergono anch'esse a valori stabili, seppur differenti tra loro, a conferma dell'equilibrio raggiunto dal sistema.

Lo stesso vale per gli Spike Server (figure 20, 21, 22), i cui valori medi risultano molto vicini a quelli del Web Server: ciò mostra che il sistema tende a sfruttare al massimo la capacità disponibile.

Possiamo concludere che, in generale, il sistema raggiunge un comportamento stazionario. Solitamente, un attacco di tipo DDoS non supera la durata dell'ora in media [7], quindi abbiamo deciso di simulare il sistema per circa un giorno per verificare se il sistema convergesse ad uno stato stazionario. È importante sottolineare che il sistema parte con tutti i centri utilizzati vuoti.

### 8.3 Simulazione Orizzonte Finito

La simulazione a orizzonte finito è stata sviluppata seguendo l'approccio delle replicazioni indipendenti. Per assicurare l'indipendenza statistica tra le repliche, è stato adottato un meccanismo di gestione dei semi basato sulla classe `rngs`. La simulazione parte da un seme iniziale fissato (1234566789), e all'inizio di ogni replica, il generatore viene inizializzato con la chiamata `rngs.plantSeed()`. Al termine della replica, lo stato raggiunto dal generatore viene salvato tramite `rngs.getSeed()` e utilizzato come seme per la replica successiva. In questo modo ciascuna run lavora su una sequenza distinta di numeri casuali, evitando sovrapposizioni e garantendo che le osservazioni raccolte siano indipendenti. Durante l'esecuzione, vengono effettuati checkpoint a intervalli regolari di 100 secondi, che si susseguono fino al tempo massimo di simulazione, fissato a 5 ore. Di seguito vengono mostrati i risultati ottenuti con la simulazione.

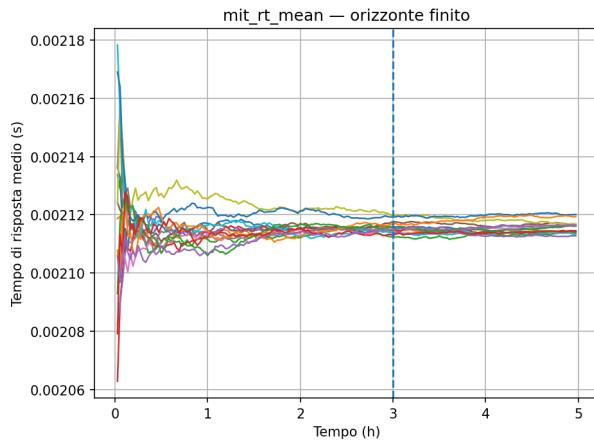


Figure 24: Tempo di risposta del Mitigation Center

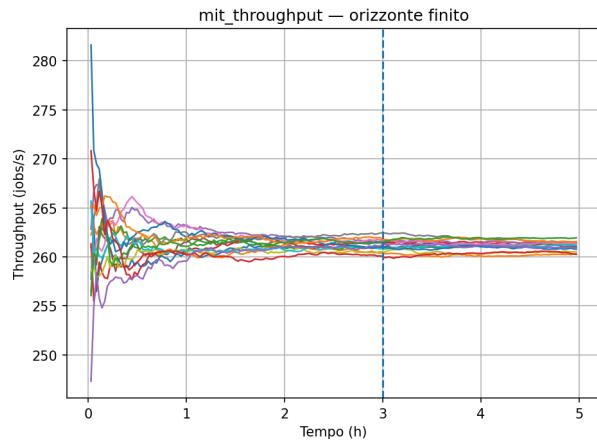


Figure 25: Throughput del Mitigation Center

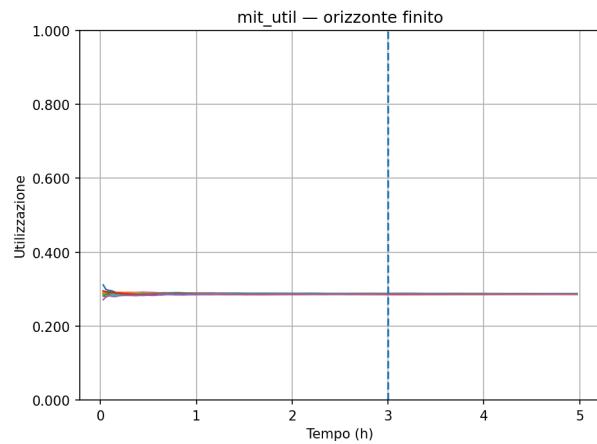


Figure 26: Utilizzazione del Mitigation Center

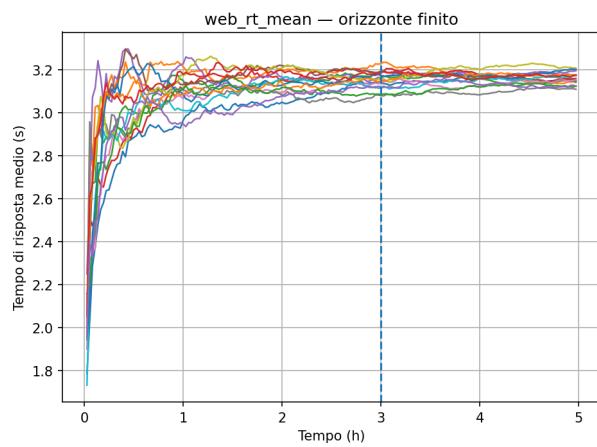


Figure 27: Tempo di Risposta del Web Server

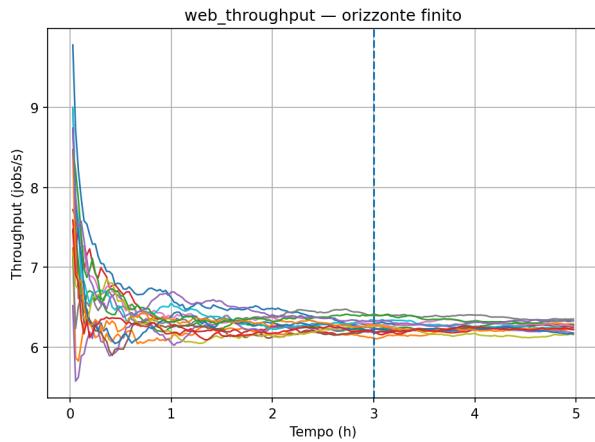


Figure 28: Throughput del Web Server

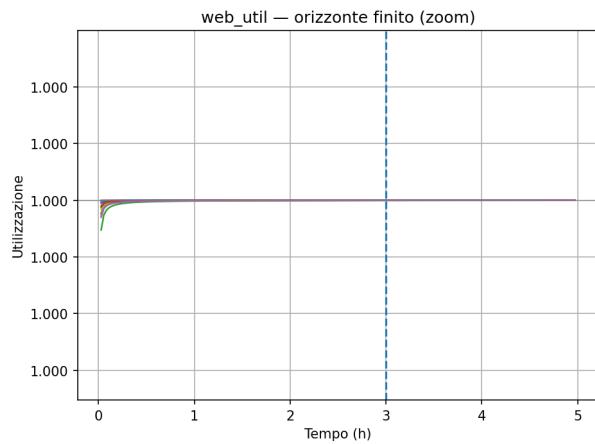


Figure 29: Utilizzazione del Web Server

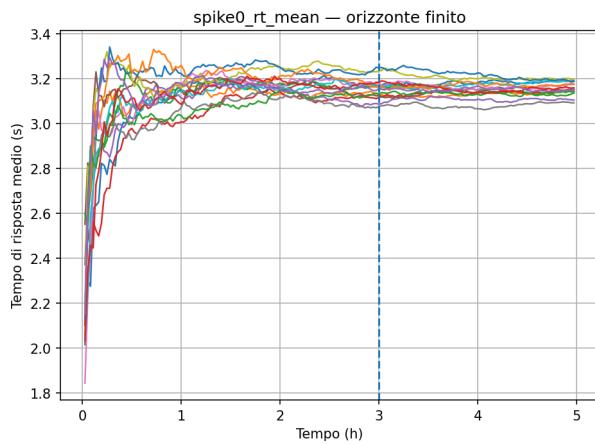


Figure 30: Tempo medio di Risposta del primo Spike Server allocato

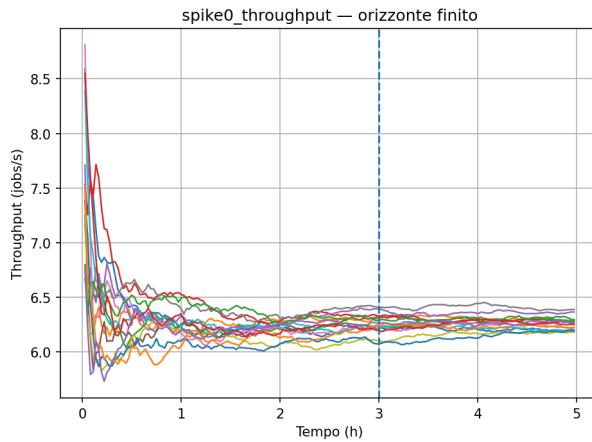


Figure 31: Throughput del primo Spike Server allocato

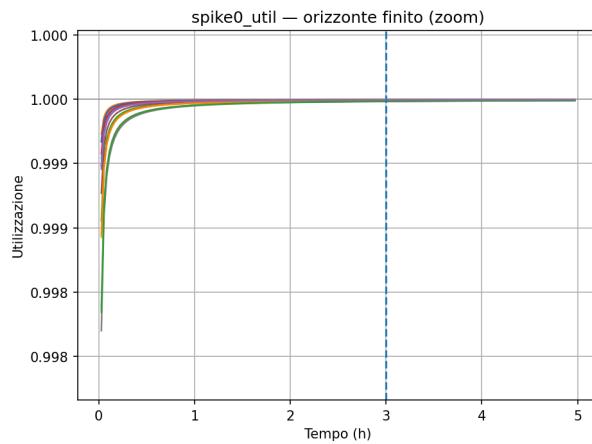


Figure 32: Utilizzazione del primo Spike Server allocato

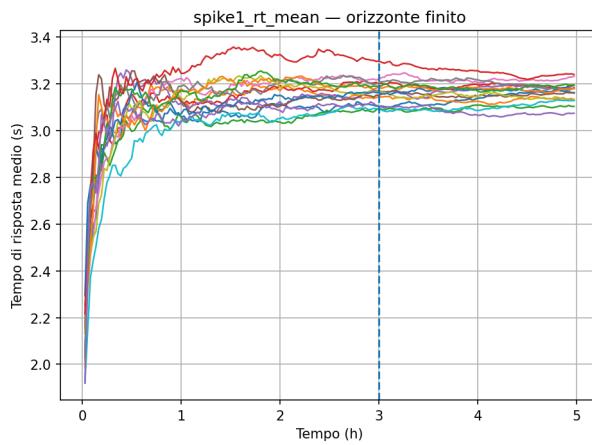


Figure 33: Tempo medio di Risposta del secondo Spike Server allocato

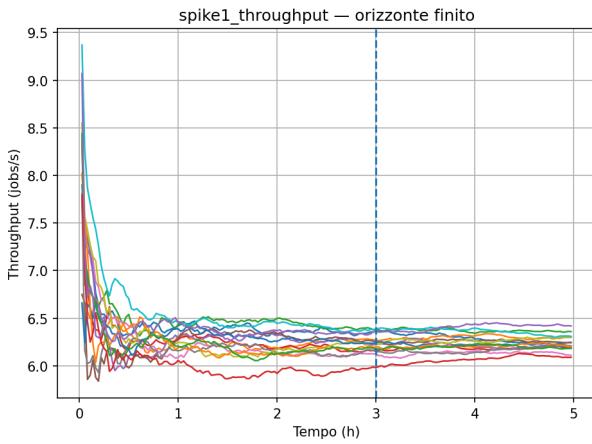


Figure 34: Throughput del secondo Spike Server allocato

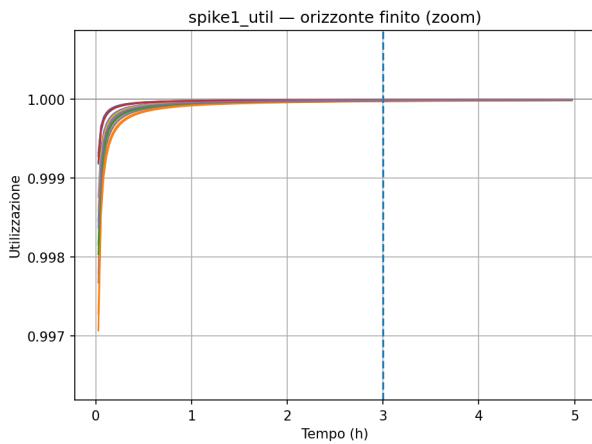


Figure 35: Utilizzazione del secondo Spike Server allocato

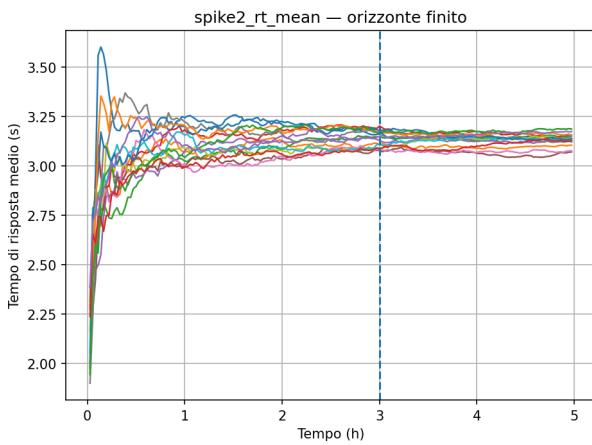


Figure 36: Tempo medio di Risposta del terzo Spike Server allocato

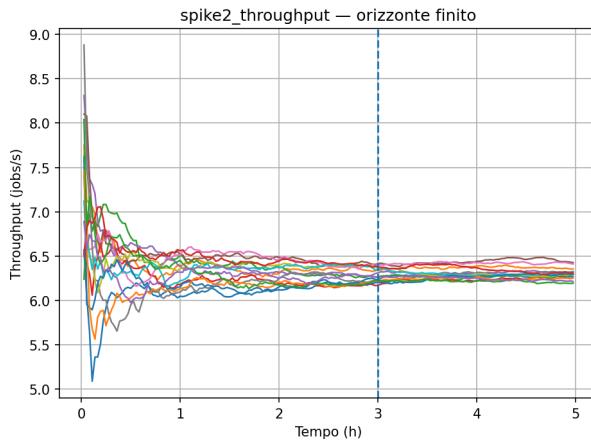


Figure 37: Throughput del terzo Spike Server allocato

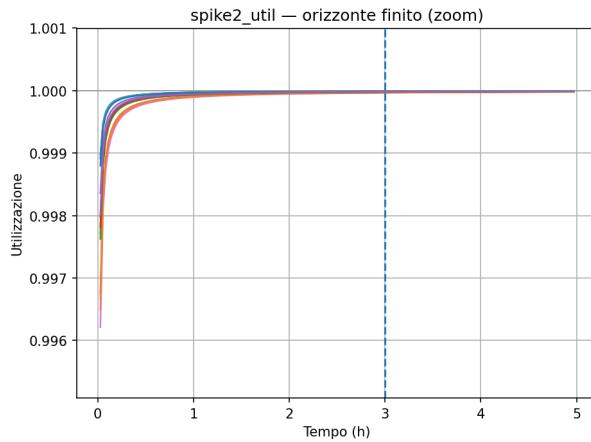


Figure 38: Utilizzazione del terzo Spike Server allocato

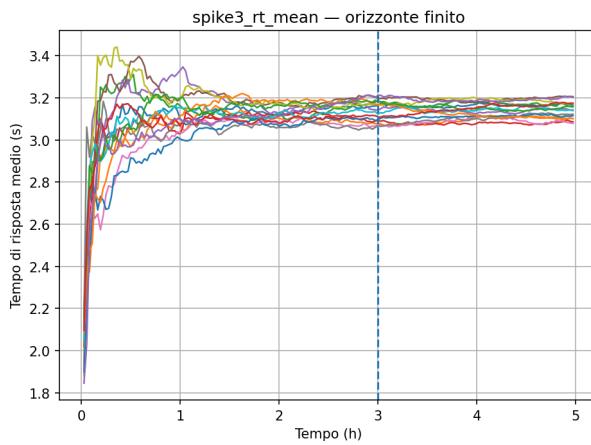


Figure 39: Tempo medio di Risposta del quarto Spike Server allocato

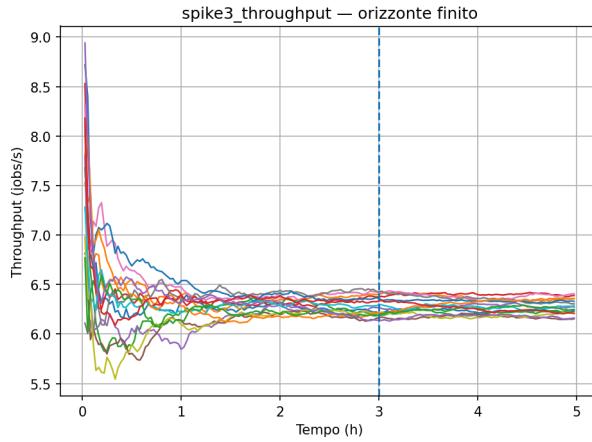


Figure 40: Throughput del quarto Spike Server allocato

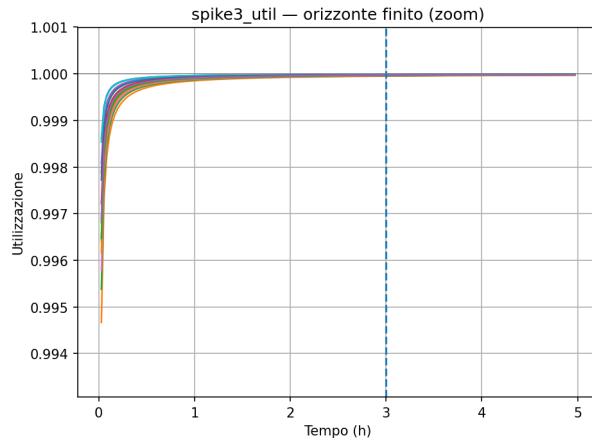


Figure 41: Utilizzazione del quarto Spike Server allocato

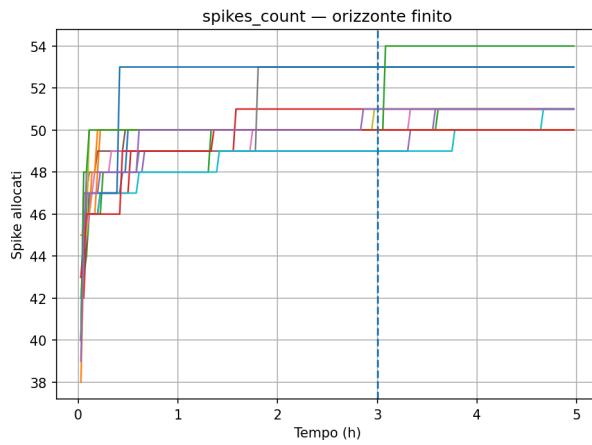


Figure 42: Numero di Spike Allocati

Per completezza, sono stati riportati i grafici dell’andamento del tempo medio di risposta, throughput e utilizzazione dei primi quattro Spike Server allocati.

