

UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

MACROAREA DI INGEGNERIA



CORSO DI LAUREA MAGISTRALE IN INGEGNERIA
INFORMATICA

Relazione Progetto
Performance Modeling of Computer
System and Networks

Aeroporto di Ciampino

Stefano Belli
Dennis Mariani

0350116
0365494

Anno Accademico 2024/2025

| | |
|---|-----------|
| 1 - Descrizione del sistema | 1 |
| 1.1 - Introduzione | 1 |
| 1.2 - Problematiche | 2 |
| 2 - Obiettivi dello studio | 2 |
| 2.1 - Requisiti qualità del servizio e sicurezza | 2 |
| 3 - Modello concettuale | 3 |
| 3.1 - Diagramma | 3 |
| 3.2 - Descrizione | 3 |
| 3.2.1 - Banchi Accettazione e Varchi Elettronici (Centro 1 e 2) | 3 |
| 3.2.2 - Controlli di Sicurezza (Centro 3, 4 e 5) | 4 |
| 3.3 - Sintesi | 5 |
| 3.4 - Stato | 6 |
| 3.5 - Eventi | 6 |
| 4 - Modello di specifica | 7 |
| 4.1 - Processo di arrivo | 7 |
| 4.2 - Probabilità di routing | 9 |
| 4.3 - Matrice di routing | 10 |
| 4.4 - Equazioni di traffico | 10 |
| 4.5 - Modellazione centri | 11 |
| Centro 1: Banchi accettazione | 11 |
| Centro 2: Varchi Elettronici (E-Gates) | 12 |
| Centro 3: Controlli a Raggi X | 13 |
| Centro 4: Trace Detection | 14 |
| Centro 5: Recupero oggetti utente | 14 |
| 5 - Modello Computazionale | 15 |
| 5.1 - Introduzione alla struttura generale del codice sorgente | 15 |
| 5.2 - Eventi, event queue e simulation clock | 16 |
| 5.3 - Jobs che attraversano la rete | 17 |
| 5.4 - Centri e routing | 18 |
| 5.5 - Costruzione del simulation model e l'event handler | 23 |
| 5.6 - Random number generation | 26 |
| 5.7 - Random variate generation | 27 |
| 6 - Verifica | 28 |
| 6.1 - Il problema dell'utilizzazione maggiore di 1 | 28 |
| 6.2 - Verifica del centro check-in M/M/8 | 29 |
| 6.3 - Verifica del centro varchi elettronici M/M/4 | 30 |
| 6.4 - Verifica del centro x-rays M/M/6 | 31 |
| 6.5 - Verifica del centro trace detection M/M/1 | 33 |
| 6.6 - Verifica del centro recupero M/M/ ∞ | 34 |
| 6.7 - Controlli di consistenza | 34 |
| 7 - Validazione | 35 |
| 7.1 - Verifica delle Probabilità di Instradamento | 35 |
| Figura 7a | 35 |
| 7.2 - Verifica dei Volumi di Traffico | 35 |

| | |
|--|-----------|
| 7.3 - Verifica dei Parametri di Servizio | 36 |
| Tabella 7b | 36 |
| 7.4 - Valutazione Macroscopica (Legge di Little) | 37 |
| Figura 7c | 37 |
| 7.5 - Verifiche di Consistenza | 38 |
| 8 - Design degli esperimenti | 38 |
| 8.1 - Analisi transitorio | 39 |
| 8.1.1 - Analisi con carico effettivo (med) | 39 |
| 8.1.2 - Determinazione del warm-up a carico ridotto | 42 |
| 8.2 - Simulazione ad orizzonte finito | 44 |
| 8.2.1 - Analisi del collo di bottiglia | 44 |
| 8.2.2 - Risultati della simulazione operativa | 46 |
| 8.3 - Simulazione ad orizzonte infinito | 49 |
| 8.4 - Conclusioni | 56 |
| 9 - Modello migliorativo | 56 |
| 9.1 - Modello concettuale | 57 |
| 9.1.1 - Diagramma | 57 |
| 9.1.2 - Descrizione | 57 |
| 9.1.3 - Sintesi | 58 |
| 9.1.4 - Stato | 58 |
| 9.1.5 - Eventi | 58 |
| 9.2 - Modello specifica | 59 |
| 9.2.1 - Probabilità di routing | 59 |
| 9.2.2 - Matrice di routing | 59 |
| 9.2.3 - Equazioni di traffico | 60 |
| 9.2.4 - Modellazione centri | 61 |
| Centro 6: Fast track (nuovo) | 61 |
| Centro 4: Trace Detection (modificato) | 62 |
| Centro 3: Controlli a Raggi X (modificato) | 62 |
| 9.3 - Modello computazionale | 62 |
| 9.3.1 - Jobs che attraversano la rete | 62 |
| 9.3.2 - Nuovo routing point | 63 |
| 9.3.3 - Improved simulation model | 63 |
| 9.4 - Verifica | 64 |
| 9.4.1 - Verifica del centro check-in M/M/8 | 64 |
| 9.4.2 - Verifica del centro varchi elettronici M/M/4 | 65 |
| 9.4.3 - Verifica del centro x-rays M/M/6 | 67 |
| 9.4.4 - Verifica del centro fast track M/M/3 | 68 |
| 9.4.5 - Verifica del centro trace detection M/M/2 | 69 |
| 9.4.6 - Verifica del centro recupero M/M/ ∞ | 70 |
| 9.4.7 - Controlli di consistenza | 71 |
| 9.5 - Validazione | 71 |
| 9.5.1 - Verifica delle Probabilità di Instradamento | 72 |
| 9.5.2 - Verifica dei Volumi di Traffico | 72 |

| | |
|--|-----------|
| 9.5.3 - Verifica dei Parametri di Servizio | 72 |
| 9.5.4 - Valutazione Macroscopica (Legge di Little) | 73 |
| 9.5.5 - Verifiche di Consistenza | 74 |
| 9.6 - Design degli esperimenti | 74 |
| 9.6.1 - Analisi transitorio | 74 |
| 9.6.2 - Simulazione ad orizzonte finito | 77 |
| 9.6.2.1 - Analisi del collo di bottiglia | 77 |
| 9.6.2.2 - Risultati della simulazione operativa | 78 |
| 9.6.3 - Simulazione ad orizzonte infinito | 82 |
| 10 - Conclusioni | 86 |
| Riferimenti Bibliografici | 88 |

Il presente elaborato illustra la progettazione, lo sviluppo e l'analisi di un simulatore a eventi discreti finalizzato allo studio dei flussi passeggeri all'interno di un terminal aeroportuale. L'obiettivo principale è valutare le prestazioni del sistema in termini di tempi di attesa, lunghezze delle code e utilizzo delle risorse, con particolare attenzione ai vincoli di Qualità del Servizio e Sicurezza.

1 - Descrizione del sistema

1.1 - Introduzione

Il sistema oggetto dell'analisi è il processo di accettazione e controllo dei passeggeri presso il Terminal Partenze dell'**aeroporto di Roma Ciampino (CIA)**, gestito dalla società **Aeroporti di Roma (AdR)**. La scelta di analizzare l'aeroporto di Ciampino è motivata dalla disponibilità di dati storici sul traffico e di documenti ufficiali, come il "Regolamento di Scalo" e la "Carta dei Servizi" pubblicati da AdR, che forniscono dei parametri precisi per modellare il sistema e gli obiettivi di qualità (QoS).

I passeggeri giungono in aeroporto secondo un processo di arrivo non stazionario, dipendente dall'orario di partenza (**STD - Scheduled Time of Departure**) dei voli. Tipicamente, per i voli Schengen, su cui si concentra il presente studio, la finestra temporale di arrivo è compresa tra 120 e 40 minuti prima della partenza, definita in base all'orario di apertura dei banchi check-in, descritto nel *regolamento di scalo* [1, pag. 77]. All'ingresso, il flusso di passeggeri si divide in due percorsi distinti in base alla tipologia di check-in effettuata:

- **Percorso A - Banco check-in:** Si dirigono qui i passeggeri che necessitano di imbarcare il bagaglio in stiva e/o richiedono l'emissione della carta d'imbarco.
- **Percorso B - Accesso diretto:** Dedicato ai passeggeri che hanno già effettuato il check-in online e viaggiano con il solo bagaglio a mano e quindi possono saltare il primo centro e dirigersi direttamente ai controlli.

Entrambi i flussi (assumendo che tutti gli utenti siano muniti di regolare titolo di viaggio) convergono verso i varchi di accesso elettronici (E-gates) per la lettura automatizzata del titolo di viaggio. Superato questo secondo centro, i passeggeri si dirigono ai controlli di sicurezza a raggi X. A questo punto ci sono tre articolazioni possibili del processo di controllo:

- **Percorso A - Servizio standard:** la maggior parte dei passeggeri segue un flusso standard, cioè si utilizza una vaschetta di plastica per riporre gli oggetti personali (cellulare, portafoglio, gioielli, ecc.), che insieme al bagaglio a mano, passeranno attraverso lo scanner a raggi X. Nel frattempo il passeggero transita attraverso il metal detector, per poi recuperare le proprie cose.
- **Percorso B - Controllo a campione:** una frazione di passeggeri viene selezionata per controlli a campione (es. rilevazione esplosivi o test antidroga) o semplicemente per approfondire i controlli in seguito al passaggio del bagaglio nello scanner. Questo introduce tempi di servizio aggiuntivi con instradamenti verso postazioni dedicate.
- **Percorso C - Uscita dal sistema:** infine, una minima percentuale di utenti può essere espulsa dal sistema a causa di non conformità (es. oggetti proibiti).

Gli utenti che completano con successo i controlli accedono all'area **airside** (area commerciale e gate), dove concludiamo l'analisi.

1.2 - Problematiche

L'aeroporto di Ciampino rappresenta uno scalo primario per il traffico **low-cost** e **turistico**, e questo comporta la possibilità di una forte congestione del sistema, soprattutto in corrispondenza di periodi di vacanza e festività, sui quali rivolgiamo particolare attenzione nell'analisi. Ricordiamo che i passeggeri spesso arrivano in raffiche nelle ore antecedenti le partenze dei voli, mettendo sotto pressione i centri di servizio.

Tipicamente le code si originano presso i **banchi check-in**, che possono essere sottodimensionati rispetto ai picchi di passeggeri che devono imbarcare bagagli da stiva, e presso i **centri di controllo a raggi X**, a causa della scarsa familiarità dei passeggeri con le procedure su liquidi e dispositivi elettronici o alla presenza di controlli casuali più approfonditi che introducono variabilità nei tempi di servizio.

A differenza di altri sistemi di trasporto, dove un ritardo comporta la semplice attesa della corsa successiva, in ambito aeroportuale il formarsi di lunghe code ai controlli può portare alla perdita del volo da parte del passeggero, con danni economici e d'immagine per lo scalo. Inoltre, un altro aspetto importante da tenere in considerazione sono le **dimensioni contenute** dell'aeroporto di Ciampino, infatti, dei ritardi nelle varie operazioni necessarie per l'imbarco, possono condurre alla saturazione della capacità fisica dell'edificio, creando sovraffollamenti che violano i vincoli previsti dalle norme di sicurezza.

2 - Obiettivi dello studio

2.1 - Requisiti qualità del servizio e sicurezza

Alla luce delle problematiche evidenziate, lo studio si propone di verificare se l'attuale configurazione del sistema sia in grado di garantire i livelli di servizio e di sicurezza richiesti, soprattutto nei periodi di picco (festività o periodi di vacanza). Gli **obiettivi** definiti sono i seguenti:

1. Il primo obiettivo riguarda le prestazioni presso il centro dei controlli di sicurezza a raggi X. In conformità con il *regolamento di scalo* [1, pag. 145], il tempo medio di attesa in coda non deve superare i 15 min: $E[T_{Q3}] \leq 15$ minuti.
2. Il secondo obiettivo verifica il **rispetto dei vincoli di affollamento** necessari per garantire sicurezza ed un'evacuazione efficace.
 - In particolare, il *D.M. 17 luglio 2014* [2, pag. 6], specifica i seguenti limiti: zona in prossimità dei controlli di sicurezza: 0.40 persone/m²
 - Nel caso di Ciampino, nella [sezione "Servizi" del sito di AdR](#), viene riportato che "il Terminal è una struttura elegante e moderna, di circa 1.800 metri quadri". Inoltre, dalla [sezione "Mappa dell'aeroporto](#) dello stesso sito, possiamo stimare in maniera approssimativa che l'area dedicata ai controlli di sicurezza occupi circa 1/3 della superficie totale. Di conseguenza, i limiti,

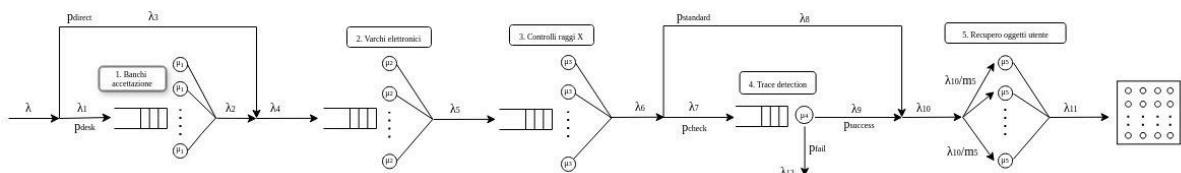
espressi in termine di numero massimo di passeggeri, diventano: zona in prossimità dei controlli di sicurezza: 240 persone: $E[N_{s_controlli}] \leq 240$

3. L'ultimo obiettivo mira a verificare l'efficacia della raccomandazione tipica secondo cui bisogna presentarsi in aeroporto 120 minuti prima della partenza. Ci chiediamo se i passeggeri siano in grado di completare l'intero percorso (dall'ingresso al terminal fino all'imbarco) **in meno di 80 minuti**, che è la media tra il tempo raccomandato minimo (STD-40') e massimo (STD-120'): $E[T_s] \leq 80$ minuti.

3 - Modello concettuale

3.1 - Diagramma

Il diagramma del sistema considerato modella il flusso dei passeggeri in partenza (Schengen) presso l'aeroporto di Ciampino.



3.2 - Descrizione

3.2.1 - Banchi Accettazione e Varchi Elettronici (Centro 1 e 2)

Gli utenti in arrivo presso il terminal, ciascuno modellato come un job nella simulazione, si dividono in due flussi sulla base delle necessità di accettazione. In particolare:

- Una parte di utenti (p_{desk}) si dirige ai **banchi accettazione (centro 1)** per imbarco bagagli da stiva o stampa carta d'imbarco. Questo centro è del tipo **MSSQ (Multi Server Single Queue)**, quindi prevede una coda unica che alimenta molteplici serventi in parallelo.
 - Se al momento dell'arrivo dell'utente almeno un server è libero, l'utente viene immediatamente servito; altrimenti, viene messo in coda secondo uno scheduling **FIFO (First In First Out)**.
 - Quando un server si libera, preleva il primo utente dalla testa della coda unica. Una volta serviti, gli utenti lasciano il centro e raggiungono il nodo successivo.
- La restante parte (p_{direct}) possiede già il titolo di viaggio digitale e bagaglio a mano (differente dal bagaglio da stiva), recandosi quindi direttamente ai varchi di accesso (saltando il primo centro).

Successivamente, i flussi si ricongiungono presso i **varchi elettronici**, chiamati anche E-Gates (**centro 2**), per la lettura del titolo di viaggio. In questo centro non si modella una probabilità di scarto, assumendo che i passeggeri giunti a questo punto siano muniti di titolo valido. Sebbene nella configurazione fisica reale ogni varco disponga di una propria corsia dedicata, nel modello si è deciso di approssimare il centro come un sistema **MSSQ (Multi Server Single Queue)**. Tale scelta è giustificata dal comportamento naturale dei passeggeri, che tendono ad adottare una logica SQF (Shortest Queue First) con la quale si dirigono verso il server con il minor carico. Di conseguenza, questo meccanismo rende le prestazioni del sistema comparabili a quelle di un'unica coda condivisa, e in tal modo possiamo fornire una dettagliata trattazione analitica del modello. Pertanto, assumiamo la presenza di una singola coda connessa ai server paralleli, ordinata secondo una disciplina FIFO.

3.2.2 - Controlli di Sicurezza (Centro 3, 4 e 5)

Superati i varchi, tutti i passeggeri si dirigono verso il centro per i **controlli di sicurezza a raggi X (centro 3)**. E' necessario precisare che i controlli di sicurezza si articolano in diversi passaggi:

- Per prima cosa, il passeggero impiega del tempo per svestirsi (cintura, gioielli, ecc.), separare i liquidi dal resto del bagaglio ed estrarre i dispositivi elettronici.
- Poi avviene l'effettiva scansione sotto la macchina a raggi X, sia del passeggero, sia del suo bagaglio e dei vari oggetti personali, posizionati nelle apposite vaschette.

Nel modello proposto, le prime due fasi (preparazione e scansione vera e propria) sono integrate in un unico centro di servizio, per l'appunto il centro 3. Questa scelta riflette l'organizzazione fisica reale dell'aeroporto: l'area di preparazione dispone di una postazione per ogni macchina a raggi X, di conseguenza, un passeggero non può accedere alla fase di scansione se non ha prima liberato la postazione di preparazione. Di fatti, la fase di preparazione diventa parte integrante del tempo di occupazione della risorsa ($S_{xRay} = T_{preparazione} + T_{scansione}$), e quindi le code si formano a monte dell'intera isola di controllo, poiché la saturazione delle postazioni di preparazione blocca l'inizio del servizio da parte di nuovi utenti in arrivo. Al contrario, la fase di **recupero oggetti utente** è stata modellata come un centro logico separato, indicato come **centro 5**, situato **dopo** il centro 3. Questo centro viene rappresentato come un nodo **Infinite Server (IS)**. Tale scelta è dovuta alla conformazione dell'area airside, dotata di numerose panchine e ampi spazi che permettono a molti passeggeri di rivestirsi in parallelo senza interferire con il flusso in uscita dalla macchina a raggi X e senza bloccare le risorse

Dunque, ora descriviamo meglio il centro dei **controlli di sicurezza a raggi X (centro 3)**. Qui il modello prevede diversi possibili comportamenti:

- Gran parte dei passeggeri viene sottoposta al controllo standard ($p_{standard}$).
- Ma una frazione di essi (p_{check}) può essere selezionata per controlli aggiuntivi (rilevazione tracce esplosivi, test antidroga, approfondimenti con apertura bagaglio), incrementando il tempo di permanenza a causa dell'accodamento al centro di **trace detection (centro 4)**.

- Inoltre, dal centro 4, è prevista una piccola probabilità di uscita dal sistema (p_{fail}) per modellare i passeggeri che, non superando i controlli per possesso di oggetti proibiti, vengono allontanati.

Quindi, il terzo centro (raggi X), è modellato come un **MSSQ**, anche qui assumendo che le code multiple possano essere approssimate ad una singola coda grazie alla presenza di apposito personale che si occupa di smistare i passeggeri affinché si distribuiscano equamente tra le code.

Mentre, il quinto centro (trace detection), è modellato come un **SSQ (Single Server Queue)** che prevede un'unica coda che alimenta un singolo server:

- Gli utenti in arrivo si accodano in un'unica linea d'attesa e vengono serviti uno alla volta dal servente.
- La disciplina di servizio adottata è **FIFO** (First-In First-Out): il server, non appena diventa libero, preleva l'utente che si trova in testa alla coda.
- Se al momento dell'arrivo di un nuovo utente il server è già libero (e quindi la coda è vuota), il servizio inizia immediatamente senza attesa.
- In caso contrario, l'utente deve attendere il completamento del servizio di tutti coloro che lo precedono.

Superati i controlli, i passeggeri recuperano i loro oggetti (centro 5) ed entrano nell'**area Airside** (negozi e gate).

3.3 - Sintesi

Dunque, ricapitolando, i centri di servizio sono i seguenti:

- Centro 1 - Banchi accettazione: MSSQ
- Centro 2 - Varchi elettronici: MSSQ
- Centro 3 - Controlli a raggi X: MSSQ
- Centro 4 - Trace detection: SSQ
- Centro 5 - Recupero oggetti utente: IS

Mentre le probabilità:

- p_{desk} = probabilità che i passeggeri si dirigano al centro 1 per l'accettazione (imbarco bagaglio in stiva o problemi tecnici).
- p_{direct} = probabilità che i passeggeri saltino il centro 1, in quanto hanno già fatto il check-in online e possiedono solamente il bagaglio a mano.
- $p_{standard}$ = probabilità che il passeggero superi i controlli di sicurezza e non venga selezionato per il controllo aggiuntivo a campione.
- p_{check} = probabilità che il passeggero venga selezionato per il controllo a campione (test esplosivi e antidroga) oppure il suo bagaglio a mano necessiti di ulteriori approfondimenti in seguito al passaggio all'interno dello scanner.

- p_{fail} = probabilità che il passeggero venga espulso dal sistema a causa di oggetti proibiti emersi dal controllo di sicurezza (in tal caso è sempre incluso l'approfondimento nel centro 4).