Dall’analisi della figure emergono due considerazioni principali:

- 1. Qualità del servizio (QoS)** I tempi di risposta osservati risultano compatibili con i

vincoli di QoS definiti in precedenza. Ciò significa che l'architettura, pur in presenza di un elevato numero di richieste, riesce a mantenere prestazioni adeguate.

2. **Costo dell'infrastruttura** Per sostenere tale livello di servizio, il sistema ha allocato in media circa 52 Spike Server 42. Considerando una VM di tipo **r6a.xlarge** (costo orario pari a 0.19 euro) il dispendio economico è stimato in circa 49.4 euro giornalieri, pari a circa 186.2 euro mensili.

## 8.4 Simulazione Orizzonte Infinito

Per questo tipo di simulazione è stato utilizzato il metodo del *Batch Means*. Questa tecnica consiste nel far evolvere il sistema per un periodo di tempo sufficientemente lungo per consentirne l'ingresso in un regime stazionario, condizione in cui le statistiche di interesse non dipendono più dallo stato iniziale. Per eliminare l'influenza della fase di *warm-up*, i dati relativi ai primi 1.000 completamenti sono stati scartati dall'analisi.

Il campione è stato diviso in  $K = 64$  batch di dimensione  $B = 1256$  jobs.

Di seguito vengono mostrati i valori del tempo di risposta medio, throughput e utilizzazione per il Centro di Mitigazione, Web Server e per il primo Spike Server allocato.

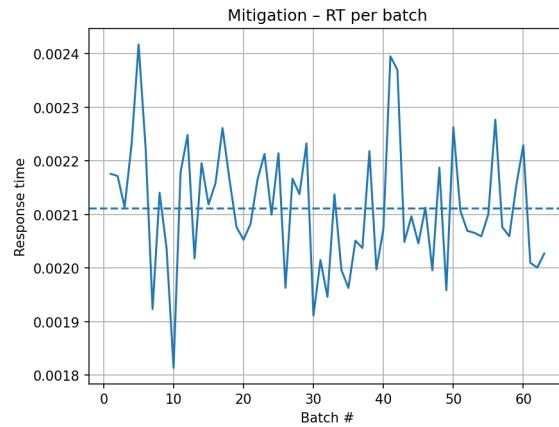


Figure 43: Tempo di Risposta Medio per il Mitigation Center

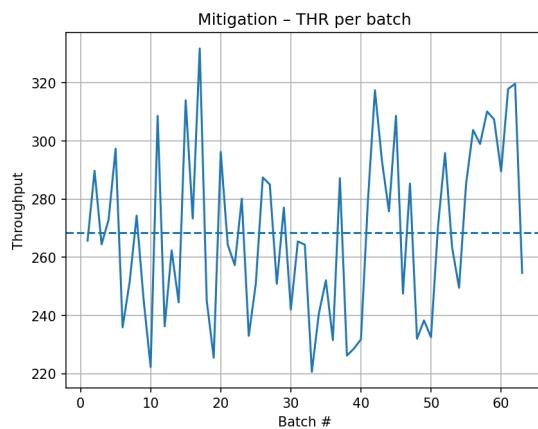


Figure 44: Throughput per il Mitigation Center

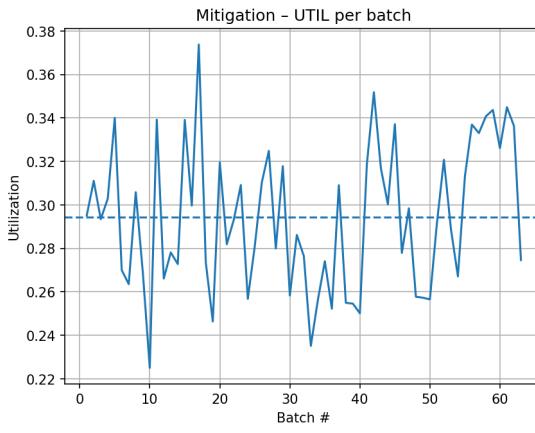


Figure 45: Utilizzazione per il Mitigation Center

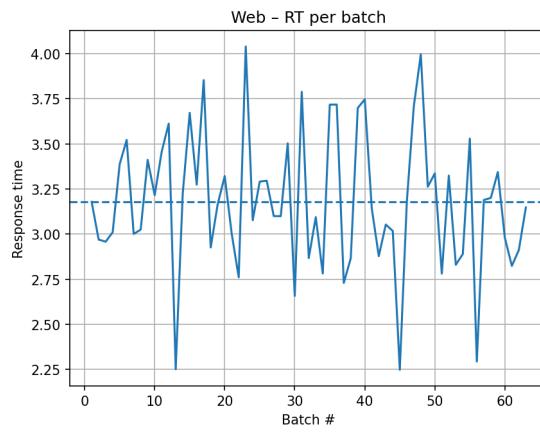


Figure 46: Tempo di Risposta Medio per il Web Server

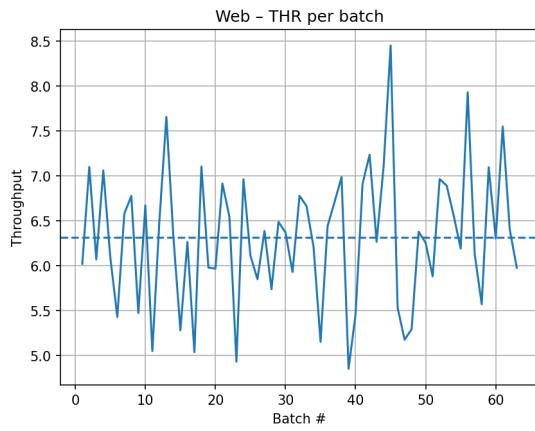


Figure 47: Throughput per il Web Server

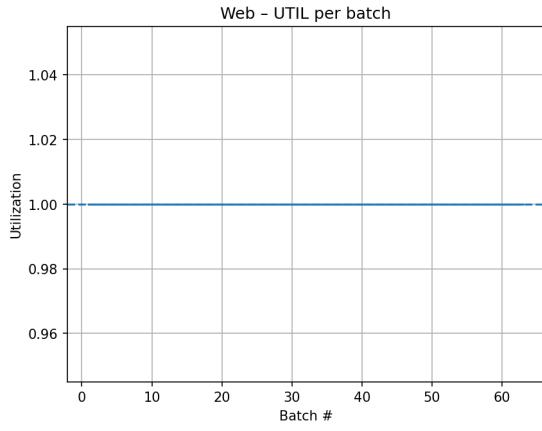


Figure 48: Utilizzazioni per il Web Server

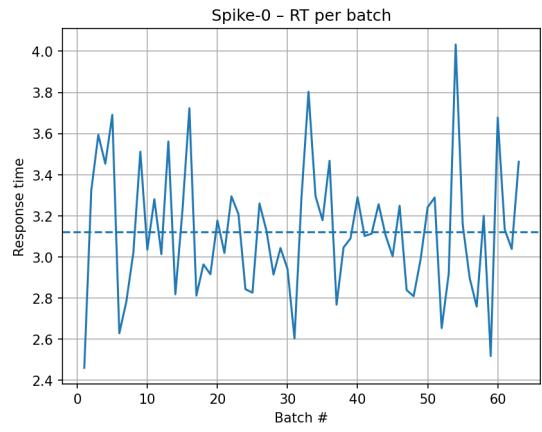


Figure 49: Tempo di Risposta Medio per il primo Spike Server allocato

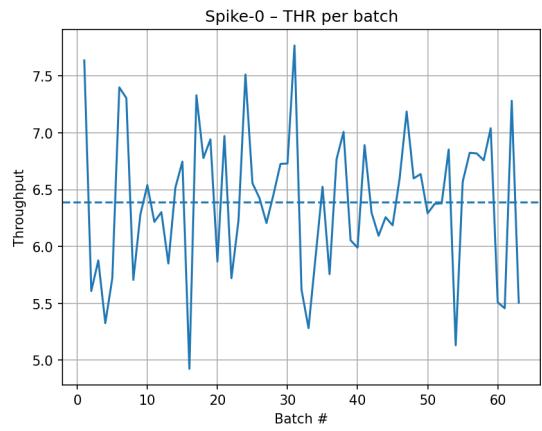


Figure 50: Throughput per il primo Spike Server allocato

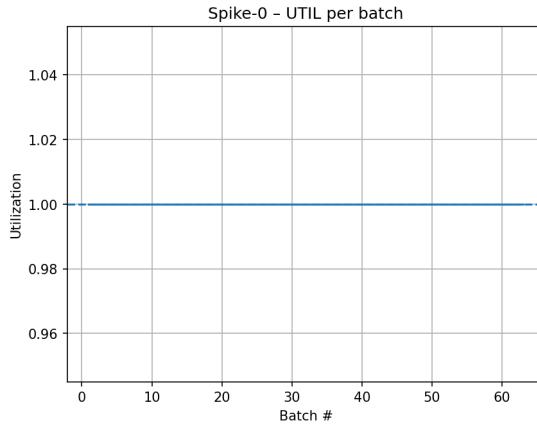


Figure 51: Utilizzazione per il primo Spike Server allocato

Analizzando i risultati ottenuti si nota come il comportamento con la simulaizone ad orizzonte infinito assumi lo stesso ottenuto con gli altri tipi di simulazione, in particolare si nota come il comportamento del primo Spike Server allocato sia pressochè identico a quello del Web Server; inoltre, i risultati confermano come il meccanismo di instradamento cerchi di utilizzare il più possibile le risorse a disposizione del sistema.

Dai risultati ottenuti con il metodo del *Batch Means* si nota come i valori medi siano molto simili a quelli ottenuti con le altre simulazioni.

Di seguito vengono mostrati in forma tabellare i risultati ottenuti.

Centro	Tempo Medio di Risposta	Throughput	Utilizzazione
Mitigation Center	$0.002432 \pm 0.000054$	$278.549740 \pm 8.299164$	$0.306251 \pm 0.009702$
Web Server	$3.161241 \pm 0.174340$	$6.290735 \pm 0.319857$	$0.999999 \pm 0.000000$
Spike Server - 0	$2.967894 \pm 0.176635$	$6.653072 \pm 0.303593$	$0.999999 \pm 0.000000$

Table 6: Tabella Riassuntiva delle metriche ottenute con la simulazione ad orizzonte infinito

### 8.4.1 Autocorrelazione

Di seguito è possibile vedere i valori per l'autocorrelazione delle statistiche fornite dal simulatore, che si appoggia sul programma `acs.py` fornito dalla libreria. Abbiamo incrementato progressivamente la dimensione del batch  $B$  fino a che l'autocorrelazione tra i vari batch è risultata inferiore alla soglia  $|r_j| < 2/\sqrt{n}$ , come proposto dalla regola empirica di Chatfield [5], con  $n$  pari al numero di batch utilizzati.

Sono riportati i valori ottenuti per tempo medio di risposta, utilizzazione e throughput per il Mitigation Center, il Web Server e per cinque Spike Server allocati a titolo esemplificativo. Per la lista completa delle metriche si rimanda al file `autocorrelation_results.txt` presente nella cartella `utils`.

```

1 mit_rt: rho_1=0.083  (|rho1|<0.252) → PASS
2 mit_util: rho_1=0.225  (|rho1|<0.252) → PASS
3 mit_thr: rho_1=0.172  (|rho1|<0.252) → PASS
4 web_rt: rho_1=-0.092  (|rho1|<0.252) → PASS
5 web_util: rho_1=0.000  (|rho1|<0.252) → PASS
6 web_thr: rho_1=-0.074  (|rho1|<0.252) → PASS
7 spike0_rt: rho_1=-0.056  (|rho1|<0.252) → PASS
8 spike0_util: rho_1=0.000  (|rho1|<0.252) → PASS
9 spike0_thr: rho_1=-0.098  (|rho1|<0.252) → PASS
10 spike1_rt: rho_1=0.085  (|rho1|<0.252) → PASS
11 spike1_util: rho_1=0.000  (|rho1|<0.252) → PASS
12 spike1_thr: rho_1=0.151  (|rho1|<0.252) → PASS
13 spike2_rt: rho_1=-0.233  (|rho1|<0.252) → PASS
14 spike2_util: rho_1=0.000  (|rho1|<0.252) → PASS
15 spike2_thr: rho_1=0.045  (|rho1|<0.252) → PASS
16 spike3_rt: rho_1=-0.057  (|rho1|<0.252) → PASS
17 spike3_util: rho_1=0.000  (|rho1|<0.252) → PASS
18 spike3_thr: rho_1=0.120  (|rho1|<0.252) → PASS
19 spike4_rt: rho_1=0.014  (|rho1|<0.252) → PASS
20 spike4_util: rho_1=0.000  (|rho1|<0.252) → PASS
21 spike4_thr: rho_1=-0.104  (|rho1|<0.252) → PASS
22 spike5_rt: rho_1=-0.056  (|rho1|<0.252) → PASS
23 spike5_util: rho_1=0.000  (|rho1|<0.252) → PASS
24 spike5_thr: rho_1=-0.037  (|rho1|<0.252) → PASS
25 spike6_rt: rho_1=0.042  (|rho1|<0.252) → PASS
26 spike6_util: rho_1=0.000  (|rho1|<0.252) → PASS
27 spike6_thr: rho_1=-0.012  (|rho1|<0.252) → PASS
28 spike7_rt: rho_1=0.102  (|rho1|<0.252) → PASS
29 spike7_util: rho_1=0.000  (|rho1|<0.252) → PASS
30 spike7_thr: rho_1=0.073  (|rho1|<0.252) → PASS
31 spike8_rt: rho_1=0.023  (|rho1|<0.252) → PASS
32 spike8_util: rho_1=0.000  (|rho1|<0.252) → PASS
33 spike8_thr: rho_1=0.238  (|rho1|<0.252) → PASS
34 spike9_rt: rho_1=-0.149  (|rho1|<0.252) → PASS
35 spike9_util: rho_1=0.000  (|rho1|<0.252) → PASS
36 spike9_thr: rho_1=-0.046  (|rho1|<0.252) → PASS
37 spike10_rt: rho_1=-0.009  (|rho1|<0.252) → PASS
38 spike10_util: rho_1=0.000  (|rho1|<0.252) → PASS
39 spike10_thr: rho_1=-0.002  (|rho1|<0.252) → PASS
40 spike11_rt: rho_1=0.016  (|rho1|<0.252) → PASS
41 spike11_util: rho_1=0.000  (|rho1|<0.252) → PASS
42 spike11_thr: rho_1=0.082  (|rho1|<0.252) → PASS
43 spike12_rt: rho_1=0.165  (|rho1|<0.252) → PASS
44 spike12_util: rho_1=0.000  (|rho1|<0.252) → PASS
45 spike12_thr: rho_1=0.005  (|rho1|<0.252) → PASS
46 spike13_rt: rho_1=0.111  (|rho1|<0.252) → PASS
47 spike13_util: rho_1=0.000  (|rho1|<0.252) → PASS
48 spike13_thr: rho_1=0.046  (|rho1|<0.252) → PASS

```

Figure 52: Valori dell'autocorrelazione

## 9 Modello Migliorativo

### 9.1 Considerazione e Limitazione del Modello di Base

I risultati ottenuti con il modello di base sono stati in generale positivi in termini di tempi di risposta medi per le richieste ma le principali limitazioni del modello di base sono relative al numero di Spike Server allocati, con il conseguente impatto economico, durante l'attacco e dalla percentuale di processamento e completamento delle richieste illecite del sistema.

Se si riuscisse a fare in modo che le due percentuali fossero nettamente inferiori allora il numero di richieste da processare sarebbe nettamente inferiore. In questo modo il numero di Spike Server allocati sarebbe molto più basso e il dispendio economico da affrontare durante l'impatto molto limitato.