3.4 - Stato

Lo stato del sistema al tempo t , indicato con $S(t)$, è definito dall'insieme delle variabili di stato necessarie a descrivere la configurazione del sistema in quel preciso istante. Dato che il sistema è costituito da diversi centri eterogenei, descriviamo lo stato con il seguente vettore:

$$S(t) = (N_1, N_2, N_3, N_4, N_5)$$

Dove:

- N_1 (**Stato Centro 1 - MSSQ**): Un numero intero che rappresenta il numero totale di utenti presenti ai banchi accettazione (utenti in servizio + utenti in coda). Essendo una coda unica, questo scalare è sufficiente a descrivere lo stato: se $N_1 > m_1$, dove m_1 è il numero di server presenti presso il centro 1, allora $N_1 - m_1$ sono in coda.
- N_2 (**Stato Centro 2 - MSSQ**): Un numero intero che rappresenta il numero totale di utenti presenti ai varchi elettronici (utenti in servizio + utenti in coda).
- N_3 (**Stato Centro 3 - MSSQ**): Un numero intero che rappresenta il numero totale di utenti presenti ai controlli a raggi X (utenti in servizio + utenti in coda).
- N_4 (**Stato Centro 4 - SSQ**): Un numero intero che indica il numero di utenti al controllo trace detection (in servizio + eventuale coda). E' sufficiente un solo valore essendo una coda unica con un singolo server.
- N_5 (**Stato Centro 5 - IS**): Un numero intero che rappresenta il numero totale di utenti presenti alle banchine per la sistemazione degli oggetti recuperati (unicamente utenti in servizio, essendo un Infinite Server).

3.5 - Eventi

Gli eventi che modificano lo stato del sistema sono:

1. **Arrivo Servizio Banchi Accettazione (arrival_service_1)**: Rappresenta l'ingresso di un nuovo passeggero nel servizio accettazione.
2. **Fine Servizio Banchi Accettazione (end_service_1)**: Il passeggero completa le operazioni al banco accettazione. Il server viene liberato (processando il prossimo utente in coda, se presente) e il passeggero transita istantaneamente verso il Centro 2 (varchi elettronici).

3. **Arrivo Servizio Varchi Elettronici (arrival_service_2):** Rappresenta l'ingresso di un nuovo passeggero nel servizio varchi elettronici.
4. **Fine Servizio Varchi Elettronici (end_service_2):** Il passeggero attraversa il tornello. Si libera la risorsa specifica tra i molteplici server e l'utente si dirige verso il Centro 3.
5. **Arrivo Servizio Controllo a Raggi X (arrival_service_3):** Rappresenta l'ingresso di un nuovo passeggero nel servizio controllo a raggi X.
6. **Fine Servizio Controllo a Raggi X (end_service_3):** Il passeggero completa il controllo raggi X. In questo istante avviene la decisione probabilistica: l'utente può dirigersi verso le banchine per il recupero degli oggetti personali oppure essere instradato verso il controllo approfondito (Centro 4).
7. **Arrivo Servizio Trace Detection (arrival_service_4):** Rappresenta l'ingresso di un nuovo passeggero nel servizio trace detection.
8. **Fine Servizio Trace Detection (end_service_4):** Il passeggero termina il controllo aggiuntivo. Si verifica la condizione di espulsione (p_{fail}): l'utente può uscire definitivamente dal sistema (fallimento) o rientrare nel flusso verso l'area Airside (successo) passando prima per le banchine per il recupero degli oggetti personali.
9. **Arrivo Servizio Recupero Oggetti Utente (arrival_service_5):** Rappresenta l'ingresso di un nuovo passeggero nell'area per il recupero degli oggetti personali dalle vaschette successive al servizio di controllo a raggi X e trace detection.
10. **Fine Servizio Recupero Oggetti Utente (end_service_5):** Il passeggero completa la fase di recupero e si dirige verso l'area airside.
11. **Campionamento (sampling):** Evento periodico utilizzato per registrare le osservazioni statistiche necessarie al calcolo degli indici di prestazione senza modificare lo stato fisico delle code.

4 - Modello di specifica

4.1 - Processo di arrivo

I passeggeri in arrivo presso il terminal sono modellati come job nella simulazione e vengono generati secondo un processo di arrivo legato allo scheduling dei voli.

- Per prima cosa, dai [report sul traffico di AdR](#), troviamo il numero totale di passeggeri commerciali in un mese di picco. Più precisamente, viene calcolato un valore medio considerando i dati nei mesi di Giugno, Luglio, Agosto e Dicembre dal 2025 fino a Luglio 2021 (post-Covid19):

$$\frac{354.145 + 350.785 + 335.792 + 302.106 + 342.599 + 338.978 + 325.146 + \dots + 350.587}{18} = \frac{5.901.592}{18}$$

Per un totale di 327.866.222222 passeggeri in un mese di picco

- Facciamo lo stesso per i movimenti commerciali:

$$\frac{3.525 + 4.267 + 3.962 + 2.932 + 3.675 + 4.140 + 3.818 + \dots + 4.807}{18} = \frac{66.347}{18}$$

Per un totale di 3.685,944444 movimenti in un mese di picco.

- Ora, nel gergo aeroportuale il termine “movimenti” si riferisce sia alle partenze sia agli arrivi, tuttavia, ai fini dell’analisi, sono rilevanti unicamente le partenze. Quindi approssimativamente calcoliamo:

$$\frac{3.685,944444}{2} = 1.842,972222 \text{ partenze in un mese di picco.}$$

- Di seguito, daremo per scontato che la trattazione stia utilizzando i dati in un mese di picco, salvo diverse indicazioni. Inoltre con il termine “volo” indichiamo i voli di partenza, salvo variazioni opportunamente segnalate.
- Ora calcoliamo il numero medio di passeggeri per ciascun volo:

$$\frac{327.866,222222 \text{ passeggeri}}{1.842,972222 \text{ voli}} = 177,900794 \text{ passeggeri/volo}$$

Questi dati hanno perfettamente senso visto che a Ciampino, e in generale per tutti gli aeroporti che trattano il traffico low-cost in area Schengen, si utilizzano principalmente aerei Boeing 737 e, in misura minore, Airbus A320/A321, che hanno rispettivamente un massimo di circa 180 e 230 passeggeri.

Definiamo lo scenario peggiore:

- Il *regolamento di scalo* [1, pag. 64] indica che 65 è il tetto massimo di movimenti giornalieri, quindi circa 32-33 partenze, distribuite nell’orario di operatività dell’aeroporto (06:00 - 24:00, cioè 18h). Analizzando le [tabelle giornaliere dei voli](#) scopriamo che nelle ore di picco possono esserci fino a 6 voli in un’ora (60 min), che corrisponde al 18% delle partenze giornaliere ed è quindi plausibile.
- Ora i passeggeri di questi voli arriveranno in una finestra temporale corrispondente a quella del possibile check-in (STD-120’ a STD-40’), ovvero 80 min. Di conseguenza:

$$\lambda_{peak} = \frac{\text{voli nell’ora di picco} \times \text{passeggeri per volo}}{\text{durata finestra di arrivo}} = \frac{6 \times 177.900794}{80 \text{ min}}$$

Ovvero 13,343 passeggeri al minuto, cioè 0,222383 passeggeri al secondo.

Passiamo poi a definire lo scenario migliore, in cui il flusso di passeggeri è equamente distribuito su tutto il tempo di apertura dell’aeroporto (06:00 - 24:00, cioè 18h). Quindi andiamo a moltiplicare il numero massimo di voli in partenza in una giornata (33) per il numero medio di passeggeri per volo, dividendo poi per le 18h:

$$\bullet \quad \lambda_{low} = \frac{33 \times 177,900794 \text{ passeggeri}}{18 \times 3600 \text{ s}} = 0,090598 \text{ pax / s}$$

Tuttavia il flusso dei passeggeri in ingresso al terminal non è costante, bensì segue un andamento dinamico correlato allo scheduling dei voli. Per approssimare ad un valore costante, che sia il più rappresentativo possibile, andiamo ad osservare le [tabelle giornaliere dei voli](#), per calcolare una media pesata in base alla distribuzione dei voli nelle varie fasce

orarie. Nello specifico ci accorgiamo che, dividendo in fasce da 1h, abbiamo sei fasce (6.00 - 7.00, 9.00 - 10.00, 13.00 - 14.00, 15.00 - 16.00, 16.00 - 17.00 e 19.00 - 20.00) in cui si concentrano molti voli, e quindi assumiamo un tasso di arrivo pari a λ_{peak} . Mentre nelle restanti 12 fasce, assumiamo un tasso di arrivo pari a λ_{low} , di conseguenza:

- $\lambda_{med} = \frac{(0,222383 \times 5) + (0,090598 \times 13)}{18} \text{ pax/s} = 0,127205 \text{ pax / s}$

Inoltre, la natura degli arrivi è assunta come casuale e indipendente: il fatto che un passeggero arrivi all'istante t non influenza la probabilità di un arrivo successivo (proprietà di **assenza di memoria** o memoryless). Per concludere, decidiamo di modellare questo comportamento come un **Processo di Poisson Omogeneo**, caratterizzato da un tasso di arrivo $\lambda = \lambda_{med} = 0,127205 \text{ pax / s}$

Di conseguenza, nel simulatore, i tempi di interarrivo (X) saranno generati secondo una distribuzione **Esponenziale Negativa** con parametro $\lambda = \lambda_{med}$:

- Funzione di densità di probabilità: $f(t) = \lambda \cdot e^{-\lambda \cdot t}, x \geq 0$
- Media $E[X] = \frac{1}{\lambda} = \frac{1}{0,127205} = 7,861326 \text{ s secondi}$

4.2 - Probabilità di routing

Per quanto riguarda le probabilità p_{desk} e p_{direct} consideriamo le seguenti informazioni:

- Molte compagnie aeree, come Ryanair, stanno cercando di attuare una transizione verso il "100% digital boarding pass", con solo una minima frazione di utenti (problemi tecnici, visti extra-UE, smartphone scarichi) che necessitano di stampa carta d'imbarco al banco. Quindi con una percentuale di check-in online vicina al 98% e di conseguenza, una percentuale di check-in in presenza del 2%.
- Ma ci si reca al banco check-in anche per imbarcare i bagagli in stiva, e nello specifico, nel *Ryanair Annual Report 2024*, leggiamo che i ricavi accessori (*Ancillary Revenue*, inclusi bagagli da stiva e priority) siano circa il 37,4674% del fatturato totale [4, pag.4].
- Non possiamo sommare le due percentuali, perché rischieremmo di contare due volte le persone che hanno sia il bagaglio in stiva sia che non hanno fatto il digital boarding pass. Usiamo la formula $p_{desk} = P(A) + P(B) - P(A \cap B)$, dove A = "Il passeggero deve imbarcare la valigia" e B = "Il passeggero non ha fatto il digital boarding pass". Inoltre, assumendo che i due eventi siano indipendenti l'intersezione è il prodotto delle probabilità.

Quindi:

- $P(A) = P(\text{"Il passeggero deve imbarcare la valigia"}) = 0,374674$
- $P(B) = P(\text{"Il passeggero non ha fatto il digital boarding pass"}) = 0,02$
- $p_{desk} = 0,374674 + 0,02 - (0,374674 * 0,02) = 0,387181$
- $p_{direct} = 1 - p_{desk} = 0,612819$

Per quanto riguarda la probabilità che il passeggero venga selezionato per controlli più approfonditi (p_{check}), non sono disponibili dati pubblici per motivi di sicurezza nazionale.

Tuttavia, numerose ricerche pubblicate su ScienceDirect e ResearchGate analizzano l'efficacia dei controlli a campione utilizzando una frazione di campionamento del 10% per bilanciare sicurezza e fluidità del traffico. Mentre, in merito alla probabilità di espulsione dal sistema per motivi di sicurezza (p_{fail}), l'analisi dei report internazionali *IATA* [5] ed *ENAC* [6] evidenzia un tasso reale di incidenza estremamente basso (inferiore allo 0,01% per eventi *unruly* o *illicit items*). Dunque:

- $p_{check} = 0.1$
- $p_{fail} = 0.0001$
- $p_{standard} = 1 - p_{check} = 0.9$
- $p_{success} = 1 - p_{fail} = 0.9999$

4.3 - Matrice di routing

Definiamo i nodi del grafo $N = \{0, 1, 2, 3, 4, 5\}$

- **0:** Esterno / Ingresso
- **1:** Centro 1 - Banchi Check-in
- **2:** Centro 2 - Varchi Elettronici
- **3:** Centro 3 - Controlli a Raggi X
- **4:** Centro 4 - Trace Detection
- **5:** Centro 5 - Recupero Oggetti Utente

Dunque la matrice di routing P è definita come segue:

| | 0 | 1 | 2 | 3 | 4 | 5 |
|----------|------------|------------|--------------|----------|-------------|----------------|
| 0 | 0 | p_{desk} | p_{direct} | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | p_{check} | $p_{standard}$ |
| 4 | p_{fail} | 0 | 0 | 0 | 0 | $p_{success}$ |
| 5 | 1 | 0 | 0 | 0 | 0 | 0 |

4.4 - Equazioni di traffico

Le equazioni che descrivono il traffico nella rete di code sono le seguenti:

$$\lambda_1 = \lambda \cdot p_{desk}$$

$$\lambda_2 = \lambda_1$$

$$\lambda_3 = \lambda \cdot p_{direct}$$

$$\lambda_4 = \lambda_2 + \lambda_3$$

$$\lambda_5 = \lambda_4$$

$$\lambda_6 = \lambda_5$$

$$\lambda_7 = \lambda_6 \cdot p_{check}$$

$$\lambda_8 = \lambda_6 \cdot p_{standard}$$

$$\lambda_9 = \lambda_7 \cdot p_{success}$$

$$\lambda_{10} = \lambda_8 + \lambda_9$$

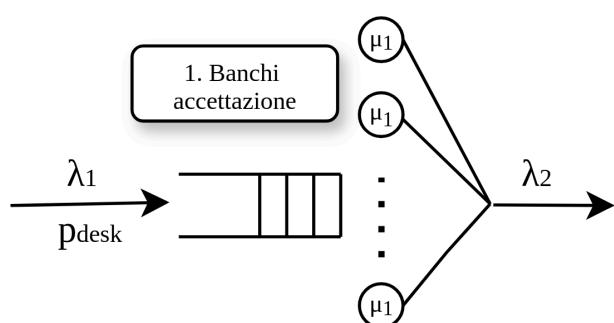
$$\lambda_{11} = \lambda_{10}$$

$$\lambda_{12} = \lambda_7 \cdot p_{fail}$$

4.5 - Modellazione centri

Per modellare i tempi di servizio dei vari centri, non possiamo utilizzare la distribuzione esponenziale, che è memoryless e ha la moda a 0. Per ogni centro, selezioniamo la distribuzione che meglio riflette la natura "umana" del processo e modella la presenza di un tempo tecnico minimo.

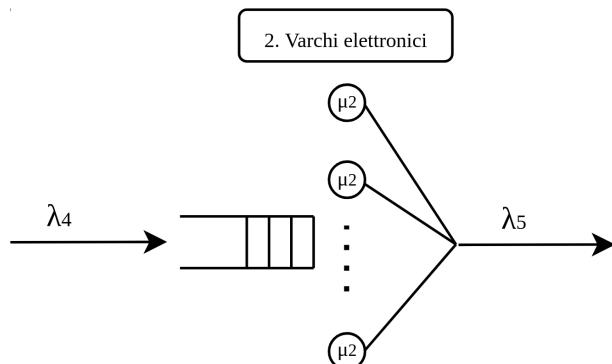
Centro 1: Banchi accettazione



- **Tipologia Centro:** Si tratta di un **MSSQ** (Multi Server Single Queue). Nello specifico dal regolamento di scalo [1, pag.77] leggiamo che "I banchi accettazione dovranno essere programmati ed utilizzati secondo le seguenti modalità operative: 1 banco accettazione ogni 50 passeggeri che necessitano di effettuare le operazioni check-in in aeroporto". Quindi, abbiamo visto che nello scenario di picco si ha $\lambda_{peak} = 13,343 \text{ passeggeri al minuto}$, quindi nella finestra STD-120' a STD-40' (80 min) di apertura dei banchi si hanno $13,343 \text{ passeggeri/min} \cdot 80 \text{ min} = 1067,44$ passeggeri. Ora sappiamo che $p_{desk} = 0,387181$, e quindi il numero di passeggeri che si dirigono ai banchi è $1067,44 \cdot 0,387181 = 413,292487$. Di conseguenza, saranno attivati $\frac{413,292487}{50} = 8,2658497$ banchi. Approssimiamo **m₁ = 8 server**.

- **Tipologia Processo:** La letteratura sulla modellazione dei servizi umani evidenzia che questi processi sono fortemente asimmetrici. Cioè, la maggior parte dei passeggeri richiede un tempo standard relativamente breve. Esiste però una "coda lunga" a destra rappresentata da una minoranza di utenti che generano tempi di servizio molto variabili (problemi con il peso bagaglio, ristampa carte d'imbarco, verifica visti).
- **Distribuzione:** La distribuzione **Normale Troncata** modella perfettamente questo comportamento.
- **Parametri:**
 - **Media ($E[S_1]$):** 120 secondi.
 - **Deviazione Standard (σ_1):** 60 secondi.
 - **Minimo (LB_1):** 60 secondi, è il limite fisico imposto dal sistema: scansione documento, stampa etichetta bagaglio e applicazione sulla valigia.
 - **Massimo (UB_1):** 180 secondi.

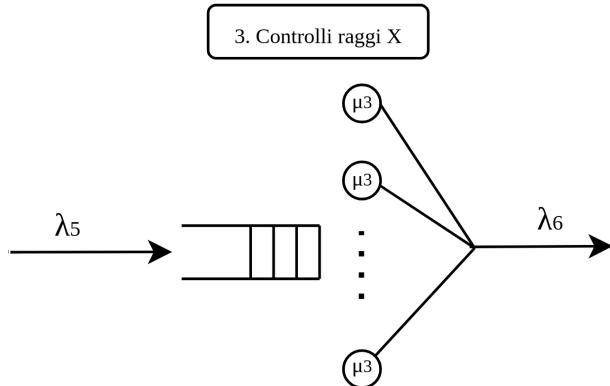
Centro 2: Varchi Elettronici (E-Gates)



- **Tipologia Centro:** Modellato come un **MSSQ (Multi Server Single Queue)**. Dal *regolamento di scalo* [1, pag. 85] leggiamo che "A monte di tale bacino avviene la verifica dell'idoneità del titolo di viaggio attraverso *n.4* tornelli elettronici...". Dunque abbiamo $m_2 = 4$ server per questo centro.
- **Tipologia Processo:** Questo è un processo prevalentemente meccanico e vincolato: scansione carta d'imbarco → apertura porte → attraversamento → chiusura.
- **Distribuzione:** La distribuzione **Normale Troncata** modella perfettamente questo comportamento.
- **Parametri:**
 - **Media ($E[S_2]$):** 15 secondi.
 - **Deviazione Standard (σ_2):** 15 secondi.

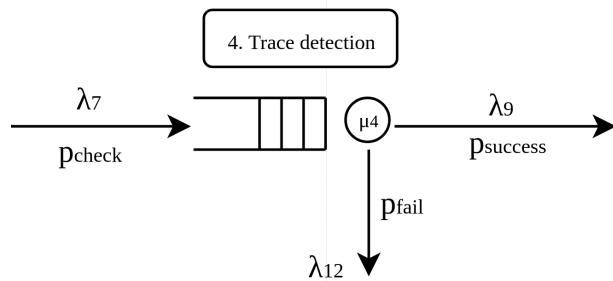
- **Minimo (LB₂)**: 10 secondi, limite fisico per scansionare la carta d'imbarco e attraversare il tornello.
- **Massimo (UB₂)**: 30 secondi.

Centro 3: Controlli a Raggi X



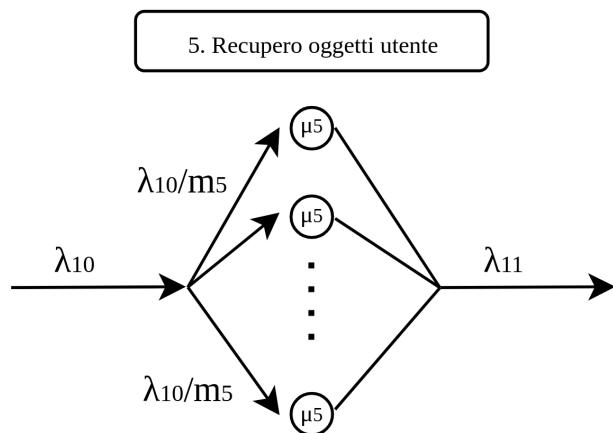
- **Tipologia Centro:** Modellato come un **MSSQ (Multi Server Single Queue)**. Dal *regolamento di scalo [1, pag. 85]* leggiamo che *"La postazione controlli di sicurezza è costituita da n.6 varchi di accesso dedicati ai passeggeri, identificati con numerazione progressiva dal n.1 al n.6..."*. Dunque abbiamo **$m_3 = 6$ server** per questo centro.
- **Tipologia Processo:** Come abbiamo preannunciato, il tempo di servizio modellato non rappresenta la sola scansione tecnica, bensì il tempo di occupazione della linea. Ovvero, sia il tempo in cui l'utente prepara le vaschette con il proprio bagaglio e oggetti personali, sia il tempo per il passaggio sotto la macchina a raggi X.
- **Distribuzione:** Si sceglie una **Normale Troncata**. Questa scelta è dettata dalla necessità di modellare la componente umana del processo: mentre la macchina ha tempi standard, i passeggeri possono introdurre una notevole variabilità. La troncatura è necessaria per evitare la generazione di tempi negativi o eccessivamente ridotti.
- **Parametri:**
 - **Media (E[S₃])**: 60 secondi.
 - **Deviazione Standard (σ_3)**: 30 secondi.
 - **Minimo (LB₃)**: 30 secondi.
 - **Massimo (UB₃)**: 70 secondi.