A fronte delle limitazioni e dei risultati del modello di base, il modello migliorativo prevede l'aggiunta di un controllo aggiuntivo preliminare prima di inoltrare le richieste al Web o ai vari Spike Server allocati.

In particolare il nuovo centro aggiunto applica degli algoritmi di Machine Learning in grado di classificare con maggiore accuratezza le richieste in arrivo al sistema, tale centro non applica meccanismo di feedback e a tempo di inferenza è in grado di classificare il tipo della richiesta analizzata.

D'altro canto, l'aggiunta di questo tipo di centro non è banale: bisogna disporre delle risorse che siano sempre disponibili e non che siano utilizzabili on-demand, come nel caso del servizio di autoscaling.

Nel caso di studio consideriamo l'utilizzo di un modello preventivamente addestrato in grado di classificare una richiesta in arrivo a tempo di inferenza.

La modifica introdotta ha un duplice scopo:

- Abbassare la percentuale di richieste illecite processate da parte del Web Server e dai vari Spike Server
- Ridurre il numero di Spike Server allocati

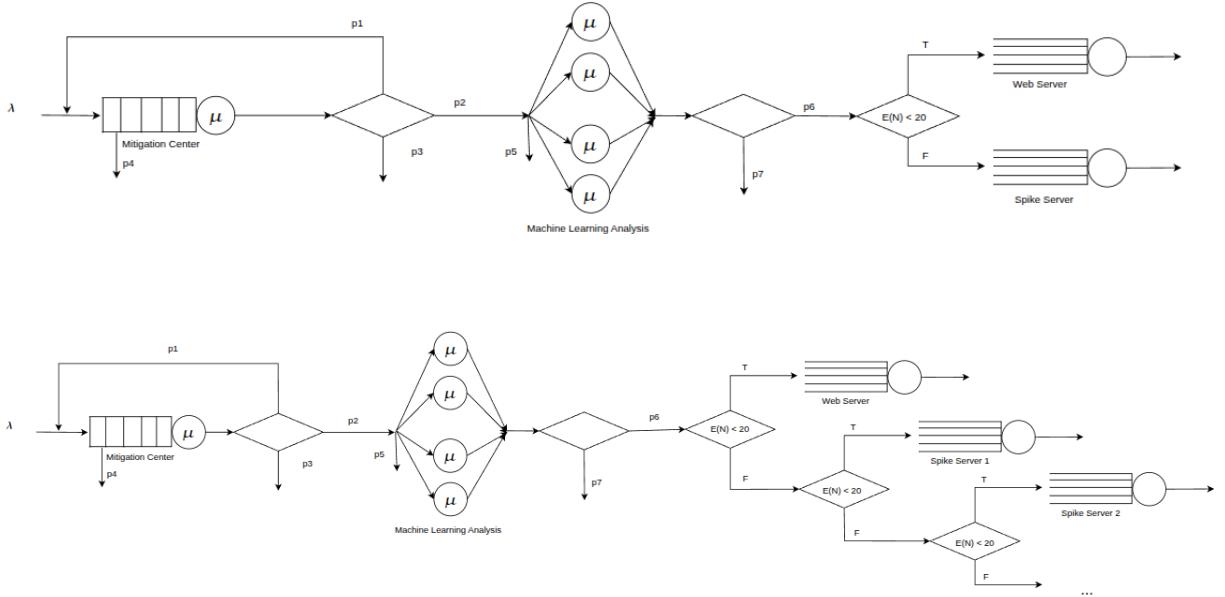
In questo modo si ha un beneficio in termini di guadagno economico per l'allocazione delle macchine per sopportare ai picchi di carico durante un attacco.

Per l'analisi del costo di tale centro, è stata assunta di utilizzare la stessa tipologia di macchina utilizzata per il Web Server e gli Spike Server, ovvero una `r6a.xlarge` con costo orario pari a 0.19 euro.

## 9.2 Modello Concettuale

Il centro aggiunto è un centro multi-server senza coda: se uno dei serventi è libero, il job viene sottoposto all'analisi e l'algoritmo di Machine Learning produce una predizione sulla natura del job. Se la richiesta viene classificata come illecita, la sua analisi sarà considerata conclusa ed uscirà dal sistema e quindi scartata. In caso contrario, la richiesta sarà verrà instradata verso il Web Server, o gli Spike Server allocati, per essere effettivamente servita. Se non ci sono serventi liberi, La richiesta viene scartata dal sistema.

Con l'aggiunta del centro nel modello migliorativo, il modello concettuale assume la seguente forma:



## 9.3 Eventi del Sistema

Oltre agli eventi precedentemente citati, vengono considerati anche gli eventi relativi al nuovo centro.

### 9.3.1 Machine Learning Analysis

- L'arrivo di una richiesta al centro si verifica quando il Mitigation Center ha classificato lecita come la richiesta analizzata. L'arrivo **non rappresenta un arrivo al sistema**.
- Il completamento dell'analisi del centro può portare a due possibili esiti:
  - la richiesta è stata classificata come *lecita* e viene instradata verso uno dei centri per l'elaborazione;
  - la richiesta è stata classificata come *illecita* e viene scartata dal sistema.

Il completamento in questo centro **non coincide** con un completamento del sistema.

## 9.4 Stato del Sistema

Lo stato del sistema, come detto precedentemente, è rappresentato dallo stato dei singoli centri e alla tupla indicata nel modello di base viene aggiunta l'indicazione dello stato del centro di analisi aggiuntivo.

Il centro aggiunto è un centro multiserver senza coda, di conseguenza il suo stato è definito dal numero totale dei serventi.

Lo stato del sistema, quindi, è descritto da una tupla formata nel seguente modo  $(i, w, m, s_1, \dots, s_n)$  dove:

- $i \in \{0, 1, 2, 3, 4, \dots, 1025\}$ : numero di job nel centro di mitigazione. Il numero job in questo centro è costituito dalla somma delle richieste poste in coda e dalla richiesta che viene processata.
- $w \in \{0, 1, \dots, 20\}$ : numero di job nel Web server.
- $m \in \{0, 1, 2, 3, 4\}$ : numero di job nel centro di analisi in cui viene applicato l'algoritmo di machine learning. Il numero job in questo centro è costituito dalle richieste che possono essere processate dal centro.
- $s_i \in \{0, 1, \dots, 20\}$ : numero di job nel Spike server.

## 9.5 Modello delle Specifiche

Per i parametri utilizzati per il modello migliorativo, si utilizzano tutti i parametri utilizzati per il modello di base per i centri in comune.

Per il nuovo centro, sono stati considerati i valori delle tempistiche riscontrate utilizzando come algoritmo di Machine Learning KNN, K Nearest Neighbors, in cui sono state ottenute le seguenti misure relative al tempo di inferenza.

```

knn = KNeighborsClassifier(n_neighbors=10, weights='uniform', metric="manhattan")
knn.fit(X_train_scaled, y_train)
start_time = time.perf_counter()
y_pred = knn.predict(X_test_scaled)
end_time = time.perf_counter()

# Tempo totale e medio
inference_time_total = end_time - start_time
inference_time_avg = inference_time_total / len(X_test_scaled)

print(f"Tempo totale di inferenza su {len(X_test_scaled)} campioni: {inference_time_total:.6f} secondi")
print(f"Tempo medio di inferenza per campione (E[S]): {inference_time_avg:.8f} secondi")

train_accuracy_knn = knn.score(X_train_scaled, y_train)
print(f"Training Accuracy: {train_accuracy_knn:.4f}")

test_accuracy_knn = accuracy_score(y_test, y_pred)
print(f"Testing Accuracy: {test_accuracy_knn:.4f}")

print("\nClassification Report (Testing Set):\n", classification_report(y_test, y_pred))

plot_confusion_matrix(y_true=y_test, y_pred=y_pred, labels=knn.classes_)
plot_roc_curve(knn, X_test, y_test)

```

Tempo totale di inferenza su 25195 campioni: 15.459075 secondi  
 Tempo medio di inferenza per campione (E[S]): 0.000061358 secondi  
 Training Accuracy: 0.9955  
 Testing Accuracy: 0.9939

Classification Report (Testing Set):				
	precision	recall	f1-score	support
0	0.99	0.99	0.99	11773
1	0.99	0.99	0.99	13422
accuracy			0.99	25195
macro avg	0.99	0.99	0.99	25195
weighted avg	0.99	0.99	0.99	25195

Tale algoritmo è in grado di fornire un'accuratezza pari al 99.39% per la classificazione binaria delle richieste. Tali misurazioni sono state ottenute tramite il progetto del corso di Machine Learning del nostro corso di Laurea, il cui scopo era quello di addestrare un modello di ML basato sul dataset KDD, contenente dati relativi ad attacchi DDoS reali [6].

Considerando la VM AWS con 4 vCPUs, otteniamo i seguenti valori per il centro aggiunto per il modello migliorativo:

- numero di core del centro: 4;
- tasso di servizio medio del centro:  $\mu = 1'629.77 \text{ job/s}$ ;
- tasso di servizio medio del singolo core:  $\mu_i = \mu/4 = 407.4425 \text{ job/s}$ ;
- tempo medio di risposta del singolo core  $E(S_i) = 0.002454 \text{ s}$ ;
- probabilità che una richiesta venga classificata come illegale e quindi scartata:  $p_7 = 0,89504\%$ ;
- probabilità che una richiesta venga classificata come legale e quindi inoltrata:  $p_6 = 0.10496\%$ .

È stata utilizzata la distribuzione esponenziale per la distribuzione del tasso di servizio, scelta motivata dalla capacità della distribuzione di rappresentare in modo semplice l'elevata variabilità dei tempi di processamento dei pacchetti, in assenza di una distribuzione empirica misurata.

## 9.6 Modello Computazionale

Il modello computazionale non subisce grandi variazioni rispetto al modello di base, viene aggiunta all'interno della directory `model` la classe `AnalysisCenterML` che rappresenta il centro di analisi aggiunto nel modello migliorativo.

Un'altra modifica che viene fatta è la funzione di instradamento delle richieste, questo perché dopo l'analisi al Centro di Mitigazione le richieste vengono instradate verso il centro aggiunto, che poi instraderà le richieste verso il Web Server o gli Spike Server allocati per processare effettivamente le richieste.

### 9.6.1 Gestione degli Eventi

Per la gestione degli eventi, la funzione di instradamento assume la seguente forma.

Funzione `MitigationManager._mitigation_process()`.

```

1  def _mitigation_process(self, job):
2      try:
3          self.center.arrival(job)
4      except Exception:
5          return
6
7      now = self.env.now
8
9      selectStream(RNG_STREAM_FALSE_POSITIVE)
10     if random() < P_FALSE_POSITIVE:
11         self.metrics["false_positives"] = self.metrics.get("false_positives",
12             0) + 1
13         if job.is_legal:
14             self.metrics["false_positives_legal"] = self.metrics.get(
15                 "false_positives_legal", 0) + 1
16
17     selectStream(RNG_STREAM_FEEDBACK)
18     if random() < P_FEEDBACK_VERIFICATION:
19         job.arrival = now
20         self.handle_job(job)

```

```

20     return
21
22     if self.analysis_center is not None:
23         job.arrival = now # arrivo locale al centro di analisi
24         accepted = self.analysis_center.arrival(job)
25         if not accepted:
26             self.metrics["discarded_analysis_capacity"] += 1
27         return
28
29     self._forward_to_service_tier(job, now)

```

Listing 11: Funzione \_mitigation\_process

#### Funzione MitigationManager.\_after\_analysis()

```

1 def _after_analysis(self, job, completion_time):
2     selectStream(RNG_STREAM_ML_DECISION)
3     if random() < P_DROP_ML:
4         if job.is_legal:
5             self.metrics["ml_drop_legal"] += 1
6         else:
7             self.metrics["ml_drop_illegal"] += 1
8         return
9     else:
10        if job.is_legal:
11            self.metrics["ml_pass_legal"] += 1
12            self._forward_to_service_tier(job, completion_time)
13        else:
14            self.metrics["ml_pass_illegal"] += 1
15            self._forward_to_service_tier(job, completion_time)
16    return

```

Listing 12: Funzione \_after\_analysis

Per la gestione degli eventi nel centro aggiunto sono le seguenti.

#### Funzione AnalysisCenterML.arrival().

```

1 def arrival(self, job):
2     core_id = self._pick_free_core()
3     if core_id is None:
4         self.metrics["discarded_analysis_capacity"] = self.metrics.get("discarded_analysis_capacity", 0) + 1
5     return False
6
7     self.core_busy[core_id] = True
8     start = self.env.now
9     self.core_busy_periods[core_id].append([start, None])
10
11    selectStream(RNG_STREAM_ANALYSIS_SERVICE)
12    svc = Exponential(ANALYSIS_ES_CORE)
13
14    def _serve():
15        yield self.env.timeout(svc)
16        now = self.env.now
17
18        periods = self.core_busy_periods[core_id]
19        if periods and periods[-1][1] is None:
20            periods[-1][1] = now
21        self.core_busy[core_id] = False
22

```

```

23     rt_local = now - job.arrival
24     self.completed_jobs.append(rt_local)
25     self.completion_times.append(now)
26     self.total_completions += 1
27     if job.is_legal: self.legal_completions += 1
28     else:           self.illegal_completions += 1
29
30     if self.on_complete:
31         self.on_complete(job, now)
32
33     self.env.process(_serve())
34     return True

```

Listing 13: Funzione arrival

Funzione `AnalysisCenterML.update()`.

```

1 def update(self, now):
2     for core_id in range(self.cores):
3         periods = self.core_busy_periods[core_id]
4         if periods and periods[-1][1] is None:
5             periods[-1][1] = now
6             periods.append([now, None])

```

Listing 14: Funzione update

## 9.7 Verifica

Per la verifica del modello migliorativo, è stato utilizzato lo stesso approccio e ipotesi del modello di base.

### 9.7.1 Parametri del modello

I nuovi parametri del centro di analisi *ML* includono:

- $\mu_{\text{single\_core}} = 407.4425 \text{ job/s}$ : tasso di servizio di ciascun core (con  $c = 4$  server paralleli).
- $p_{\text{drop\_ml}} = 0,89504$ : probabilità media che un job venga scartato al termine del servizio di analisi.
- $p_{\text{pass}} = 0.10496$ : probabilità media che un job superi l'analisi e venga instradato verso i server Web/Spike.

### 9.7.2 Spazio degli Stati

Lo stato del sistema è descritto dalla seguente tupla  $(i, m, w, s)$ , dove:

- $i \in \{0, 1, 2, 3, 4\}$ : numero di job nel centro di mitigazione;
- $m \in \{0, 1, 2, 3, 4\}$ : numero di job nel centro di analisi di machine learning;
- $w \in \{0, 1, \dots, 20\}$ : numero di job nel Web Server;
- $s \in \{0, 1, \dots, 20\}$ : numero di job nello Spike Server.

Lo spazio degli stati  $\mathcal{S}$  è quindi dato da:

$$\mathcal{S} = \{(i, m, w, s) | 0 \leq i \leq 4, 0 \leq m \leq 4, 0 \leq w \leq 20, 0 \leq s \leq 20\}$$

Il numero totale di stati è  $5 \times 5 \times 21 \times 21 = 11025$ .

### 9.7.3 Transazione del processo

Il processo è a tempo continuo e presenta le stesse transizioni del modello di base con l'aggiunta delle transizioni del centro aggiunto:

- **Completamento nel centro di Mitigation Center:**
  - Con probabilità  $p_{\text{discard}}$ :  $(i, a, w, s) \rightarrow (i - 1, a, w, s)$  con tasso  $\mu_{\text{mit}} \cdot p_{\text{discard}}$ .
  - Con probabilità  $p_{\text{forward}}$ :
    - \* Se  $a < 4$ :  $(i, a, w, s) \rightarrow (i - 1, a + 1, w, s)$  con tasso  $\mu_{\text{mit}} \cdot (p_{\text{forward}})$ .
    - \* Se  $a = 4$  (ML Analysis pieno):  $(i, a, w, s) \rightarrow (i - 1, a, w, s)$  con tasso  $\mu_{\text{mit}} \cdot (p_{\text{discard}})$ .
- **Completamento ML Analysis:**
  - Drop ML:  $(i, a, w, s) \rightarrow (i, a - 1, w, s)$  con tasso  $\mu_{\text{ML}}(a) \cdot p_{\text{drop\_ml}}$ .
  - Instradamento verso PS con probabilità  $p_{\text{pass}}$ :
    - \* Se  $w < K_{\text{web}}$ :  $(i, a, w, s) \rightarrow (i, a - 1, w + 1, s)$  con tasso  $\mu_{\text{ML}}(a) \cdot p_{\text{pass}}$ .
    - \* Altrimenti, se  $s < K_{\text{spike}}$ :  $(i, a, w, s) \rightarrow (i, a - 1, w, s + 1)$  con tasso  $\mu_{\text{ML}}(a) \cdot p_{\text{pass}}$ .
    - \* Se  $w = K_{\text{web}}$  e  $s = K_{\text{spike}}$ : drop per capacità del sistema

$$(i, a, w, s) \rightarrow (i, a - 1, w, s) \quad \text{con tasso } \mu_{\text{ML}}(a) \cdot p_{\text{pass}}.$$

#### 9.7.4 Risoluzione della Catena di Markov

Generalizziamo ora il modello includendo il centro di analisi *ML*. Le equazioni di bilanciamento assumono la forma generale:

$$\pi_{i,a,w,s} \cdot \left( 1_{\{i < K_{\text{mit}}\}} \lambda + 1_{\{i > 0\}} \mu_{\text{mit}} + 1_{\{a > 0\}} \mu_{\text{ML}}(a) + 1_{\{w > 0\}} \mu_w + 1_{\{s > 0\}} \mu_s \right) = \\ = \sum_{(i',a',w',s')} \pi_{i',a',w',s'} q_{(i',a',w',s') \rightarrow (i,a,w,s)}$$

dove  $q_{x \rightarrow y}$  rappresenta i tassi di transizione elencati in precedenza (Arrivi, completamenti in Mitigation, ML Analysis, Web e Spike).