Centro 4: Trace Detection



- **Tipologia Centro e Processo:** Modellato come un **SSQ (Single Server Queue)** in cui vi è per l'appunto un operatore dedicato per tamponi o perquisizione personale e/o del bagaglio a mano.
- **Distribuzione:** Si sceglie la distribuzione **Normale Troncata**.
- **Parametri:**
 - **Media ($E[S_4]$):** 60 secondi.
 - **Deviazione Standard (σ_4):** 20 secondi.
 - **Minimo (LB₄):** 30 secondi.
 - **Massimo (UB₄):** 90 secondi.

Centro 5: Recupero oggetti utente



- **Tipologia Centro:** Modellato come un **Infinite Server**, quindi con $m_5 = \infty$. Anche in questo caso, le panchine e i rulli di scivolo a valle dei controlli permettono a molteplici utenti di rivestirsi in parallelo.
- **Tipologia Processo:** Human Activity, rappresenta il tempo di recupero oggetti, riallacciamento scarpe/cinture, riposizionamento laptop nello zaino. È un processo tipicamente più lento della preparazione.

- **Distribuzione:** Scegliamo anche qui la **Normale troncata**, per catturare la variabilità tra chi ha meno operazioni da compiere rispetto ad altri che devono recuperare meno oggetti.
- **Parametri:**
 - **Media ($E[S_5]$)**: 120 secondi.
 - **Deviazione Standard (σ_5)**: 30 secondi.
 - **Minimo (LB₅)**: 60 secondi.
 - **Massimo (UB₅)**: 180 secondi.

5 - Modello Computazionale

Il linguaggio di programmazione scelto è Java, il build system utilizzato è Maven che consente un rapido packaging in un unico file jar da utilizzare. Se si vuole eseguire il progetto, è sufficiente, dalla root di quest'ultimo, eseguire `mvn clean package` e poi `java -jar target/simulation-0.0.0-jar-with-dependencies.jar`. Apparirà il menu con gli esperimenti eseguibili.

5.1 - Introduzione alla struttura generale del codice sorgente

Il codice sorgente del progetto si trova sotto la cartella `src/main/java` del progetto e il package iniziale è `mbpmcsn` che contiene a sua volta altri subpackages:

- `center`: contiene le classi che definiscono i center e la loro logica di funzionamento.
- `core`: contiene le classi che definiscono le costanti del sistema e il modello di simulazione, ottenuto assemblando i vari center tramite il routing, processi, ecc.
- `csv`: contiene classi di utilità per scrivere i file CSV utilizzati per costruire i grafici e le annotazioni per la definizione dello schema di un file CSV.
- `desbook`: contiene le classi dal libro *“Discrete-Event Simulation: A First Course”*, utilizzate per la random variate generation, i multistream, ecc.
- `entity`: contiene la classe `Job`, che descrive un job che arriva al sistema e le sue proprietà che ne influenzano l'attraversamento del sistema.
- `event`: contiene le classi che implementano la logica definizione e gestione degli eventi del sistema, e il simulation clock.
- `process`: contiene classi che permettono una definizione veloce di processi di arrivo e di servizio, contiene il subpackage:
 - `rvg`: che contiene le classi che implementano un `RandomVariateGenerator`, ovvero le classi che, quando utilizzate, generano tempi di servizio esponenziali o truncated normal.
- `routing`: contiene le classi per la definizione di punti di routing nella rete (quando il job passa: qual è il prossimo centro? Bisogna mandare il job all'esterno della rete?)
- `runners`: contiene altri subpackage:

- `finitehorizon`: contiene il runner per l'orizzonte finito e una classe che definisce una `SingleReplication`. E' il main driver che genera output su terminale, CSVs per i grafici e gestisce le replicazioni.
- `steadystate`: contiene il runner per l'orizzonte infinito e una classe che definisce una `VeryLongRun`, riutilizzata per la verifica.
- `verification`: esegue il sistema per una "VeryLongRun" e confronta i risultati ottenuti con le formule analitiche.
- `smbuilders`: contiene dei builders per i simulation model
- `stats`: contiene altri subpackage:
 - `accumulating`: contiene classi per la definizione di time-based statistics, population-based statistics e di utilità
 - `batchmeans`: contiene classi di definizione e utilità per i batch means
 - `ie`: contiene una classe che fornisce metodo di utilità per calcolare la larghezza di un intervallo di confidenza
 - `sampling`: contiene classi di utilità per la raccolta di statistiche di un centro tramite gli eventi di sampling.

Contiene, inoltre, il file sorgente `App.java` che è il menu iniziale di scelta di esecuzione degli esperimenti.

I plot e le tabelle nella relazione sono stati prodotti grazie a degli script python che si possono trovare sotto la cartella `scripts/` a partire dalla root del progetto. Questi script usano `matplotlib`, `seaborn`, `pandas` per generare, dai CSV prodotti dall'applicativo di simulazione Java, questi plot e tabelle.

L'output in formato CSV dell'applicativo Java e i plot prodotti dagli script python si trovano in subdirectories di `output/`, sempre dalla root del progetto. Seguiamo un approccio di sviluppo del codice orientato alla riusabilità.

5.2 - Eventi, event queue e simulation clock

Gli eventi sono rappresentati dall'oggetto di tipo `mbpmcsn.event.Event`

```
public final class Event implements Comparable<Event> {

    /* time at which this event happens */
    private final double t;

    /* event type */
    private final EventType type;

    /* job info that triggered event */
    private final Job job;

    /* target center */
    private final Center targetCenter;

    /* optional args */
    private final Object args;
```

Identifica un evento che avverrà al tempo `t`, l'evento è di tipo `type`, il `job` che ha causato il trigger dell'evento, il `targetCenter` che è il centro in cui si verificherà l'evento, e `args` sono

argomenti opzionali passati poi all'implementazione del gestore dell'evento nel centro. Il `type` insieme al `targetCenter` identificano il tipo di evento nella sua completezza: per esempio sarà un arrival al check-in, una departure dai varchi elettronici, un sampling del centro degli X-Ray, ecc.

L'oggetto di tipo `mbpmcsn.event.EventType` è una enum:

```
public enum EventType {
    ARRIVAL,
    DEPARTURE,
    SAMPLING
}
```

La classe `mbpmcsn.event.EventQueue` è la coda di eventi e contiene il simulation clock che scorre con gli eventi: il nostro `Event` implementa l'interfaccia `Comparable<Event>` che permette un'implementazione semplice della event queue tramite la `PriorityQueue<T>` fornita dal package standard di Java - invocare `queue.pop()` restituisce l'evento più imminente (quello con l'attributo `t` più basso nella priority queue) e fa scorrere il simulation clock (l'attributo `currentClock` in `EventQueue`), contenuto sempre nella classe `EventQueue`. Gli eventi vengono inseriti, col metodo `add` di `EventQueue`, dai centri, dopo aver costruito l'oggetto `Event` che lo identifica.

```
public class EventQueue {

    // data struct for the queue
    private final PriorityQueue<Event> queue = new PriorityQueue<>();

    // time of the last processed event
    private double currentClock;

    // adds a new event to the queue in the correct time order
    public void add(Event e) {
        if (e.getTime() < currentClock) {
            System.err.println("WARNING: Tentativo di schedulare evento nel passato: " +
                + e.getTime() + " < " + currentClock);
        }
        queue.add(e);
    }

    /*
     * removes and returns the next event (lowest time)
     * updates currentClock
     */
    public Event pop() {
        Event e = queue.poll();
        if (e != null) {
            // updating current clock
            currentClock = e.getTime();
        }
        return e;
    }
}
```

5.3 - Jobs che attraversano la rete

L'oggetto `mbpmcsn.entity.Job` è il passeggero che effettua il percorso previsto prima dell'imbarco:

```

public final class Job {

    private static int ID_COUNTER = 0;
    private final int id;

    private final double arrivalTime; // entry time in the system
    private double lastQueuedTime; // T_in_queue (arrival at center)
    private double lastStartServiceTime; // T_start_service (service begins)
    private double lastEndServiceTime; // T_out (departure from center)

    private boolean checkedBaggage;
    private boolean securityCheckRequested;
    private boolean securityCheckFailed;
}

```

E' fondamentalmente una classe di getter e setter con vari attributi:

- `arrivalTime` è l'istante di entrata nel sistema.
- `lastQueuedTime` è l'istante in cui il job è stato accodato in un server.
- `lastStartServiceTime` è l'istante in cui il job ha iniziato ad essere servito.
- `lastEndServiceTime` è l'istante in cui il job ha completato il servizio.
- `checkedBaggage`: il passeggero ha eseguito la prenotazione online?
- `securityCheckRequested`: al passeggero è stato richiesto di effettuare ulteriori controlli di sicurezza?
- `securityCheckFailed`: il passeggero ha effettuato i controlli di sicurezza e ha fallito il controllo?

I primi 4 sono utilizzati per derivare le statistiche di tempi di risposta, tempi di servizio, tempi di coda, ecc... Gli ultimi 3 sono specifici del dominio d'applicazione e vengono settati dai "network routing point" in base alle probabilità di routing della rete (es. probabilità di entrare in trace detection?)

5.4 - Centri e routing

Il punto centrale del modello computazionale è sicuramente l'implementazione dei centri.

La classe astratta `mbpmcsn.center.Center` è utilizzata come base per le altre, perché contiene definizioni e interfacce comuni a tutti i tipi di centro:

```

public abstract class Center {

    protected final int id; // ID (es. Constants.ID_CHECKIN)
    protected final String name;

    protected final ServiceProcess serviceProcess;
    private final NetworkRoutingPoint networkRoutingPoint;
    protected final StatCollector statCollector;
    private final SampleCollector sampleCollector;
    private final BatchCollector batchCollector;
}

```

Possiamo vedere, senza entrare troppo nel dettaglio, che tutti i centri condividono l'utilizzo di oggetti di utility per collezionare statistiche, il `serviceProcess` che definisce il processo di servizio del server e il `NetworkRoutingPoint` che permette al centro di sapere verso quale altro centro inviare i job che ha terminato di processare.

```
/*...*/
protected Center(
    int id,
    String name,
    ServiceProcess serviceProcess,
    NetworkRoutingPoint networkRoutingPoint,
    StatCollector statCollector,
    SampleCollector sampleCollector,
    BatchCollector batchCollector) {
```

Il costruttore protetto di `Center`, invocabile solo da classi che ereditano da `Center`, permette una costruzione modulare, semplice ed estendibile del simulatore. Oltre a vari metodi helper che le classi che ereditano possono utilizzare, in particolare per la collezione di statistiche, in `Center` c'è la definizione di un'interfaccia molto importante per quanto riguarda la gestione degli eventi:

```
/* called by event handler */
public abstract void onArrival(Event event, EventQueue eventQueue);

/* called by event handler */
public abstract void onDeparture(Event event, EventQueue eventQueue);

/* called by event handler */
public final void onSampling(Event event, EventQueue eventQueue) {
    /*...*/
    doSample();
    /*...*/
}

/* called when a sampling event is received:
 * inheriting class must implement and return its specific data */
protected abstract Object doSample();
```

Le classi che ereditano da `Center` devono implementare i metodi astratti riportati sopra: in particolare l'event handler, quando effettua il `pop()` dalla `EventQueue`, ha a disposizione l'oggetto `targetCenter` in `Event`, sul quale può invocare `onArrival`, `onDeparture` e `onSampling`, in base al `type` dell'evento: le classi ereditarie che implementano l'interfaccia appena descritta, gestiranno gli eventi nel modo appropriato, ricevendo in input, tramite passaggio di parametri, l'`event` e la `eventQueue`. Si nota che `onSampling` è un metodo wrapper che contiene controlli aggiuntivi, e che è `doSample` il metodo che le classi ereditarie dovranno implementare per gestire gli eventi di sampling, ma è `onSampling`, che tutti gli oggetti ereditano, ad essere chiamato dall'event handler.