La condizione di normalizzazione è:

$$\sum_{i=0}^{K_{\text{mit}}} \sum_{a=0}^c \sum_{w=0}^{K_{\text{web}}} \sum_{s=0}^{K_{\text{spike}}} \pi_{i,a,w,s} = 1.$$

#### 9.7.5 Risultati Ottenuti

Nelle prossime due figure vengono mostrati i risultati ottenuti tramite il risolutore della catena di Markov e la verifica del simulatore.

```
=====
MODELLO ANALITICO - Variante MIGLIORATIVA (con Analysis Center ML)
=====

UTILIZZAZIONI:
Mitigation : 0.007334
Analysis   : 0.004050
Web         : 0.110838
Spike       : 0.000000

THROUGHPUT (job/s):
Mitigation : 6.666666
Analysis   : 6.599999
Web         : 0.692736
Spike       : 0.000000

TEMPI DI RISPOSTA (s):
Mitigation : 0.001108
Analysis   : 0.002454
Web         : 0.179945
Spike       : 0.000000

===== INTERVALLI DI CONFIDENZA =====

-- Web Server --
Response Time:          : 0.182167 ± 0.004502 (95% CI)
Utilization:            : 0.108293 ± 0.002674 (95% CI)
Throughput:             : 0.680729 ± 0.016281 (95% CI)

-- Spike Server --
Spike-0:
Response Time:          : 0.000000 ± 0.000000 (95% CI)
Utilization:            : 0.000000 ± 0.000000 (95% CI)
Throughput:             : 0.000000 ± 0.000000 (95% CI)

-- Mitigation Center --
Response Time:          : 0.001098 ± 0.000012 (95% CI)
Utilization:            : 0.007301 ± 0.000078 (95% CI)
Throughput:             : 6.643646 ± 0.048400 (95% CI)

-- Analysis Center --
Response Time:          : 0.002460 ± 0.000016 (95% CI)
Utilization:            : 0.004041 ± 0.000039 (95% CI)
Throughput:             : 6.575104 ± 0.047163 (95% CI)
```

Si nota come i valori analitici siano tutti compresi all'interno degli intervalli di confidenza ottenuti con il simulatore.

## 9.8 Validazione

Nella fase di validazione si vuole verificare che il modello implementato sia consistente rispetto al modello migliorativo analizzato. Possiamo fare delle considerazioni analoghe a quelle fatte per il modello base e analizzare il comportamento del sistema al variare dei fattori moltiplicativi applicati a  $\lambda$ , osservando se le metriche ottenute seguono l'andamento atteso.

### 9.8.1 Mitigation Center

Nelle prossime figure vengono rappresentate il tempo di risposta, il throughput e utilizzazione del Mitigation Center.

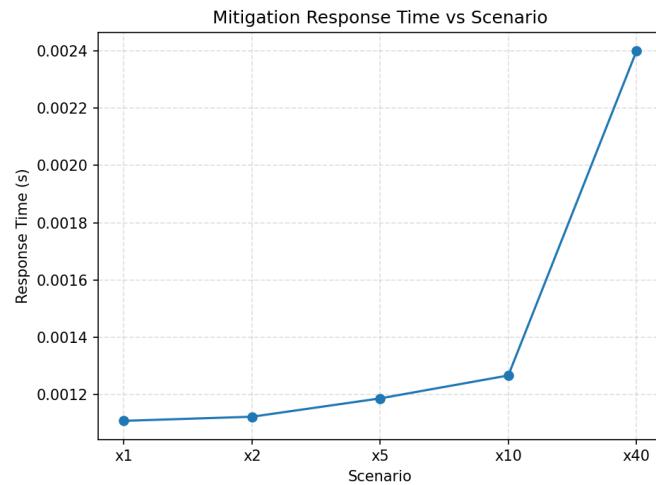


Figure 53: Tempo Medio di Risposta per il Mitigation Center al variare del tasso di arrivo

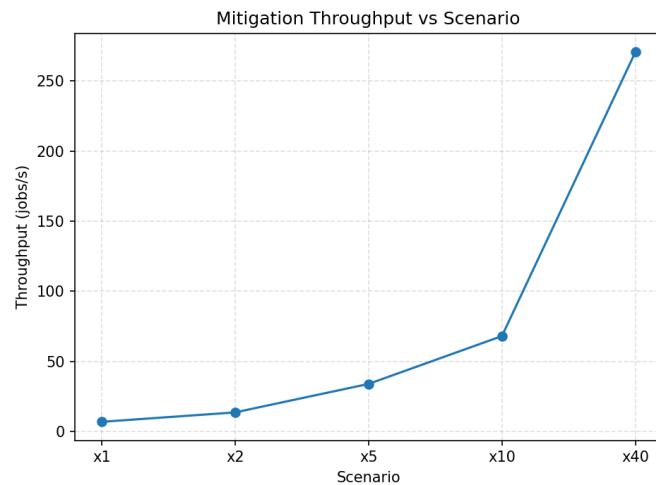


Figure 54: Throughput per il Mitigation Center al variare del tasso di arrivo

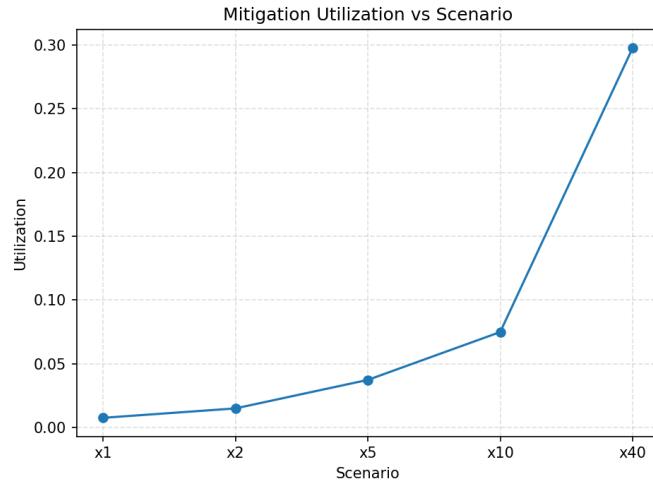


Figure 55: Utilizzazione per il Mitigation Center al variare del tasso di arrivo

All'aumentare di  $\lambda$ , si osserva un incremento coerente di tutte le metriche considerate. Tale comportamento conferma l'andamento atteso del modello e contribuisce a validare questa parte del sistema.

Scenario	Tempo Medio di Risposta	Throughput	Utilizzazione
<b>x1</b>	0.19093476182124253	0.7010022492324693	0.11349156542703687
<b>x2</b>	0.22719778380122654	1.3820514438204183	0.22284382078824513
<b>x5</b>	0.42510922565726444	3.4510409023132564	0.5500213338146339
<b>x10</b>	1.9372214389175135	6.174442497051954	0.9767906137231158
<b>x40</b>	3.1917549311183917	6.065995432855839	0.9966174453836792

Table 7: Tabella Riassuntiva delle metriche ottenute per il Mitigation Center

### 9.8.2 Machine Learning Analysis

Nella successiva figura si nota l'andamento del tempo di risposta del centro aggiunto con il modello migliorativo all'aumentare del carico di lavoro.

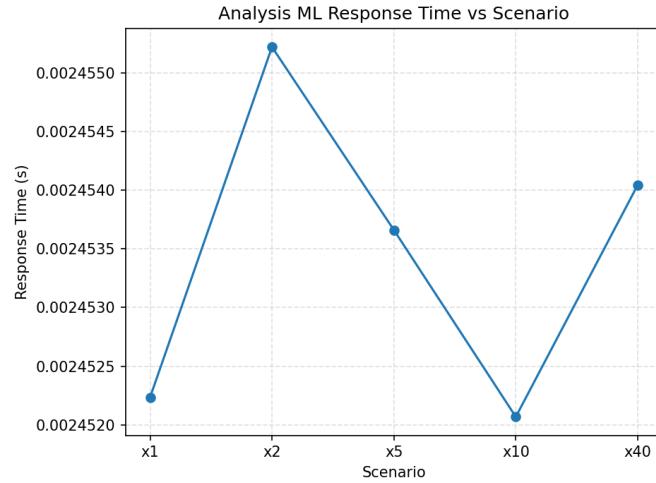


Figure 56: Tempo Medio di Risposta per il Machine Learning Analysis Center al variare del tasso di arrivo

Le variazioni apparenti, che a un primo impatto potrebbero sembrare un incremento o un decremento significativo, risultano in realtà dell'ordine di  $10^{-6}$  secondi. Tale comportamento è stato assunto come un comportamento dovuto al rumore statistico ed errori di approssimazione. Quindi il tempo medio di risposta rimane costante.

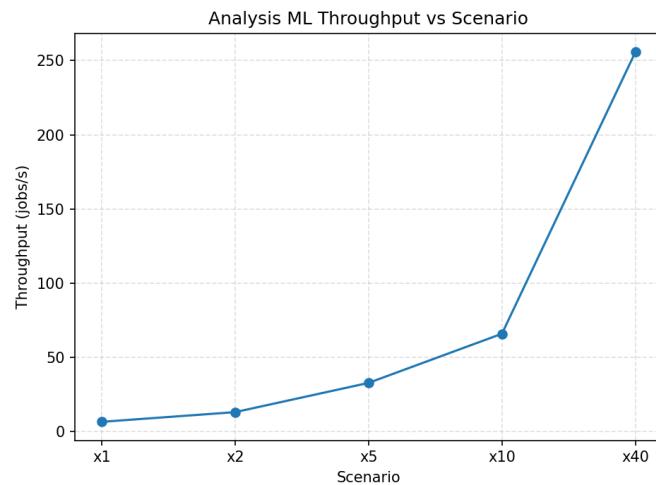


Figure 57: Throughput per il Machine Learning Analysis Center al variare del tasso di arrivo

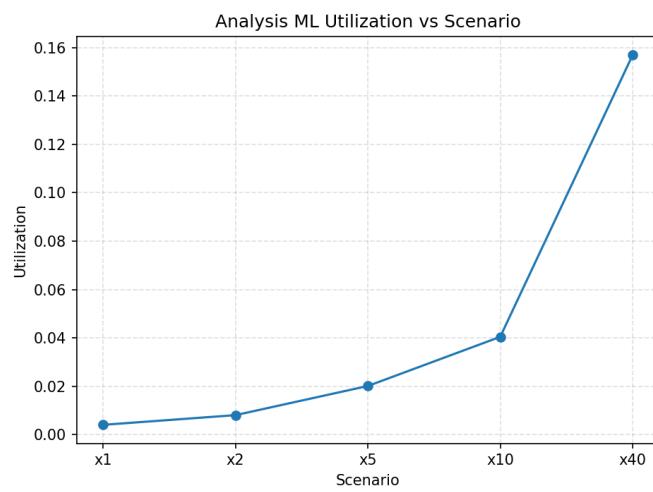


Figure 58: Utilizzazione per il Machine Learning Analysis Center al variare del tasso di arrivo

Scenario	Tempo Medio di Risposta	Throughput	Utilizzazione
<b>x1</b>	0.002452234378617207	6.670560454812777	0.004089444417984082
<b>x2</b>	0.0024552210502132455	13.176999937473338	0.00808811190628579
<b>x5</b>	0.0024536604539080963	32.8995908945598	0.020181106281934067
<b>x10</b>	0.0024520676602178284	65.9118739258113	0.040405093619459154
<b>x40</b>	0.0024540448151439613	254.08961096107276	0.1558868230902417

Table 8: Tabella Riassuntiva delle metriche ottenute per il Machine Learning Analysis Center

I comportamenti osservati corrispondono a quelli supposti, quindi validano questa parte di sistema.

### 9.8.3 Web Server

Nella prossima figura viene rappresentato il tempo di risposta medio del Web Server al variare dei valori del tempo di interarrivo delle richieste.

Il comportamento ottenuto è un comportamento atteso: più richieste arrivano al sistema e più il tempo di risposta medio tenderà ad aumentare.

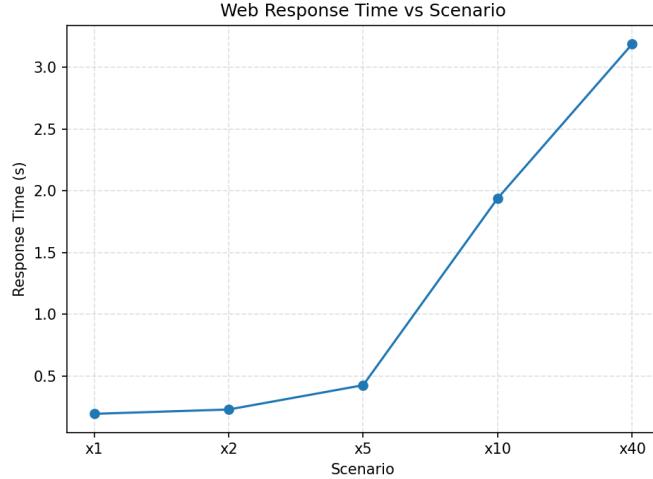


Figure 59: Tempo Medio di Risposta per il Web Server al variare del tasso di arrivo

Nella prossima figura vengono mostrati i valori del throughput del centro di mitigazione ottenuti al variare del tempo di interarrivo. Anche in questo caso il comportamento ottenuto è consistente: maggiore è il numero di richieste che arrivano al sistema e maggiore sarà il throughput misurato.

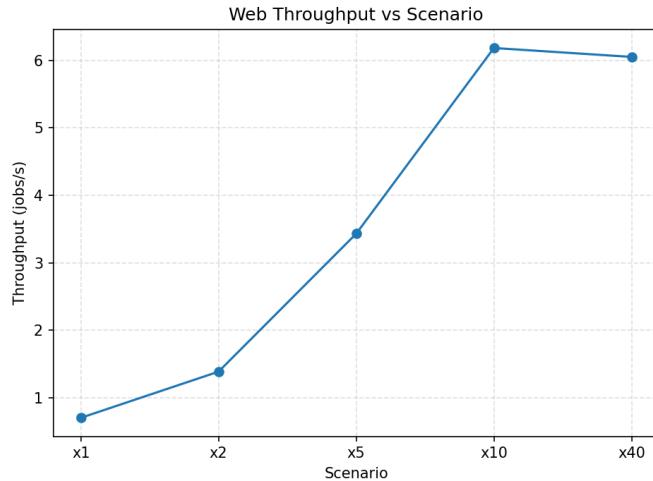


Figure 60: Throughput per il Web Server al variare del tasso di arrivo

Nella figura si nota anche come il valore del throughput tende leggermente a diminuire. Tale comportamento è dovuto al crescente utilizzo degli Spike Server, impiegati in misura maggiore per far fronte all'incremento del carico di richieste.

Nella prossima figura vengono mostrati i valori ottenuti relativi all'utilizzazione del centro al variare del tempo di interarrivo. I risultati ottenuti sono coerenti con il comportamento atteso del centro: all'aumentare del numero delle richieste il valore dell'utilizzazione del centro tenderà ad aumentare.

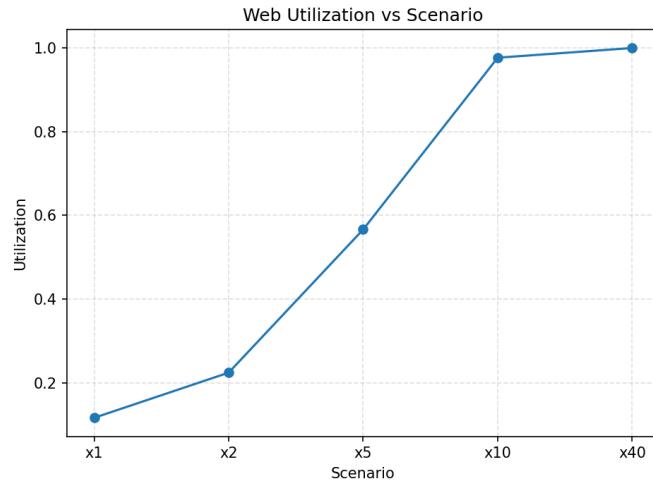


Figure 61: Utilizzazione per il Web Server al variare del tasso di arrivo

Scenario	Tempo Medio di Risposta	Throughput	Utilizzazione
x1	0.19093476182124253	0.7010022492324693	0.11349156542703687
x2	0.22719778380122654	1.3820514438204183	0.22284382078824513
x5	0.42510922565726444	3.4510409023132564	0.5500213338146339
x10	1.9372214389175135	6.174442497051954	0.9767906137231158
x40	3.1917549311183917	6.065995432855839	0.9966174453836792

Table 9: Tabella Riassuntiva delle metriche ottenute per il Web Server

I comportamenti osservati corrispondono ai comportamenti attesi e di conseguenza validano questa parte del sistema.

#### 9.8.4 Primo Spike Server Allocato

Di seguito sono mostrati i dati relativi al primo spike server allocato tramite il meccanismo di autoscaling al variare del tasso di interarrivo.