Per quanto riguarda i vari centri, le classi che ereditano da `Center` sono svariate: `InfiniteServer`, `MultiServerSingleQueue`, `SingleServerSingleQueue`. Tutte e quattro condividono la stessa logica, quindi, per esaminare a grandi linee l'implementazione, ne prendiamo uno come riferimento: per esempio il SSSQ.

Il costruttore e gli attributi sono abbastanza semplici: essendo singolo server e singola coda è sufficiente un booleano per indicare se il server è attualmente in esecuzione o no e una

coda di job da eseguire, la coda è chiaramente FIFO implementata con classi fornite dai package standard Java.

```
public class SingleServerSingleQueue extends Center {
    private boolean activeServer; // binary state
    private Queue<Job> jobQueue = new LinkedList<>(); // FIFO waiting line

    public SingleServerSingleQueue(
        int id,
        String name,
        ServiceProcess serviceProcess,
        NetworkRoutingPoint networkRoutingPoint,
        StatCollector statCollector,
        SampleCollector sampleCollector,
        BatchCollector batchCollector) {

        super(
            id,
            name,
            serviceProcess,
            networkRoutingPoint,
            statCollector,
            sampleCollector,
            batchCollector
        );
    }
}
```

L'implementazione del metodo `onArrival`, invocato dall'event handler è abbastanza semplice: dopo aver ottenuto l'istante di tempo corrente, si aggiornano i time stats, si incrementa il numero di job nel nodo dato che comunque la coda ha capacità infinita e quindi, o va in servizio se il server è idle (work-conserving) o viene aggiunto alla FIFO. Si aggiornano i timestamp del Job (`lastQueuedTime`, in particolare) al clock corrente. Si determina infine se, come già detto, il server è idle o meno: se è idle (`activeServer = false`) allora il job va eseguito immediatamente (il server è busy, quindi `activeServer = true`) e quindi si inserisce l'evento di departure in event queue con `scheduleDepartureEvent`. Se il server, al momento dell'arrival, è busy, inseriamo il job in coda, il metodo termina.

```
@Override
public void onArrival(Event event, EventQueue eventQueue) {
    double now = eventQueue.getCurrentClock();

    // 1. UPDATE TIME-BASED STATS
    collectTimeStats(now);

    // 2. UPDATE GLOBAL COUNTER
    numJobsInNode++;

    // 3. JOB TIMESTAMPING
    Job job = event.getJob();
    job.setLastQueuedTime(now); // T_in_queue

    // 4. RESOURCE CHECK
    if (activeServer) {
        jobQueue.add(job);
        return;
    }

    // Resource IDLE: activate immediately
    activeServer = true;
}
```

```

        scheduleDepartureEvent(now, job, eventQueue);
    }
}

```

Anche la `onDeparture` è molto semplice e lineare: si aggiornano/raccolgono statistiche, si decrementa il numero di job nel nodo, si determinano i tempi di servizio e risposta del job appena terminato, si determina il prossimo centro verso cui indirizzare il job: se `targetCenter` è diverso da `null` allora il job va effettivamente inviato verso un altro centro, e non verso l'esterno. Al contrario, se `targetCenter = null`, va mandato il job verso l'esterno. Come si "invia" il job al prossimo centro? Creando un nuovo evento di tipo `EventType.ARRIVAL`, con destinazione l'oggetto `targetCenter`, quindi aggiungendolo alla event queue. Infine, se la coda dell'SSSQ è vuota allora il server viene posto in stato `idle`, altrimenti viene estratto il prossimo job dalla coda FIFO e ed eseguito, quindi `scheduleDepartureEvent`.

```

@Override
public void onDeparture(Event event, EventQueue eventQueue) {
    double now = eventQueue.getCurrentClock();

    // 1. UPDATE TIME-BASED STATS
    collectTimeStats(now);

    /* collect batch stats */
    batchCollect(now);

    // 2. UPDATE GLOBAL COUNTER
    numJobsInNode--;

    // 3. JOB STATS RECORDING
    Job job = event.getJob();
    job.setLastEndServiceTime(now);

    sampleServiceTime(job);
    sampleResponseTime(job);

    // 4. ROUTING
    Center nextCenter = getNextCenter(job);

    if (nextCenter == null) {
        sampleSystemResponseTimeSuccess(now, job);
    } else {
        Event arrivalEvent = new Event(
            now, EventType.ARRIVAL, nextCenter, job, null);

        eventQueue.add(arrivalEvent);
    }

    // 5. RESOURCE MANAGEMENT
    if (jobQueue.isEmpty()) {
        activeServer = false;
        return;
    }

    job = jobQueue.remove();
    scheduleDepartureEvent(now, job, eventQueue);
}

```

Infine, il corpo del metodo `scheduleDepartureEvent`, che è utilizzato sia da `onArrival` che `onDeparture` quando opportuno. La proprietà del job `lastStartServiceTime` è settata ad "ora", è quindi possibile registrare il tempo di coda che ha affrontato il job perchè è appena entrato in servizio. Il server affronterà un certo tempo di servizio in base alla distribuzione scelta, quindi viene creato un evento di tipo `EventType.DEPARTURE` per questo stesso server (`this`), e viene aggiunto alla event queue.

```
private void scheduleDepartureEvent(
    double now, Job job, EventQueue eventQueue) {

    job.setLastStartServiceTime(now);

    sampleQueueTime(job); // Tq = T_start - T_in_queue

    double svc = serviceProcess.getService();
    Event departureEvent = new Event(
        now + svc, EventType.DEPARTURE, this, job, null);

    eventQueue.add(departureEvent);
}
```

Come si connettono i vari centri? Tramite i `mbpmcsn.routing.NetworkRoutingPoint`, che consiste in un'interfaccia che viene passata ai vari center e, quando opportunamente implementata, permette al centro di determinare il prossimo centro a cui inviare il job (si veda la chiamata a `getNextCenter` nel metodo `onDeparture`):

```
public interface NetworkRoutingPoint {
    /* passing job allows to change attributes of the incoming job,
     * that is, domain-specific infos, e.g. checkedBaggage, determined
     * by probabilities */
    Center getNextCenter(Rngs rngs, Job job);
}
```

E' un'interfaccia generica, quindi un esempio di implementazione è:

```
public final class FixedRouting implements NetworkRoutingPoint {

    private final Center nextCenter;

    public FixedRouting(Center nextCenter) {
        this.nextCenter = nextCenter;
    }

    @Override
    public Center getNextCenter(Rngs r, Job job) {
        return nextCenter;
    }
}
```

Se si vogliono connettere due centri A e B, in maniera deterministica quindi, sarà sufficiente passare al centro A l'oggetto `FixedRouting`, costruito come `new FixedRouting(centerB)`. Se si vuole un'esempio di routing più complesso:

```
public final class XRayRouting implements NetworkRoutingPoint {

    private final Center traceDetection;
    private final Center recovery;
```

```

private final int streamIndex;

public XRayRouting(Center traceDetection, Center recovery, int streamIndex) {
    this.traceDetection = traceDetection;
    this.recovery = recovery;
    this.streamIndex = streamIndex;
}

@Override
public Center getNextCenter(Rngs r, Job job) {
    r.selectStream(streamIndex);

    boolean areFurtherChecksNeeded = r.random() < P_CHECK;
    job.setSecurityCheckRequested(areFurtherChecksNeeded);
    return areFurtherChecksNeeded ? traceDetection : recovery;
}
}

```

In questo caso, dopo gli XRay: si manda il job in trace detection o direttamente verso il centro di recupero? Questo è determinato dalle probabilità, come descritto nel modello concettuale e XRayRouting ne è l'implementazione.

5.5 - Costruzione del simulation model e l'event handler

Il vantaggio di aver pensato a una soluzione più modulare e orientata al riuso del codice è che è semplice costruire il modello: basta, a tutti gli effetti assemblare i componenti, il tutto è nella classe mbpmcsn.core.BaseSimulationModel:

```

public final class BaseSimulationModel extends SimulationModel {

    /* arrival */
    private ArrivalProcess arrivalProcess;

    /* centers */
    private Center centerCheckIn; // ID 1
    private Center centerVarchi; // ID 2
    private Center centerXRay; // ID 4
    private Center centerTrace; // ID 5
    private Center centerRecupero; // ID 6

    private List<Center> centers;

    /* entry point routing */
    private NetworkRoutingPoint routingIngresso;

    /* service generators */
    private RandomVariateGenerator rvgCheckIn;
    private RandomVariateGenerator rvgVarchi;
    private RandomVariateGenerator rvgXRay;
    private RandomVariateGenerator rvgTrace;
    private RandomVariateGenerator rvgRecupero;
}

```

Per procedere all'istanziazione del simulatore, quindi creare i vari rvgs, verranno utilizzati dai vari centri (le costanti MEAN_S1, STD_S1, ... sono definite in mbpmcsn.core.Constants):

```

rvgCheckIn = new TruncatedNormalGenerator(MEAN_S1, STD_S1, LB1, UB1);
rvgVarchi = new TruncatedNormalGenerator(MEAN_S2, STD_S2, LB2, UB2);
rvgXRay = new TruncatedNormalGenerator(MEAN_S3, STD_S3, LB3, UB3);
rvgTrace = new TruncatedNormalGenerator(MEAN_S4, STD_S4, LB4, UB4);

```

```
rvgRecupero = new TruncatedNormalGenerator(MEAN_S5, STD_S5, LB5, UB5);
```

Bisogna istanziare la classe che genera gli arrivi (getArrival):

```
RandomVariateGenerator rvgArrival = new ExponentialGenerator(arrivalsMeanTime);
arrivalProcess = new ArrivalProcess(rvgArrival, rngs, STREAM_ARRIVALS);
```

I vari centri vengono quindi creati e “collegati” tramite i NetworkRoutingPoint, per ogni centro viene istanziato il ServiceProcess (che fornisce la getService) passando al costruttore di questa classe l’rvg istanziato prima, corrispondente al centro.

```
// 2. Varchi
ServiceProcess sp2 = new ServiceProcess(rvgVarchi, rngs, STREAM_S2_SERVICE);
NetworkRoutingPoint routingVarchi = new FixedRouting(centerXRay);
centerVarchi = new MultiServerSingleQueue(
    ID_VARCHI_ELETTRONICI,
    "Varchi",
    sp2,
    routingVarchi,
    statCollector,
    sampleCollector,
    batchCollector,
    M2
);

// 1. Check-in
ServiceProcess sp1 = new ServiceProcess(rvgCheckIn, rngs, STREAM_S1_SERVICE);
NetworkRoutingPoint routingCheckIn = new FixedRouting(centerVarchi);
centerCheckIn = new MultiServerSingleQueue(
    ID_BANCHI_CHECKIN,
    "CheckIn",
    sp1,
    routingCheckIn,
    statCollector,
    sampleCollector,
    batchCollector,
    M1
);

// 0. Infine, l'Ingresso (Ora che CheckIn e Varchi esistono)
routingIngresso = new EntryRouting(centerCheckIn, centerVarchi, STREAM_ARRIVALS);
```

La classe base **SimulationModel**, da cui eredita **BaseSimulationModel**, contiene il metodo **processEvent**, invocato dall’event handler. Lo scopo di **processEvent** è chiamare **onArrival**, **onDeparture** o **onSampling** del centro target:

```
/* called from the runner */
public final void processEvent(Event e) {
    // center that manages the event
    Center target = e.getTargetCenter();

    // if target == null --> exit job from the system
    if (target == null) {
        recordJobSystemExit(e);
        return;
    }

    switch (e.getType()) {
        case ARRIVAL:
            target.onArrival(e, eventQueue);
            break;
        case DEPARTURE:
            target.onDeparture(e, eventQueue);
    }
}
```

```

        break;
    case SAMPLING:
        target.onSampling(e, eventQueue);
        break;
    default:
        throw new IllegalStateException("Tipo evento sconosciuto: " + e.getType());
    }
}
}

```

L'event handler invoca anche il metodo `planNextArrival` del `SimulationModel`, lo scopo è ovviamente quello di generare il prossimo arrivo al sistema, generando il "next arrival time" con la `getArrival` (si vedano i paragrafi successivi per ulteriori dettagli a riguardo), creando un nuovo job e quindi determinando se il job va al check-in o direttamente ai varchi elettronici. Determinate tutte queste informazioni, si crea un nuovo evento di tipo `EventType.ARRIVAL`, al tempo "nextArrivalTime", e quindi aggiunto alla event queue:

```

/* called from the runner */
@Override
public final void planNextArrival() {
    // calculate when a pax arrives (update sarrival)
    double nextArrivalTime = arrivalProcess.getArrival();

    // create the job associated to the pax
    Job newJob = new Job(nextArrivalTime);

    // initial routing: Check-in o Varchi?
    Center firstCenter = routingIngresso.getNextCenter(rngs, newJob);

    // create event
    Event arrivalEvent = new Event(
        nextArrivalTime,
        EventType.ARRIVAL,
        firstCenter,
        newJob,
        null
    );
    eventQueue.add(arrivalEvent);
}

```

L'event handler, infine, è contenuto nei runners: ad esempio in `mbpmcsn.runners.finitehorizon.SingleReplication`, è il punto centrale del sistema: effettua il pop dalla event queue (che è una priority queue) del prossimo evento più imminente. Nello statement dell'if, si controlla se la simulazione deve continuare in base al simulation time del finite horizon (in questo caso), se l'evento che esce dalla event queue è un arrivo e se si tratta di un arrivo al sistema (`e.getTime() == e.getJob().getArrivalTime()`) - se la condizione appena descritta è vera allora genera il prossimo arrivo al sistema (`simulationModel.planNextArrival()`). In ogni caso, invoca il metodo `simulationModel.processEvent(e)` sull'evento appena tirato fuori dalla priority queue.

```

while (!eventQueue.isEmpty()) {
    boolean simulationMustContinue =
        eventQueue.getCurrentClock() < simulationTime;

```

```

    // extract the upcoming event and process
    Event e = eventQueue.pop();

    // new arrival, if simulation shall
    // terminate, await for queue to drain
    if (
        simulationMustContinue &&
        e.getType() == EventType.ARRIVAL &&
        e.getTime() == e.getJob().getArrivalTime()) {

        simulationModel.planNextArrival();
    }

    simulationModel.processEvent(e);
}

```

5.6 - Random number generation

Sempre a supporto di un design modulare, la classe base `mbpmcsn.process.Process` è utilizzata a supporto delle classi che ereditano da essa e che implementano effettivamente il `getService` e `getArrival`, dato che condividono alcuni elementi (entrambe hanno uno `stream`, l'`rngs` e l'`rvg`)

```

class Process {
    protected final Rngs rngs;
    protected final int streamIdx;
    protected final RandomVariateGenerator rvg;

    protected Process(RandomVariateGenerator rvg, Rngs rngs, int streamIdx) {
        this.streamIdx = streamIdx;
        this.rvg = rvg;
        this.rngs = rngs;
    }

    public RandomVariateGenerator getRvg() {
        return rvg;
    }

    public int getStreamIdx() {
        return streamIdx;
    }

    public Rngs getRngs() {
        return rngs;
    }
}

```

Una volta istanziata, `mbpmcsn.process.ArrivalProcess` è in grado di generare i tempi di interarrivo. Eredita da `mbpmcsn.process.Process`, e quindi ha tutti i suoi attributi, per cui serve passare al suo costruttore un `rvg`, un `rngs` e uno `stream index`. La `getArrival` selezionerà lo `stream` desiderato dall'`rngs`, genera il tempo dall'`rvg` e incrementa la sua proprietà specifica `sarrival`.

```

public final class ArrivalProcess extends Process {
    private double sarrival;

    public ArrivalProcess(RandomVariateGenerator rvg, Rngs rngs, int streamIdx) {
        super(rvg, rngs, streamIdx);
    }

    public double getArrival() {

```

```

        rngs.selectStream(streamIdx);
        sarrival += rvg.generate(rngs);
        return sarrival;
    }
}

```

La classe `mbpmcsn.process.ServiceProcess` implementa la `getService`, che è leggermente più semplice della `getArrival` perchè non ha la necessità di mantenere ulteriore stato: seleziona lo stream, quindi genera il tempo di servizio dall'rvg.

```

public final class ServiceProcess extends Process {
    public ServiceProcess(RandomVariateGenerator rvg, Rngs rngs, int streamIdx) {
        super(rvg, rngs, streamIdx);
    }

    public double getService() {
        rngs.selectStream(streamIdx);
        return rvg.generate(rngs);
    }
}

```

5.7 - Random variate generation

Questa interfaccia permette facilmente di specificare una funzione di generazione delle variate, da passare poi al costruttore di `mbpmcsn.process.ArrivalProcess` e `mbpmcsn.process.ServiceProcess`, in modo che possano essere generati con semplicità tempi di servizio esponenziali o truncated normal, per esempio.

```

public interface RandomVariateGenerator {
    double generate(Rngs rngs);
}

```

La classe `mbpmcsn.process.rvg.TruncatedNormalGenerator` implementa l'interfaccia `mbpmcsn.process.rvg.RandomVariateGenerator`: il metodo overridden `generate`, genera una normale troncata. Al costruttore della `TruncatedNormalGenerator` vengono passati i parametri della distribuzione e i lower/upper bound. Per la realizzazione della normale troncata, sono state seguite le linee guida del libro di testo di riferimento del corso.

```

public final class TruncatedNormalGenerator implements RandomVariateGenerator {
    private final double mean;
    private final double devstd;
    private final double lowerBound;
    private final double upperBound;

    public TruncatedNormalGenerator(double mean, double devstd, double lowerBound, double
upperBound) {
        this.mean = mean;
        this.devstd = devstd;
        this.lowerBound = lowerBound;
        this.upperBound = upperBound;
    }

    @Override
    public double generate(Rngs rngs) {
        Rvms rvms = new Rvms();

        double a = rvms.cdfNormal(mean, devstd, lowerBound - 1);
        double b = 1.0 - rvms.cdfNormal(mean, devstd, upperBound);
    }
}

```

```

        double u = rvms.idfUniform(a,1.0-b, rngs.random());
        return rvms.idfNormal(mean, devstd, u);
    }
}

```

Infine, per completezza, il codice della classe generatrice dei tempi esponenziali: `mbpmcsn.process.rvg.ExponentialGenerator` (come al solito, al costruttore viene passato il parametro della distribuzione e in `generate` c'è il corpo del generatore vero e proprio)

```

public final class ExponentialGenerator implements RandomVariateGenerator {
    /* costruttore con parametro passato mean
     * e attributo di istanza mean omessi */

    @Override
    public double generate(Rngs rngs) {
        return -mean * Math.log(1.0 - rngs.random());
    }
}

```

6 - Verifica

Nella fase di verifica ci preoccupiamo di rispondere alla domanda: *abbiamo costruito il modello correttamente?* Paragoniamo i risultati forniti dal simulatore con le formule analitiche dalla teoria delle code. Abbiamo a che fare con una rete di code: per poter fare il confronto dobbiamo fare delle approssimazioni:

- tempi di servizio esponenziali,
- **tempo medio di interarrivo raddoppiato rispetto al normale.**

Confronteremo i risultati generati dal simulatore nel caso stazionario, con simulazione ad orizzonte infinito. Le approssimazioni sopra elencate, sono attuate anche per il simulatore. L'intervallo di confidenza è del 95%. Confronteremo centro-per-centro.

Perchè si effettua l'approssimazione sui tempi medi di interarrivo? E' stato necessario prendere questa decisione a causa del fatto che il sistema non converge con il tasso di arrivi "regolare": come effettuare quindi la verifica se non è possibile raggiungere uno stato stazionario e soprattutto, l'utilizzazione è maggiore o uguale a 1, per almeno un centro, nelle formule analitiche?

Ridurre il tasso degli arrivi al sistema, mantenendo il sistema invariato (ovviamente) è risultata la scelta più ovvia per permettere la corretta esecuzione della fase di verifica.

L'output della verifica, ossia i confronti tra formule analitiche e risultati prodotti dal simulatore, è trascritto in tabella per ogni centro.