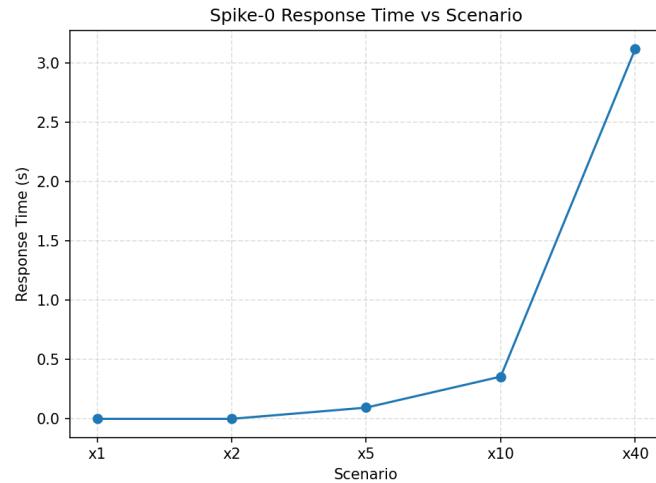


Figure 62: Tempo Medio di Risposta del primo Spike Server al variare del tasso di arrivo

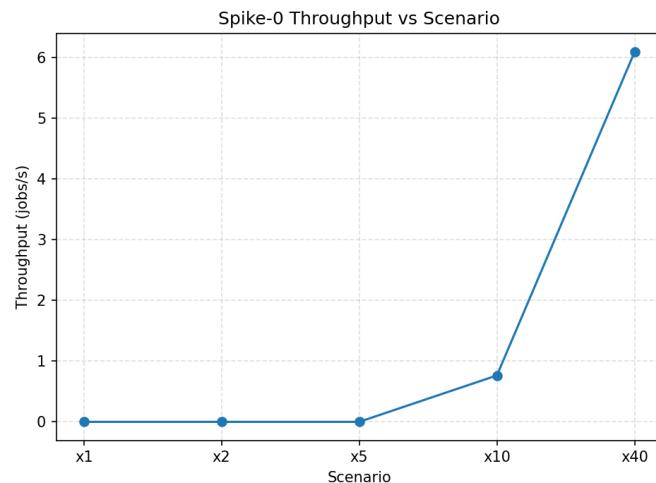


Figure 63: Throughput del primo Spike Server al variare del tasso di arrivo

In questo caso si nota come il sistema utilizzi in modo massimo lo Spike Server solo nell'ultimo scenario, ovvero quando si simula il carico di lavoro nel caso di un attacco.

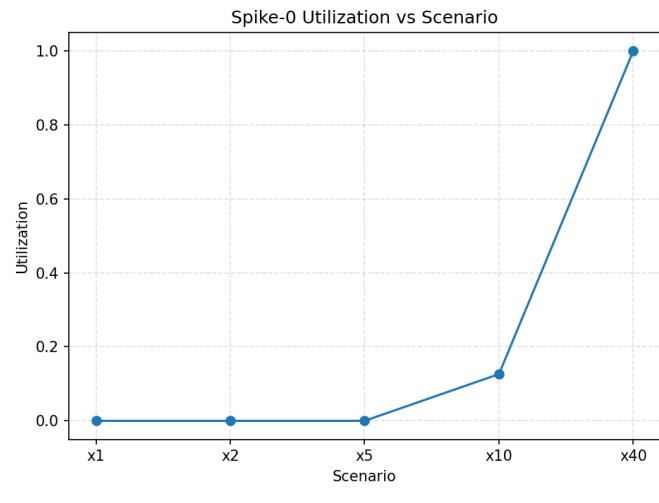


Figure 64: Utilizzazione del primo Spike Server al variare del tasso di arrivo

Scenario	Tempo Medio di Risposta	Throughput	Utilizzazione
<b>x1</b>	0	0	0
<b>x2</b>	0	0	0
<b>x5</b>	0.09445858757089809	0.00026589932792551335	$1.6092728929369204e - 05$
<b>x10</b>	0.3569222030407264	0.7688413133060487	0.12815511849542613
<b>x40</b>	3.1319766761387644	6.067049426241506	0.9989759821321368

Table 10: Tabella Riassuntiva delle metriche ottenute per il primo Spike Server allocato

Di conseguenza, i comportamenti osservati corrispondono ai comportamenti attesi e validano questa parte di sistema.

### 9.8.5 Numero di Spike Server allocati

Di seguito è riportato il grafico raffigurante il numero di Spike Server allocati al variare del tempo di interarrivo.

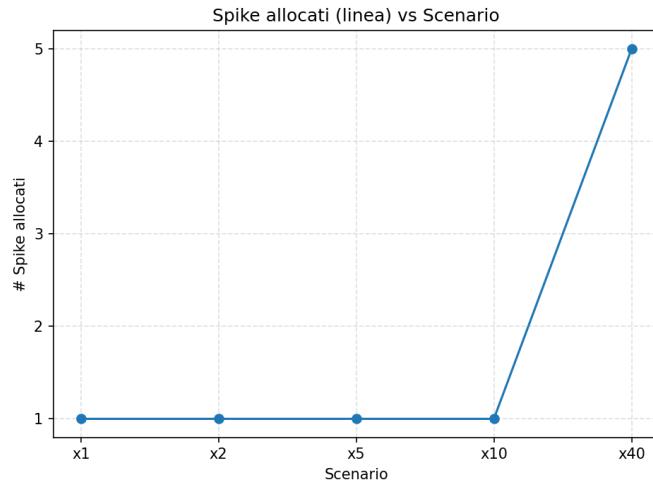


Figure 65: Numero di Spike Server al variare del tasso di arrivo

Il comportamento osservato corrisponde a quello atteso dal sistema, in particolare all'aumentare del numero di richieste da processare allora il numero di spike server allocati tenderà ad aumentare.

Scenario	Spike Server Allocati
x1	0
x2	0
x5	1
x10	1
x40	5

Table 11: Tabella Riassuntiva del numero di Spike Server allocati

In questo caso si nota l'impatto dell'utilizzo del centro di analisi aggiunto migliori di gran lunga il sistema, si nota infatti come il numero di Spike Server allocati cali drasticamente rispetto al modello di base.

## 9.9 Progettazione degli Esperimenti

### 9.9.1 Analisi del Transitorio

Per effettuare questo studio abbiamo eseguito 7 run della simulazione ad orizzonte finito con seed differenti.

Per poter valutare se il sistema raggiungesse un certo valore di convergenza abbiamo impostato la variabile **STOP\_CONDITION\_TRANSITORY = 100'000 s**, ovvero circa quasi un giorno in cui il sistema viene sottoposto ad un attacco, dove la raccolta delle metriche avviene con un intervallo di 100 secondi per la raccolta delle metriche di interesse.

Di seguito vengono mostrati i grafici ottenuti dell'analisi del transitorio ottenuti tramite la simulazione ad orizzonte finito.

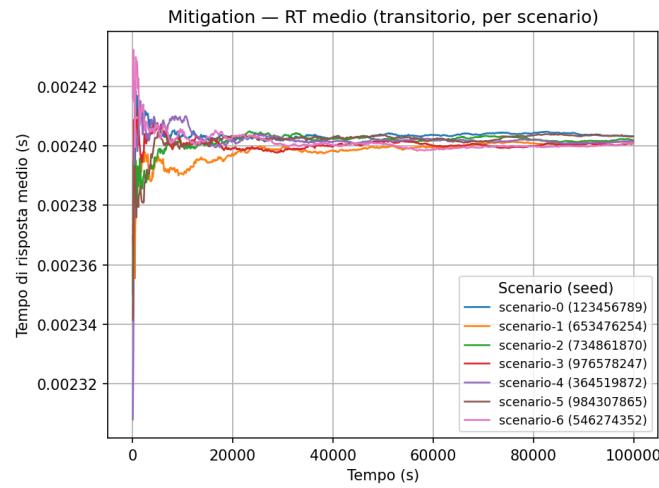


Figure 66: Tempo di Risposta Medio del Centro di Mitigazione

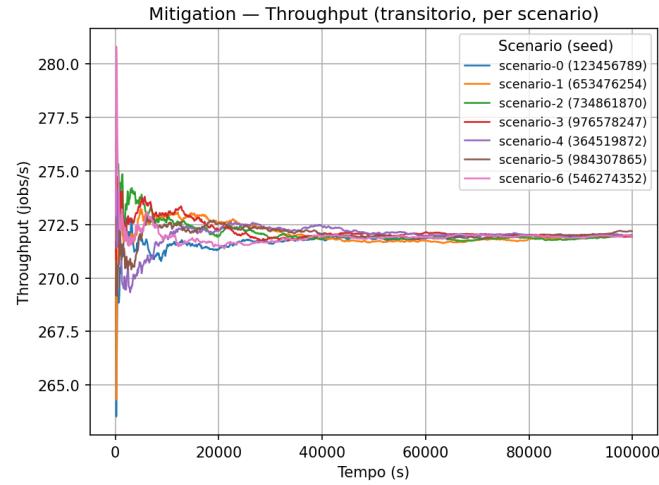


Figure 67: Throughput del Centro di Mitigazione

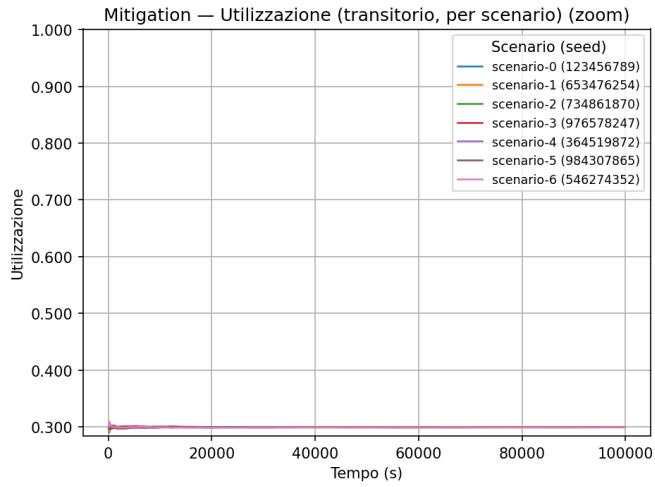


Figure 68: Utilizzazione del Centro di Mitigazione

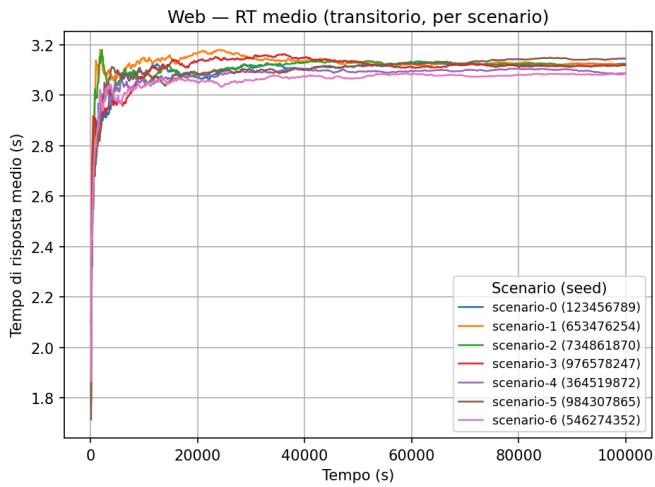


Figure 69: Tempo di Risposta Medio del Web Server

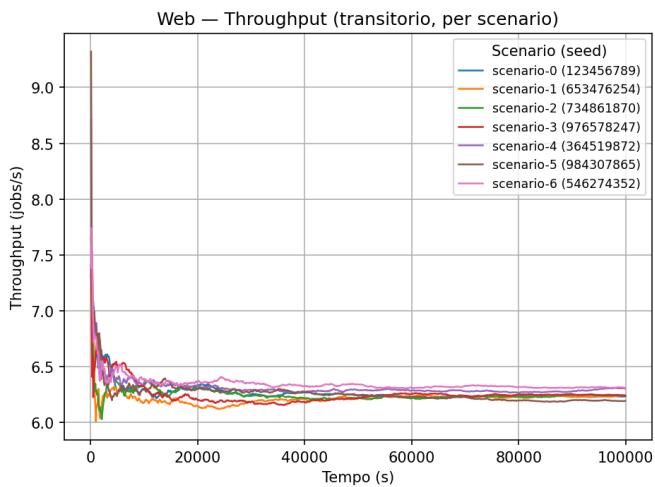


Figure 70: Throughput del Web Server

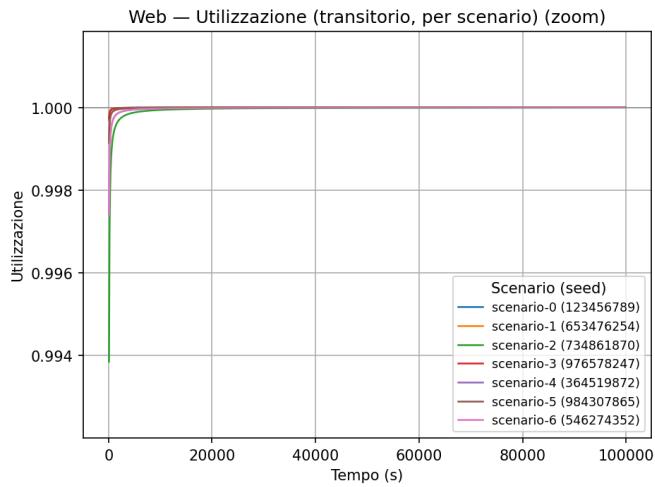


Figure 71: Utilizzazione del Web Server

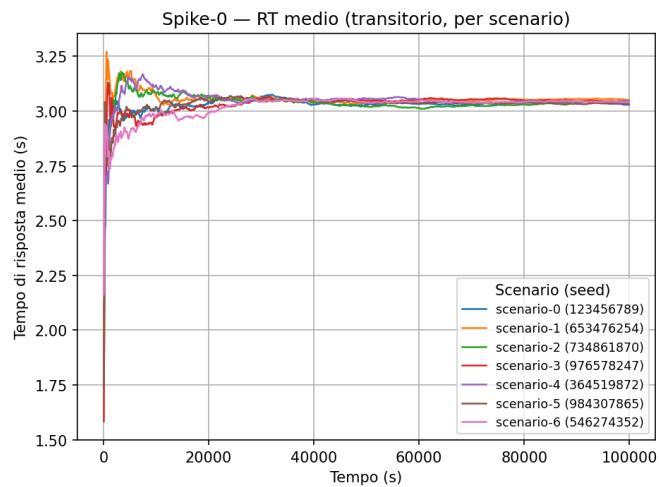


Figure 72: Tempo di Risposta Medio del primo Spike Server allocato

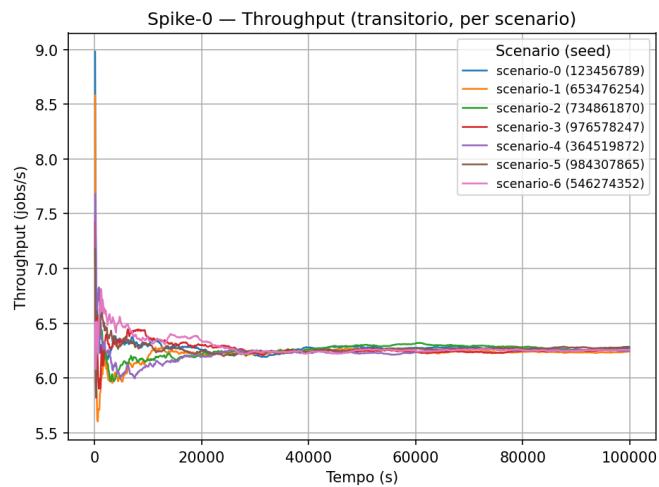


Figure 73: Throughput del primo Spike Server allocato

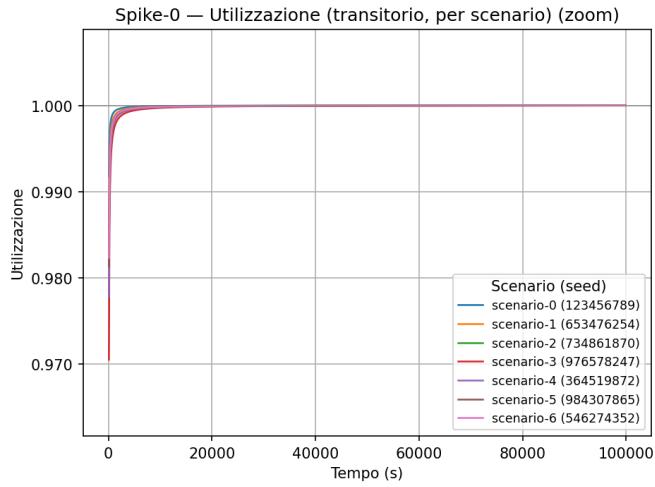


Figure 74: Utilizzazione del primo Spike Server allocato

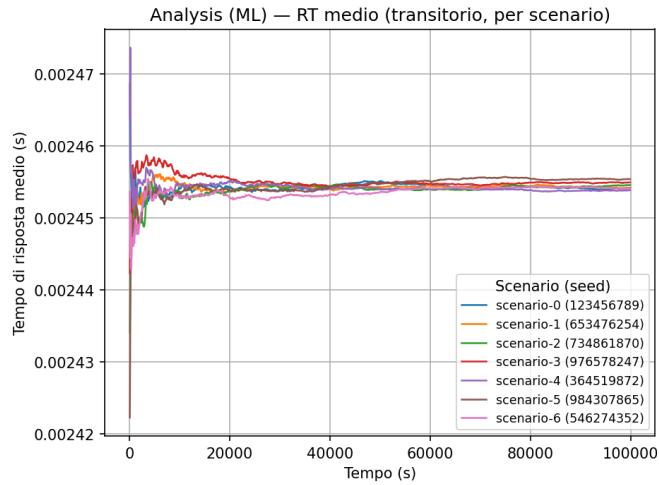


Figure 75: Tempo Medio di Risposta per il centro di Machine Learning Analysis

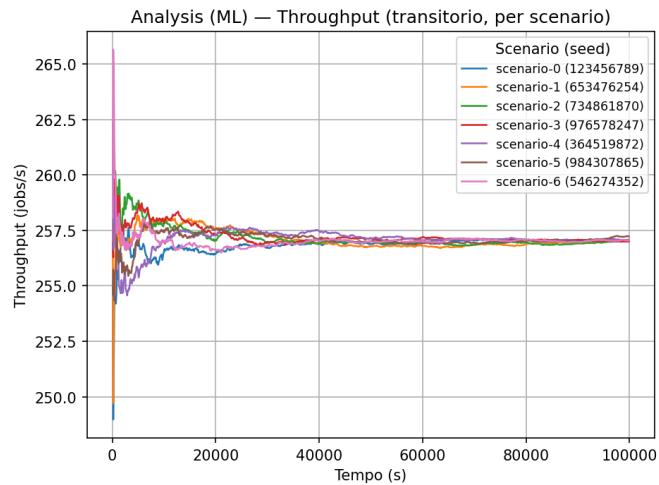


Figure 76: Throughput per il centro di Machine Learning Analysis

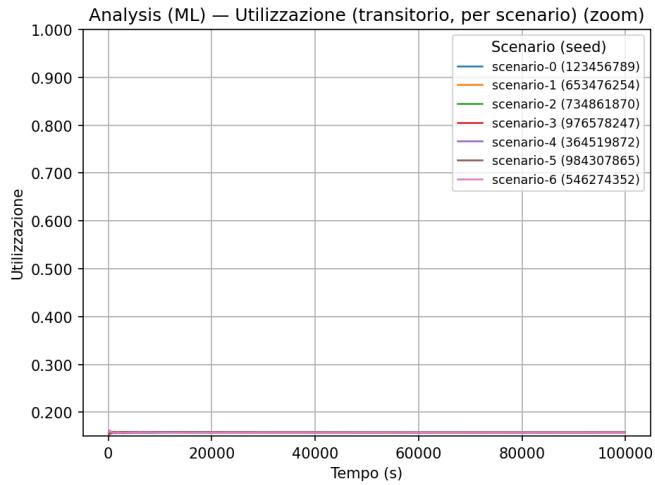


Figure 77: Utilizzazione per il centro di Machine Learning Analysis

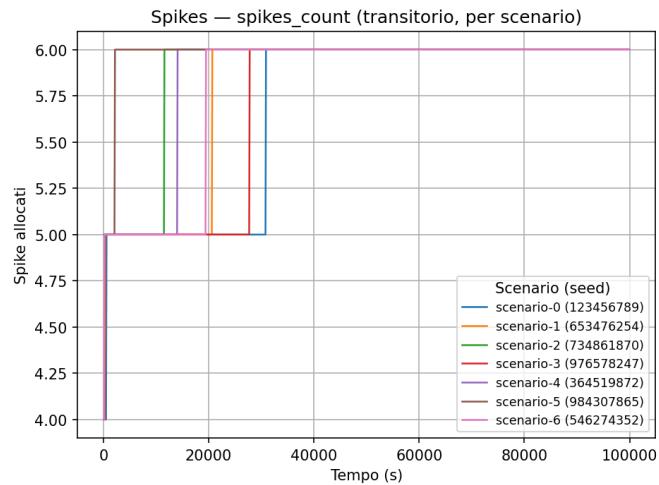


Figure 78: Numero di Spike Server allocati

Nella figura 66 viene mostrato l'andamento del tempo medio di risposta del Centro di Mitigazione e si nota come questa parte del sistema raggiunga uno stato stazionario, a discapito una differenza tre le varie simulazione di un valore inferiore ad un valore di 0.00005.

Nella figura 67 viene mostrato l'andamento del throughput del centro di mitigazione; si nota come anche in questo caso questa parte di sistema raggiunge uno stato stazionario.

Nella figura 68 si nota come il centro analizzato raggiunge fin da subito un comportamento stazionario.

Nelle figure 69, 70 e 71 si nota come rispettivamente il tempo medio di risposta, il throughput e l'utilizzazione del web server assumino un comportamento stazionario.

Nelle figure 72, 73 e 74 si hanno considerazioni analoghe a quelle del Web Server.

Nelle figure 75, 76 e 77 si può notare come anche per questo centro la stazionarietà è raggiunta.

In figura 78 si nota come vengano allocati 6 Spike Server anche con differenti seeds utilizzati.

### 9.9.2 Simulazione ad Orizzonte Finito

Come nel caso del modello di base, la simulazione a orizzonte finito è stata sviluppata seguendo l'approccio delle replicazioni indipendenti utilizzando gli stessi parametri.

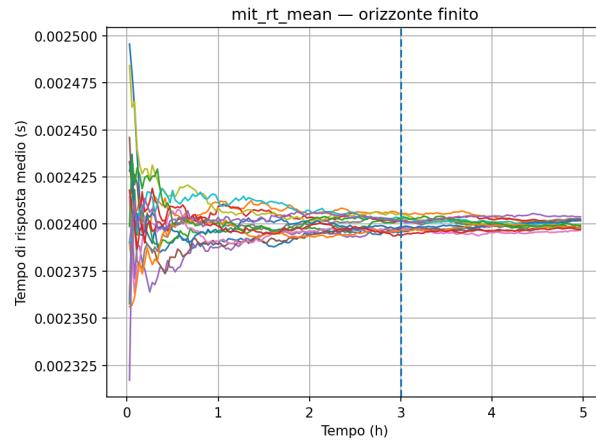


Figure 79: Tempo di risposta del Mitigation Center

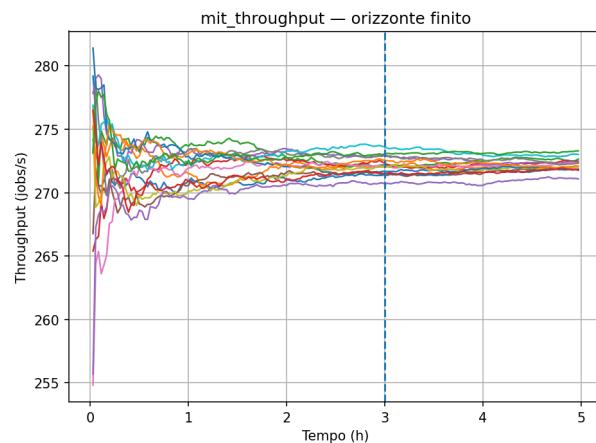


Figure 80: Throughput del Mitigation Center

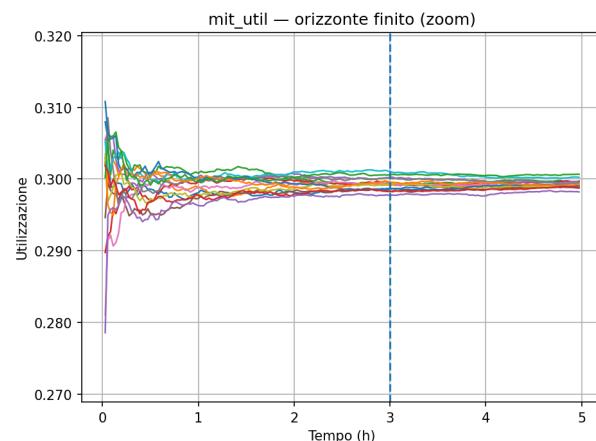


Figure 81: Utilizzazione del Mitigation Center

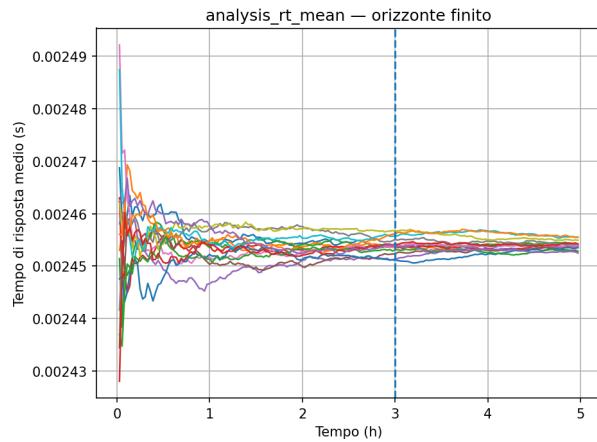


Figure 82: Tempo Medio di Risposta del Machine Learning Analysis

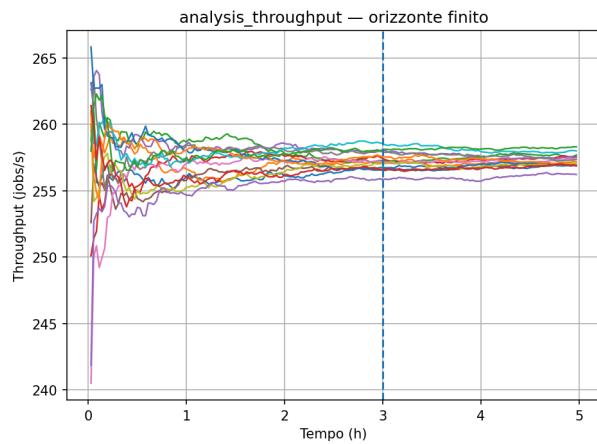


Figure 83: Throughput del Machine Learning Analysis

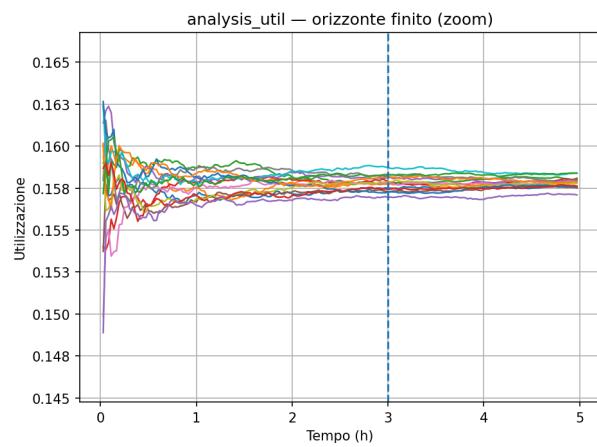


Figure 84: Utilizzazione del Machine Learning Analysis

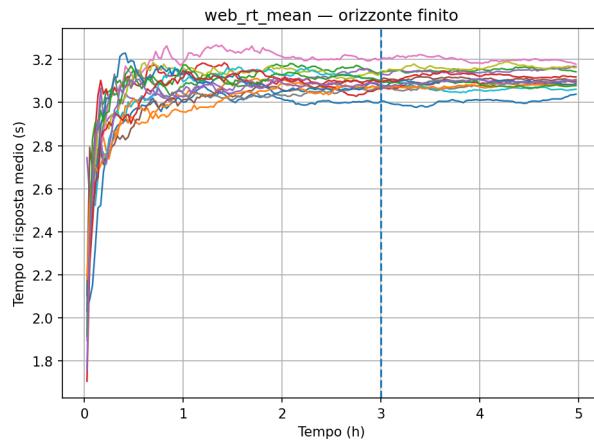


Figure 85: Tempo Medio di Risposta del Web Server

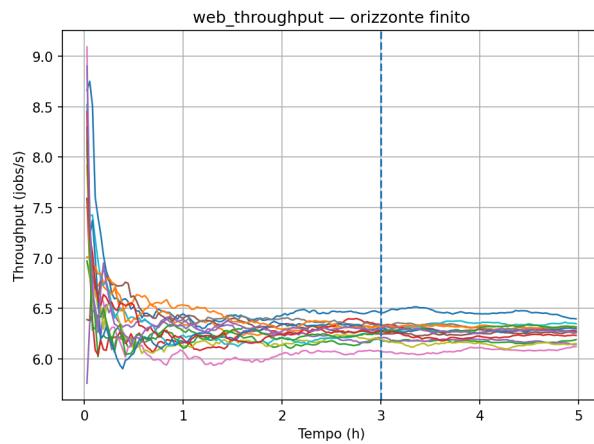


Figure 86: Throughput del Web Server

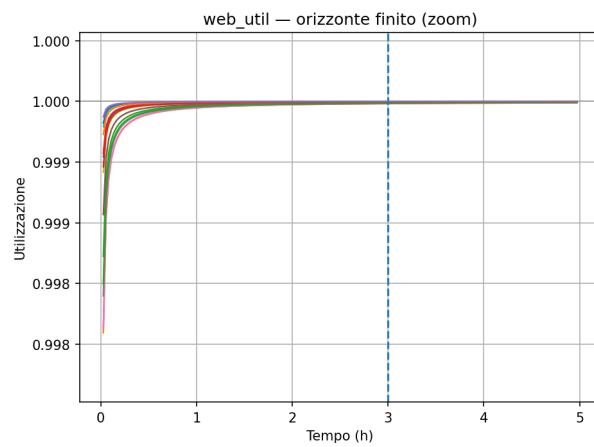


Figure 87: Utilizzazione del Web Server

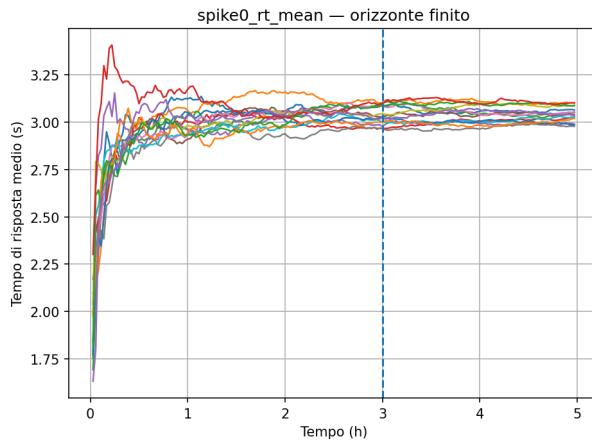


Figure 88: Tempo Medio di Risposta del primo Spike Server allocato

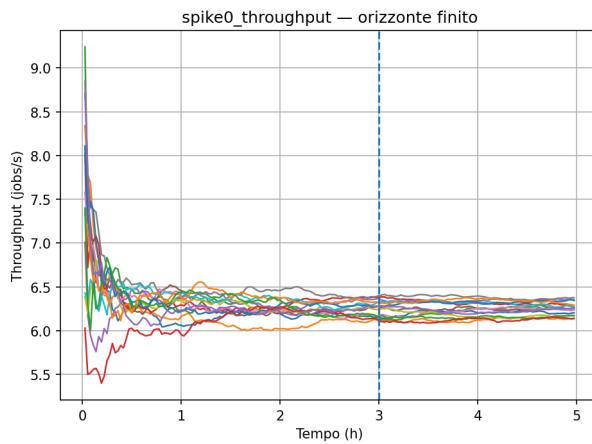


Figure 89: Throughput del primo Spike Server allocato

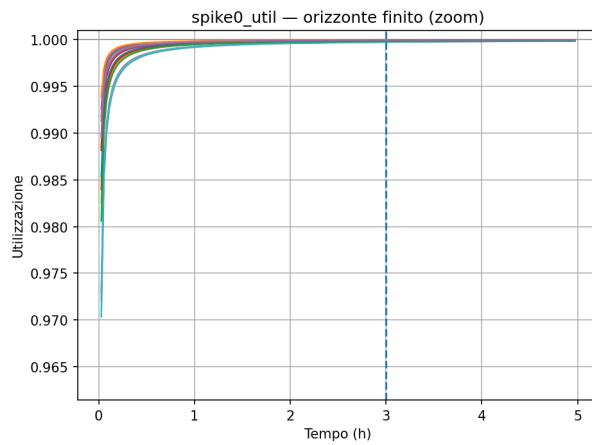


Figure 90: Utilizzazione del primo Spike Server allocato

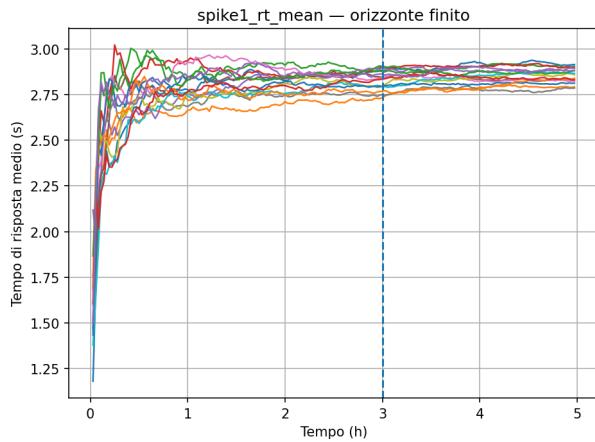


Figure 91: Tempo Medio di Risposta per il secondo Spike Server allocato

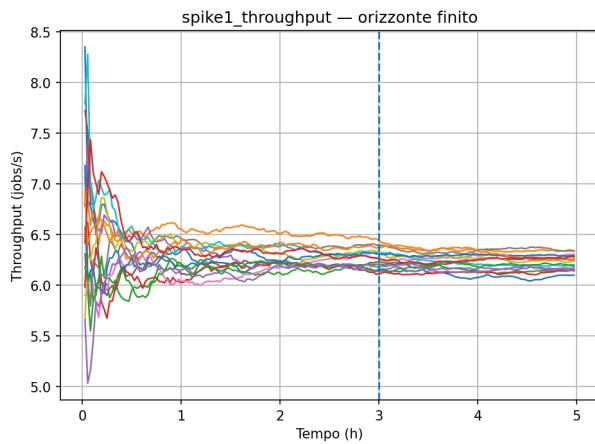


Figure 92: Throughput per il secondo Spike Server allocato

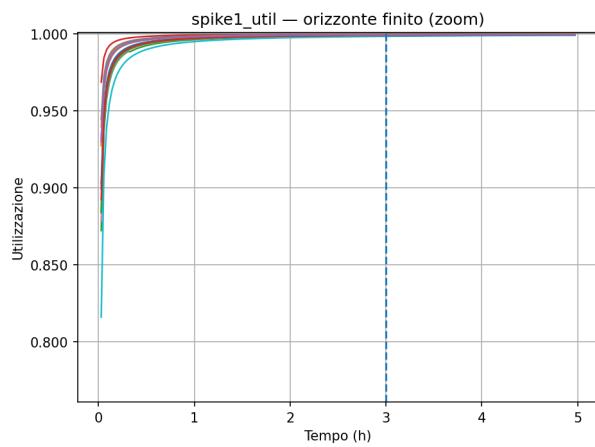


Figure 93: Utilizzazione per il secondo Spike Server allocato

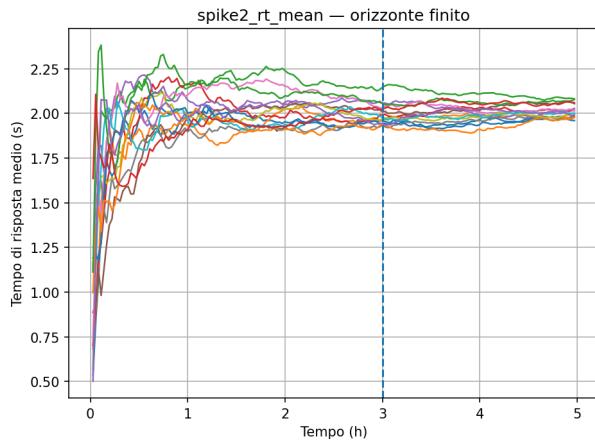


Figure 94: Tempo Medio di Risposta per il terzo Spike Server allocato

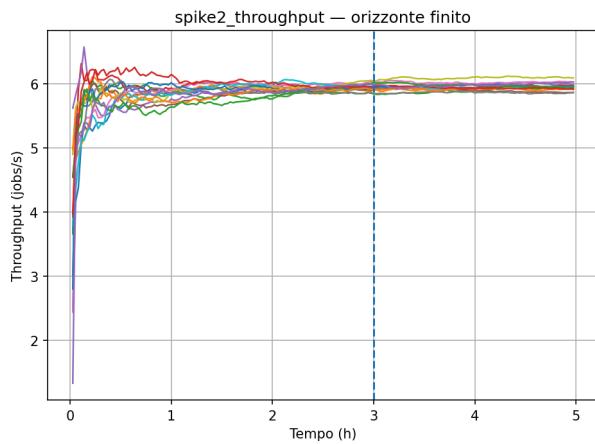


Figure 95: Throughput per il terzo Spike Server allocato

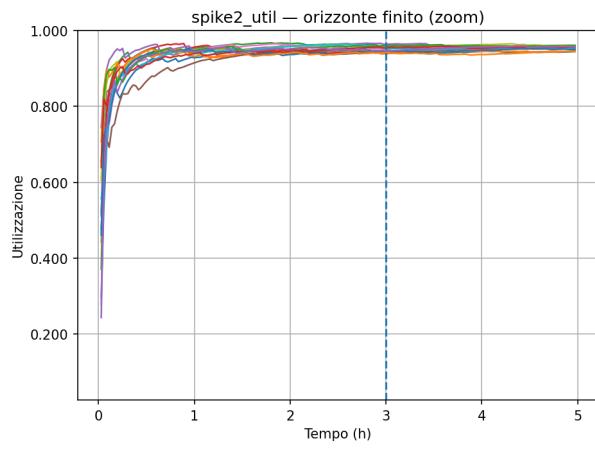


Figure 96: Utilizzazione per il terzo Spike Server allocato

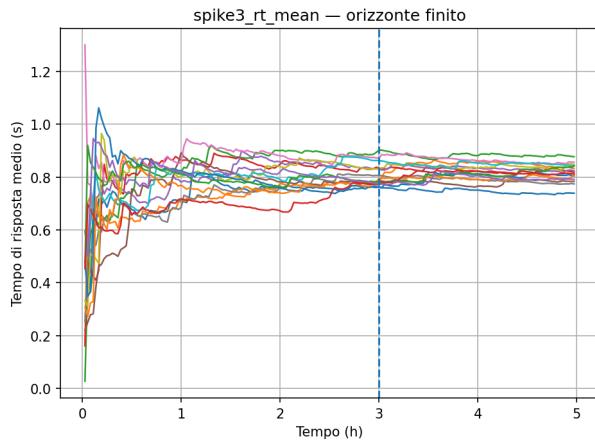


Figure 97: Tempo Medio di Risposta per il quarto Spike Server allocato

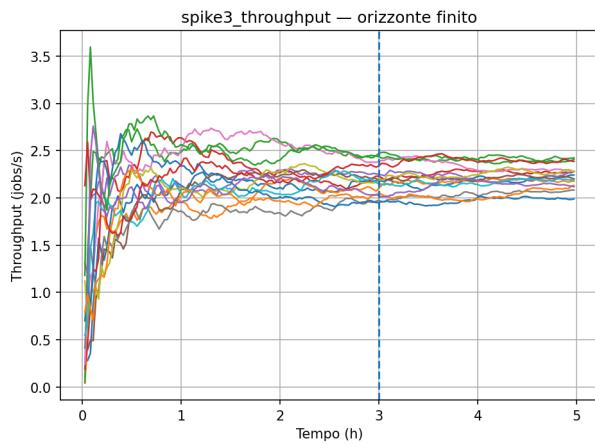


Figure 98: Throughput per il quarto Spike Server allocato

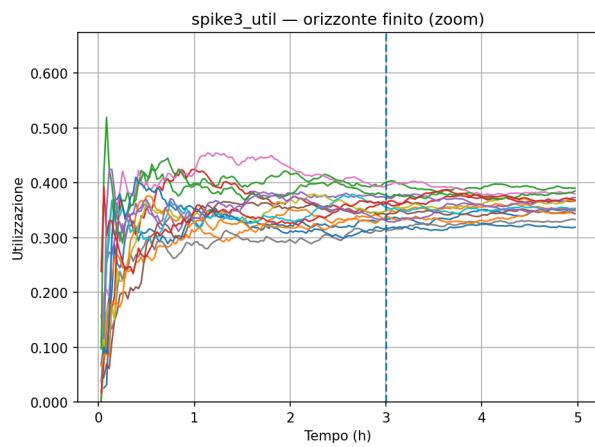


Figure 99: Utilizzazione per il quarto Spike Server allocato

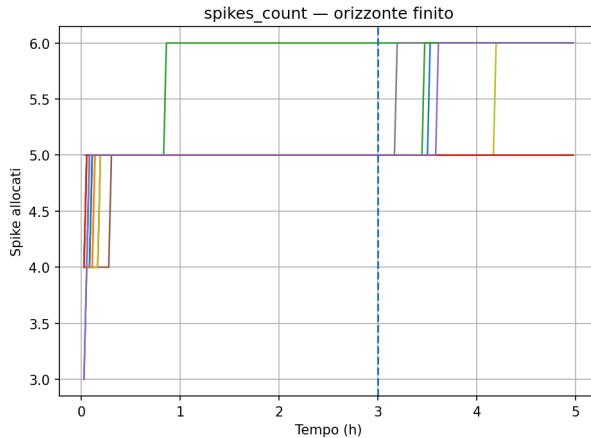


Figure 100: Numero di Spike Allocati

Per completezza sono stati riportati i grafici dell’andamento del tempo medio di risposta, throughput e utilizzazione dei primi quattro Spike Server allocati. Si nota come in questo caso solo i primi due Spike Server tendano ad avere un comportamento molto simile a quello del Web Server; il terzo Spike allocato ha valore del throughput leggermente più basso: questo significa che i job da processsare sono molti di meno rispetto a quelli che si ottenevano con il modello di base.

Tale comportamento risulta ancora più evidente analizzando i risultati relativi al quarto Spike allocato. Questo conferma come il meccanismo di instradamento privilegi inizialmente il Web Server e i primi Spike. Ciò dimostra che l’introduzione del centro aggiuntivo nel modello migliorativo produce effettivamente i benefici attesi.

In figura 100 si nota come il numero di spike server utilizzati in media sia 5. Considerando questo valore, l’ammontare del dispendio economico ammonta a circa 4.75 euro per una VM **r6a.xlarge** con costo orario pari a 0.19, che sommati al costo mensile si arriva ad un ammontare di 141.55 euro. Oltre a questo valore va sommato il valore del costo del centro di analisi aggiuntivo in cui deve essere sempre attivo per poter prevenire gli attacchi e deve essere sempre attivo. In definitiva, l’ammontare economico arriva a 278.35 euro.

### 9.9.3 Simulazione ad Orizzonte Infinito

Come per il modello di base, la simulazione ad orizzonte infinito è stata svolto mantenendo la configurazione utilizzata finora, quindi con i centri utilizzati che partono vuoti e con i valori di interarrivo relativi all'attacco.

Rispetto al modello base, il campione è stato diviso in  $K = 128$  batch di dimensione  $B = 1500$  jobs.

Di seguito vengono mostrati i valori del tempo di risposta medio, throughput e utilizzazione per il Centro di Mitigazione, Web Server, per il primo Spike Server allocato e il centro aggiunto con il modello migliorativo, ossia il Machine Learning Analysis Center.

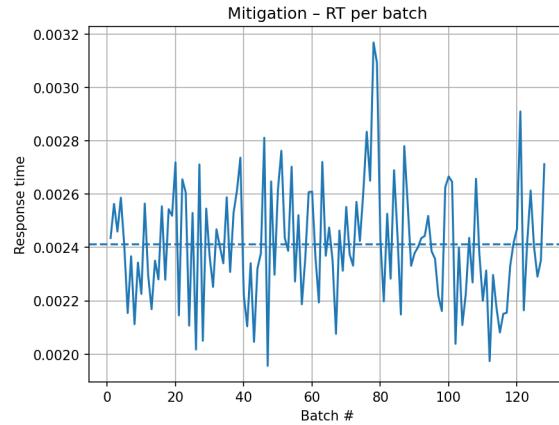


Figure 101: Tempo di Risposta Medio per il Mitigation Center

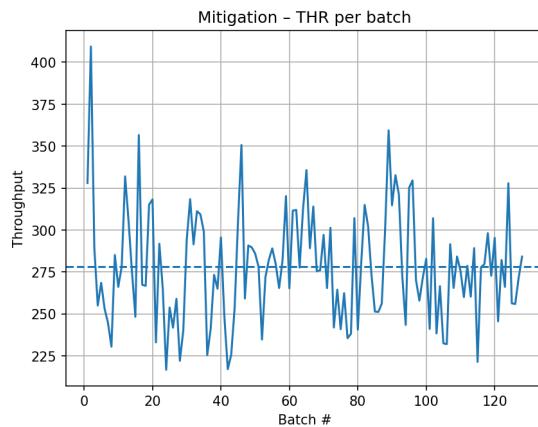


Figure 102: Throughput per il Mitigation Center

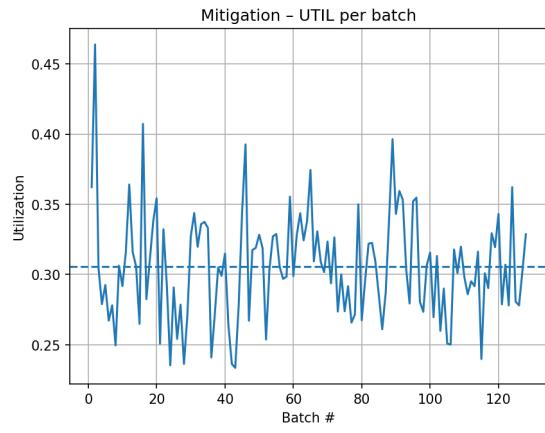


Figure 103: Utilizzazione per il Mitigation Center

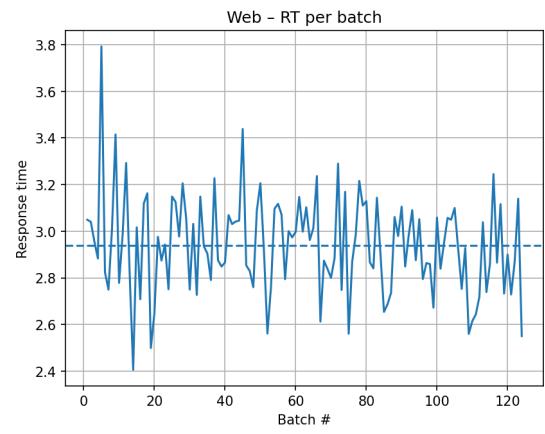


Figure 104: Tempo di Risposta Medio per il Web Server

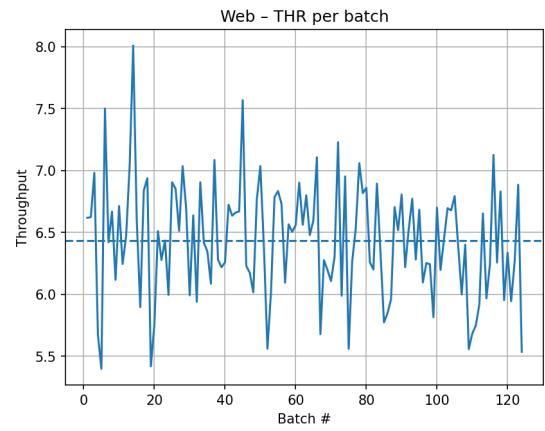


Figure 105: Throughput per il Web Server

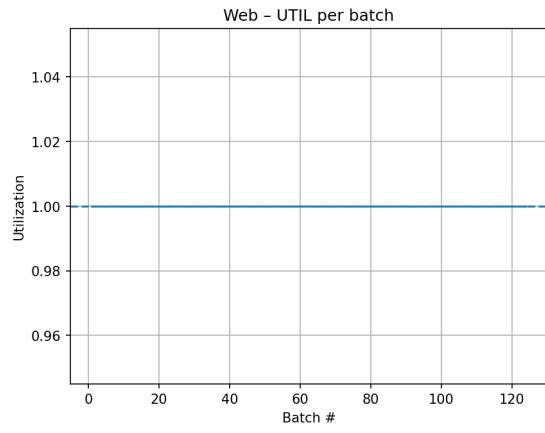


Figure 106: Utilizzazioni per il Web Server

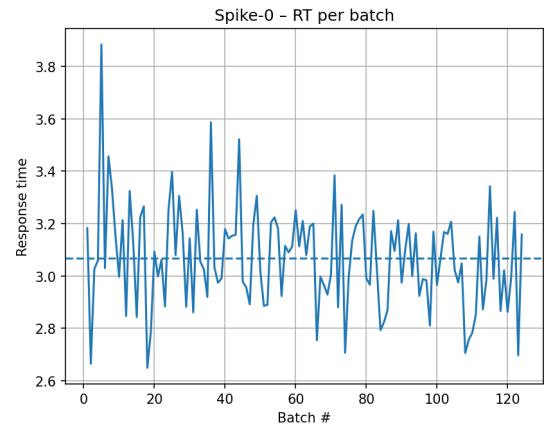


Figure 107: Tempo di Risposta Medio per il primo Spike Server allocato

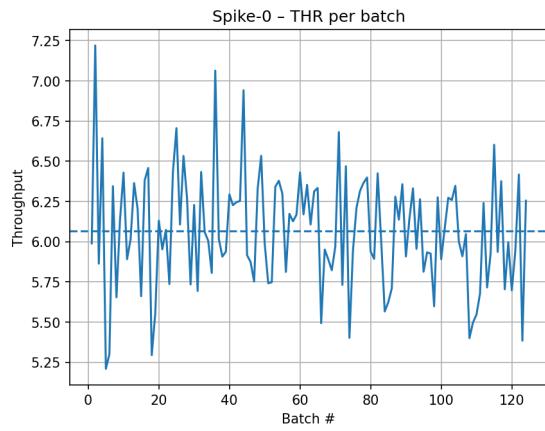


Figure 108: Throughput per il primo Spike Server allocato

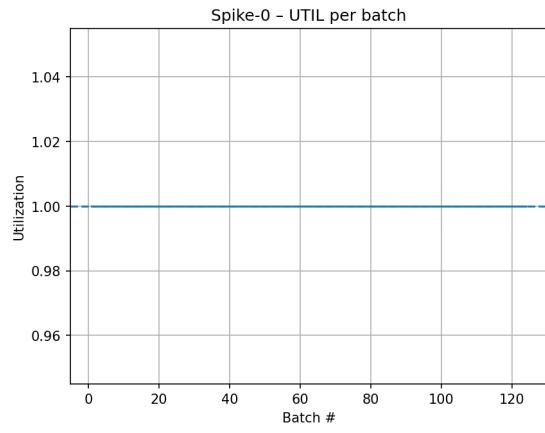


Figure 109: Utilizzazione per il primo Spike Server allocato

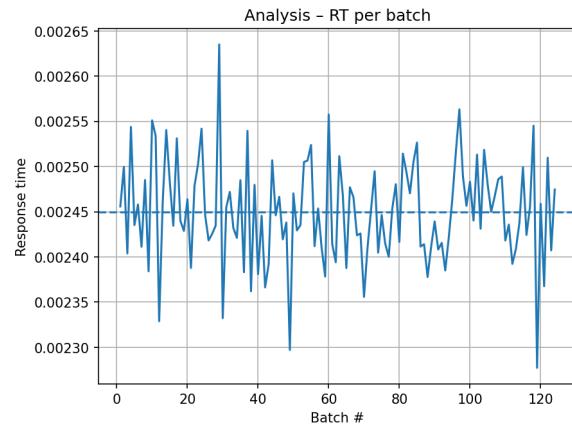


Figure 110: Tempo di Risposta per il Machine Learning Analysis

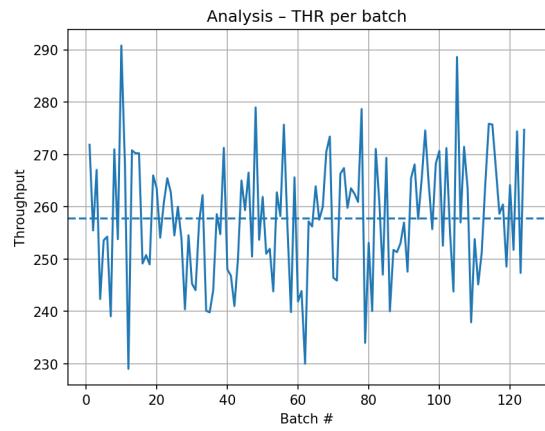


Figure 111: Throughput per il Machine Learning Analysis

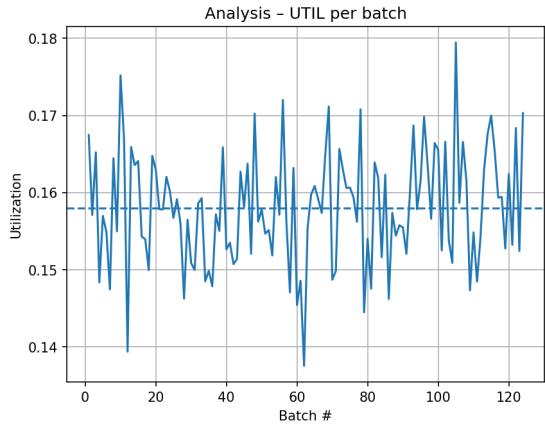


Figure 112: Utilizzazione per il Machine Learning Analysis

In generale, analizzando i risultati ottenuti nella simulazione ad orizzonte infinito, si nota come i risultati siano molto simili a quelli ottenuti con la simulazione ad orizzonte finito.

Di seguito sono rappresentati i risultati ottenuti in forma tabellare.

Centro	Tempo Medio di Risposta	Throughput	Utilizzazione
Mitigation Center	$0.002416 \pm 0.000045$	$279.072660 \pm 8.779909$	$0.306669 \pm 0.009632$
Web Server	$3.019905 \pm 0.176420$	$6.432457 \pm 0.260237$	$0.999995 \pm 0.000010$
Spike Server - 0	$3.015479 \pm 0.154447$	$6.262225 \pm 0.270538$	$0.999999 \pm 0.000000$
Machine Learning Analysis	$0.002451 \pm 0.000016$	$255.733333 \pm 2.971603$	$0.156675 \pm 0.001842$

Table 12: Tabella Riassuntiva delle metriche ottenute con la simulazione ad orizzonte infinito

#### 9.9.4 Autocorrelazione

Di seguito si mostrano i valori dell'autocorrelazione per il modello migliorativo.

```
1  web_rt: rho_1=0.089  (|rho1|<0.177) → PASS
2  web_util: rho_1=0.000  (|rho1|<0.177) → PASS
3  web_thr: rho_1=0.062  (|rho1|<0.177) → PASS
4  spike0_rt: rho_1=-0.052  (|rho1|<0.177) → PASS
5  spike0_util: rho_1=0.000  (|rho1|<0.177) → PASS
6  spike0_thr: rho_1=0.081  (|rho1|<0.177) → PASS
7  spike1_rt: rho_1=0.150  (|rho1|<0.177) → PASS
8  spike1_util: rho_1=-0.008  (|rho1|<0.177) → PASS
9  spike1_thr: rho_1=0.175  (|rho1|<0.177) → PASS
10 spike2_rt: rho_1=0.109  (|rho1|<0.177) → PASS
11 spike2_util: rho_1=0.041  (|rho1|<0.177) → PASS
12 spike2_thr: rho_1=0.010  (|rho1|<0.177) → PASS
13 spike3_rt: rho_1=0.053  (|rho1|<0.177) → PASS
14 spike3_util: rho_1=0.047  (|rho1|<0.177) → PASS
15 spike3_thr: rho_1=0.010  (|rho1|<0.177) → PASS
16 spike4_rt: rho_1=0.021  (|rho1|<0.177) → PASS
17 spike4_util: rho_1=-0.141  (|rho1|<0.177) → PASS
18 spike4_thr: rho_1=-0.005  (|rho1|<0.177) → PASS
19 spike5 - SKIPPED
20 mit_rt: rho_1=0.106  (|rho1|<0.177) → PASS
21 mit_util: rho_1=0.121  (|rho1|<0.177) → PASS
22 mit_thr: rho_1=0.134  (|rho1|<0.177) → PASS
23 ana_rt: rho_1=-0.150  (|rho1|<0.177) → PASS
24 ana_util: rho_1=0.000  (|rho1|<0.177) → PASS
25 ana_thr: rho_1=0.170  (|rho1|<0.177) → PASS
```

Figure 113: Valori dell'autocorrelazione

## 10 Conclusioni

Si nota come l'inserimento del centro aggiuntivo porti benefici in termini di Spike Server allocati e utilizzati, di conseguenza anche in termini di risorse on-demand utilizzate per il processamento delle richieste illecite.

L'integrazione di un centro di questo tipo può richiedere una macchina dedicata esclusivamente all'esecuzione degli algoritmi di Machine Learning. In assenza di hardware dedicato, si può ricorrere a una VM su AWS, come nel caso di studio, con i conseguenti costi operativi per mantenerla attiva.

Considerando macchine più costose e con migliori caratteristiche si possono avere guadagni economici anche in caso di attacchi.

Nome istanza	Tariffa oraria (USD)	Costo orario €	vCPUs	Costo mensile € - Modello Base	Costo mensile € - Modello Migiorativo
r6a.xlarge	0,2268	0.19	4	186.2	278.35
i7ie.3xlarge	1,5594	1,35	12	2'295	1'977.75
i7ie.6xlarge	3,1188	2,70	24	4'590	3'955,5
x2iedn.8xlarge	6,669	5,77	32	9'809	8'453,05
p3.8xlarge	12,24	10,59	32	18'003	15'514.34

Nella tabella precedente vengono mostrati l'impatto economico dell'aggiunta del centro di analisi aggiuntivo.

Tuttavia, è essenziale considerare un aspetto dinamico cruciale. I vettori di attacco DDoS non sono statici, ma sono in continua e rapida evoluzione. Di conseguenza, l'efficacia dell'algoritmo di Machine Learning non può essere data per scontata a lungo termine. Per mantenere una capacità di mitigazione costante e affidabile, l'algoritmo deve essere regolarmente aggiornato. Tale aggiornamento comporta l'avvio di una nuova fase di addestramento, che integra i dati relativi alle nuove tipologie di attacco. Questa necessità di un addestramento periodico si traduce in ulteriori costi operativi e in un'ulteriore richiesta di risorse computazionali, che devono essere attentamente valutate nella scelta di implementare una soluzione di questo tipo.

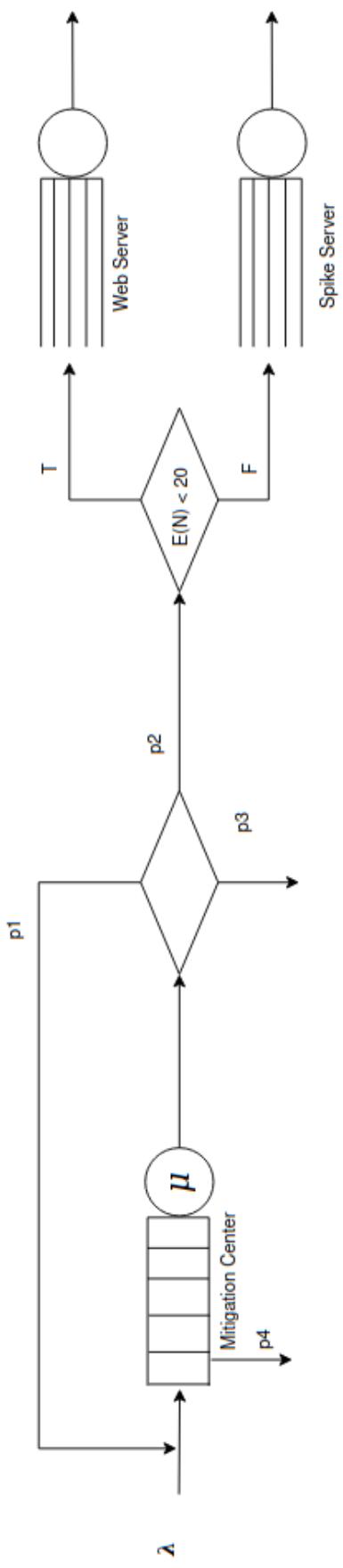


Figure 114: Modello Concettuale del Modello di Base

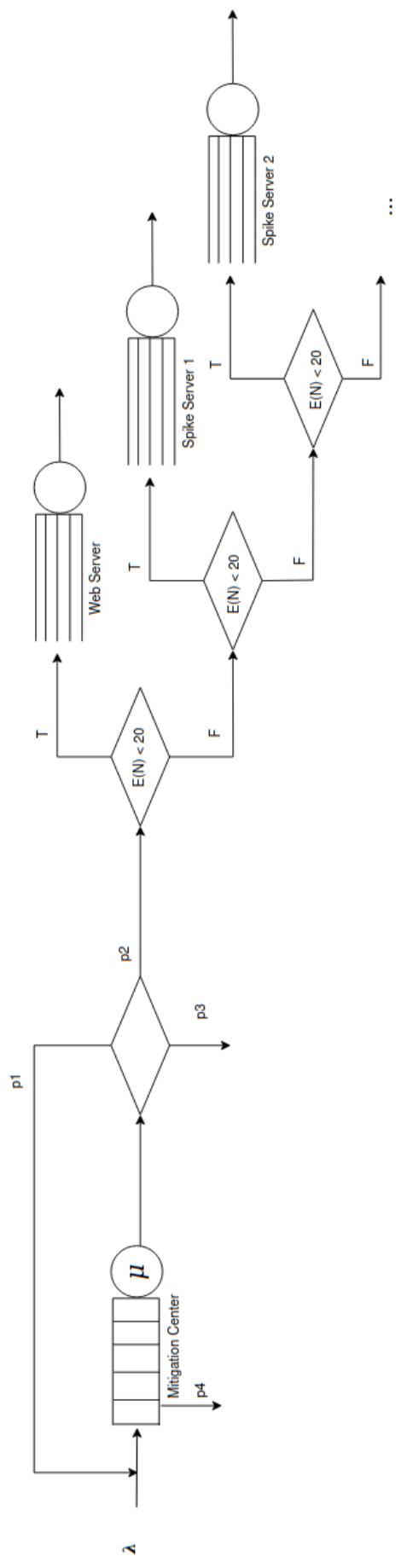


Figure 115: Modello Concettuale del Modello di Base

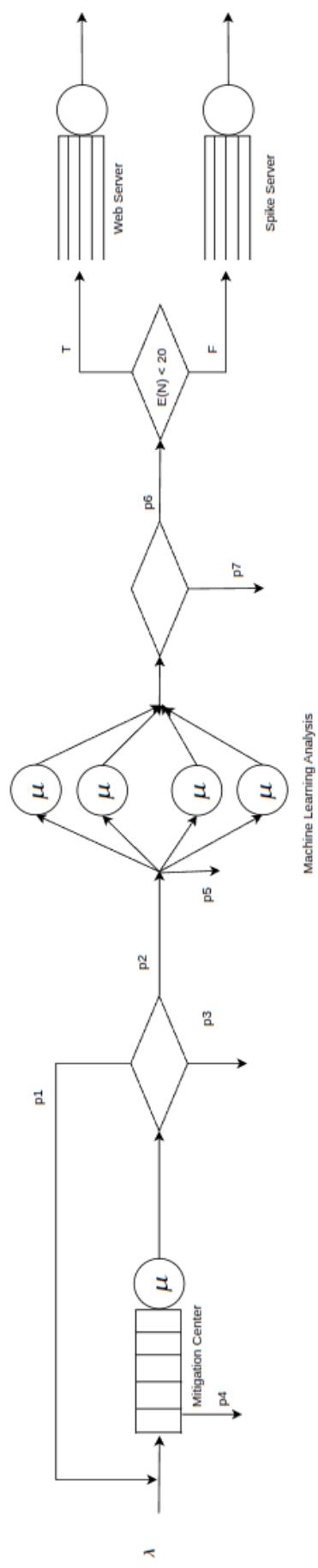


Figure 116: Modello Concettuale del Modello Migliorativo

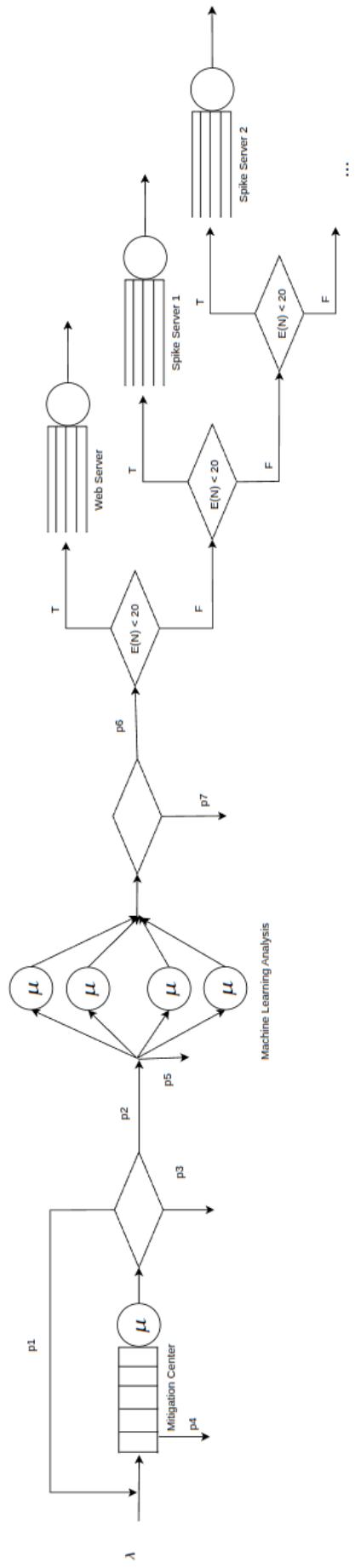


Figure 117: Modello Concettuale del Modello Migliorativo

## References

- [1] Amazon AWS. *Best Practices for DDoS Resiliency*. <https://docs.aws.amazon.com/pdfs/whitepapers/latest/aws-best-practices-ddos-resiliency/aws-best-practices-ddos-resiliency.pdf>.
- [2] Cisco. *DDoS Mitigation*. <https://blogs.cisco.com/sp/ddos-mitigation-for-modern-peering>.
- [3] EC2. *AWS - Amazon Web Service*.  
URL: <https://aws.amazon.com/it/ec2/pricing/on-demand/>.
- [4] Angelo Furfaro et al. *A Simulation Model for the Analysis of DDoS Amplification Attacks*. Available at: <https://ieeexplore.ieee.org/document/7576555>. DIMES - University of Calabria, Open Knowledge Technologies S.r.l. 2016.
- [5] Brooks Automation Barry L Nelson Jerry Banks John S Carson II and David M Nicol. *Discrete-event system simulation fourth edition*.
- [6] kaggle. *NSL-KDD*. <https://www.kaggle.com/datasets/hassan06/nslkdd>.
- [7] netScout. *Performance Evaluation of Intrusion Detection Systems*. <https://www.netscout.com/threatreport/global-highlights/>. 2024.
- [8] ResearchGate. *Performance Evaluation of Intrusion Detection Systems*. [https://www.researchgate.net/publication/335954299\\_Exploring\\_New\\_Opportunities\\_to\\_Defeat\\_Low-Rate\\_DDoS\\_Attack\\_in\\_Container-Based\\_Cloud\\_Environment](https://www.researchgate.net/publication/335954299_Exploring_New_Opportunities_to_Defeat_Low-Rate_DDoS_Attack_in_Container-Based_Cloud_Environment). 2019.
- [9] Auto Scaling. *AWS - Amazon Web Services*.  
URL: <https://aws.amazon.com/it/autoscaling/>.
- [10] Gianfranco Serazzi. *Performance Engineering: Learning Through Applications Using JMT*. Paragrafo 6.2. Springer, 2020.
- [11] Cisco Systems. *Cisco ASA: default queue-limit 1024 packets*. Default queue limit is 1024 packets, configurable via `queue-limit` <https://www.cisco.com/c/en/us/support/docs/switches/catalyst-9500-series-switches/218444-understand-buffer-allocation-on-ca.html>.
- [12] Cisco Systems. *Intrusion Policy Performance — Cisco Secure Firewall Management Center 7.2*. <https://www.cisco.com/c/en/us/td/docs/security/secure-firewall-management-center/device-config/720/management-center-device-config-72/intrusion-performance.html>.