6.1 - Il problema dell'utilizzazione maggiore di 1

Andando ad eseguire il codice per il confronto delle formule analitiche con la simulazione a orizzonte infinito con tempo medio di interarrivo λ_{med} , l'output è il seguente, per il centro

XRay, che ricordiamo essere fin da subito sospettato di essere il collo di bottiglia del sistema:

```
>>> Centro: XRay [M/M/6] (Erlang-C)
[ERRORE CRITICO] Sistema Instabile (Rho = 1,2721 >= 1.0).
La teoria prevede code infinite. Impossibile verificare.
```

Infatti, osserviamo che $\rho_3 = \frac{\lambda_{med}}{m_3 \cdot \mu_3} = \frac{0,127205 \text{ pax/s}}{6 \cdot 0,016667 \text{ pax/s}} = 1,272024$. Si vuole comunque procedere alla verifica del simulatore costruito e la soluzione migliore è risultata essere quella di aumentare i tempi di interarrivo in modo da abbassare l'utilizzazione del centro e poter utilizzare le formule e i teoremi alla base della teoria delle reti di code, che richiedono la stabilità del sistema. Per le verifiche dei singoli centri, considereremo $\lambda_{ver} = \frac{\lambda_{med}}{2} = 0,063602 \text{ pax/s}$, che è in grado di far mantenere al centro XRay un'utilizzazione del 65% circa.

6.2 - Verifica del centro check-in M/M/8

Utilizziamo le formule per un centro M/M/8:

$$\rho_1 = \frac{\lambda_{ver} \cdot p_{desk}}{m_1 \cdot \mu_1} = 0,3694$$

$$p_{0,1} = \left(\sum_{i=0}^{m_1-1} \frac{(m_1 \rho_1)^i}{i!} + \frac{(m_1 \rho_1)^{m_1}}{m_1! (1-\rho_1)} \right)^{-1} = 0,052025$$

$$P_{Q,1} = \frac{(m_1 \rho_1)^{m_1}}{m_1! (1-\rho_1)} \cdot p_{0,1} = 0,011898$$

$$E(T_{Q,1}) = \frac{P_{Q,1}}{m_1 \cdot \mu_1 - \lambda_{ver} \cdot p_{desk}} = 0,2830s$$

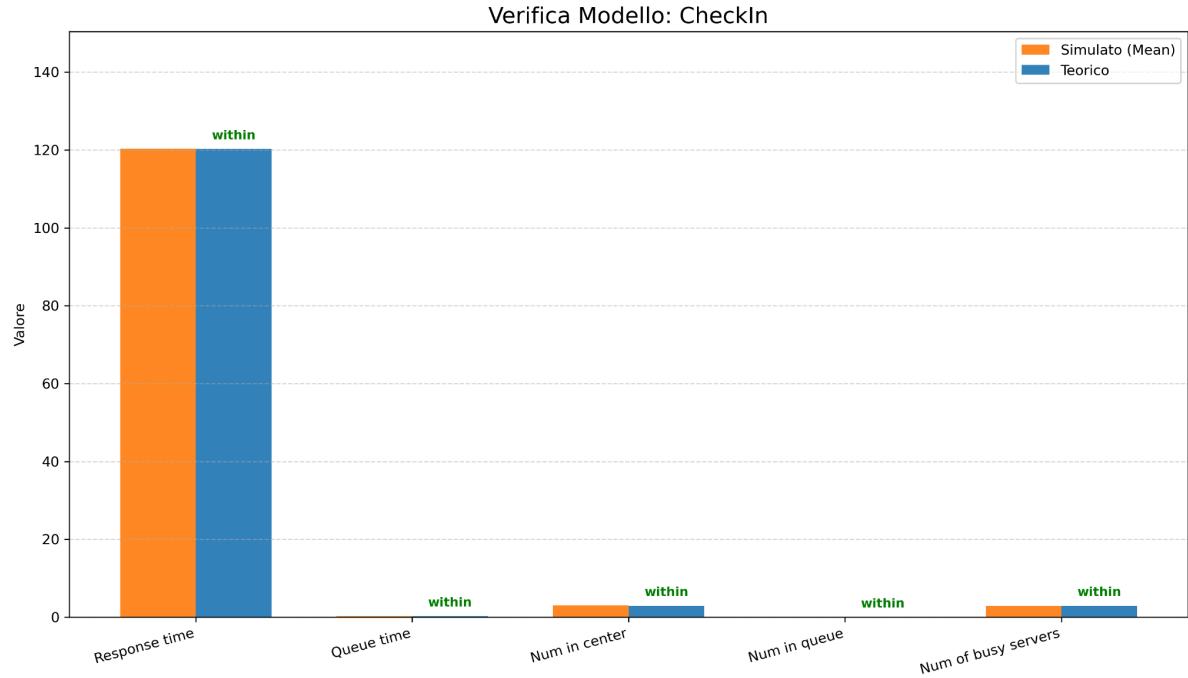
$$E(S_{i,1}) = \frac{1}{\mu_1} = 120s$$

$$E(T_{S,1}) = E(T_{Q,1}) + E(S_{i,1}) = 120,2830s$$

$$E(N_{Q,1}) = \lambda_{ver} \cdot p_{desk} \cdot E(T_{Q,1}) = 0,0070$$

$$E(N_{S,1}) = \lambda_{ver} \cdot p_{desk} \cdot E(T_{S,1}) = 2,9621$$

$$m_1 \rho_1 = 2,9551$$



6.3 - Verifica del centro varchi elettronici M/M/4

Utilizziamo le formule per un centro M/M/4:

$$\rho_2 = \frac{\lambda_{ver}}{m_2 \cdot \mu_2} = 0.2385$$

$$p_{0,2} = \left(\sum_{i=0}^{m_2-1} \frac{(m_2 \rho_2)^i}{i!} + \frac{(m_2 \rho_2)^{m_2}}{m_2! \cdot (1-\rho_2)} \right)^{-1} = 0.384736$$

$$P_{Q,2} = \frac{(m_2 \rho_2)^{m_2}}{m_2! \cdot (1-\rho_2)} \cdot p_{0,2} = 0.017440$$

$$E(T_{Q,2}) = \frac{P_{Q,2}}{m_2 \cdot \mu_2 - \lambda_{ver}} = 0,0859s$$

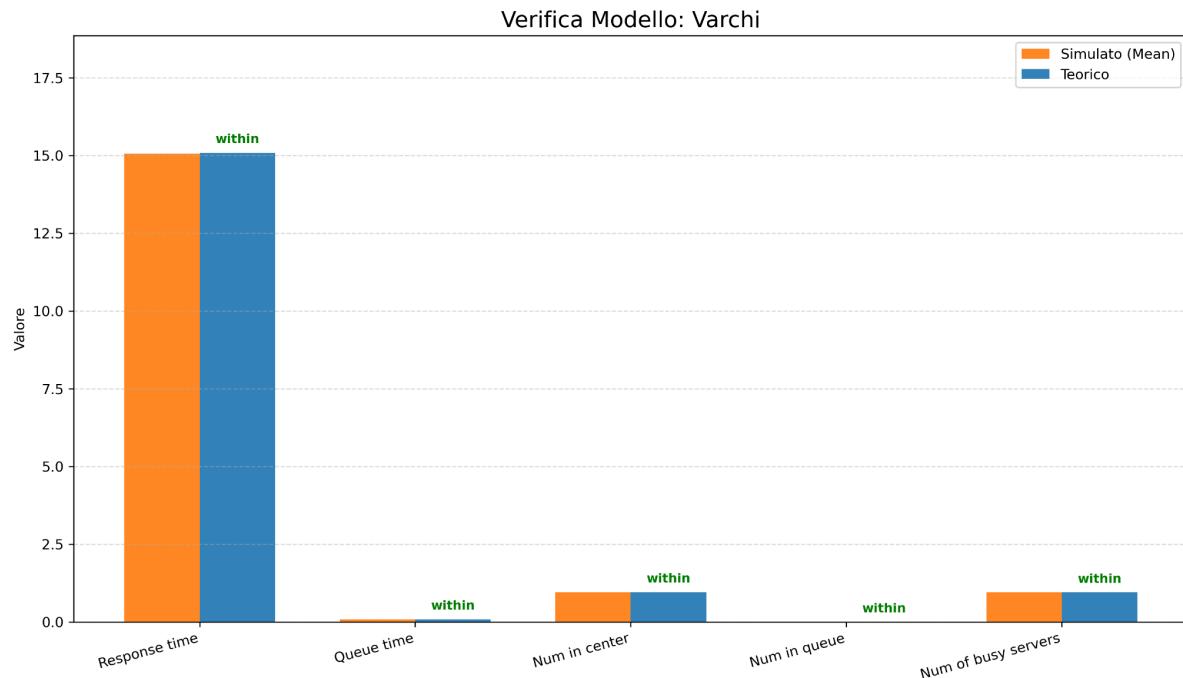
$$E(S_{i,2}) = \frac{1}{\mu_2} = 15s$$

$$E(T_{S,2}) = E(T_{Q,2}) + E(S_{i,2}) = 15,0859s$$

$$E(N_{Q,2}) = \lambda_{ver} \cdot E(T_{Q,2}) = 0,0055$$

$$E(N_{S,2}) = \lambda_{ver} \cdot E(T_{S,2}) = 0,9595$$

$$m_2 \rho_2 = 0,9540$$



| Metrica | SimMean | SimMin | SimMax | SimWidth | TheoValue | WithinInt |
|--------------------|----------|----------|----------|------------|-----------|-----------|
| $E(T_{S,2})$ | 15,0676s | 14,9751s | 15,1600s | 0,09245973 | 15,0859s | within |
| $E(T_{Q,2})$ | 0,0828s | 0,0730s | 0,0925s | 0,00974349 | 0,0859s | within |
| $E(N_{S,2})$ | 0,9608 | 0,9523 | 0,9693 | 0,00854563 | 0,9595 | within |
| $E(N_{Q,2})$ | 0,0053 | 0,0047 | 0,0059 | 0,00063145 | 0,0055 | within |
| $m_2 \cdot \rho_2$ | 0,9555 | 0,9472 | 0,9638 | 0,00833940 | 0,9540 | within |

6.4 - Verifica del centro x-rays M/M/6

Utilizziamo le formule per un centro M/M/6:

$$\rho_3 = \frac{\lambda_{ver}}{m_3 \cdot \mu_3} = 0,6360$$

$$p_{0,3} = \left(\sum_{i=0}^{m_3-1} \frac{(m_3 \rho_3)^i}{i!} + \frac{(m_3 \rho_3)^{m_3}}{m_3! \cdot (1-\rho_3)} \right)^{-1} = 0.020523$$

$$P_{Q,3} = \frac{(m_3 \rho_3)^{m_3}}{m_3! \cdot (1-\rho_3)} \cdot p_{0,3} = 0.241869$$

$$E(T_{Q,3}) = \frac{P_{Q,3}}{m_3 \cdot \mu_3 - \lambda_{ver}} = 6,6452s$$

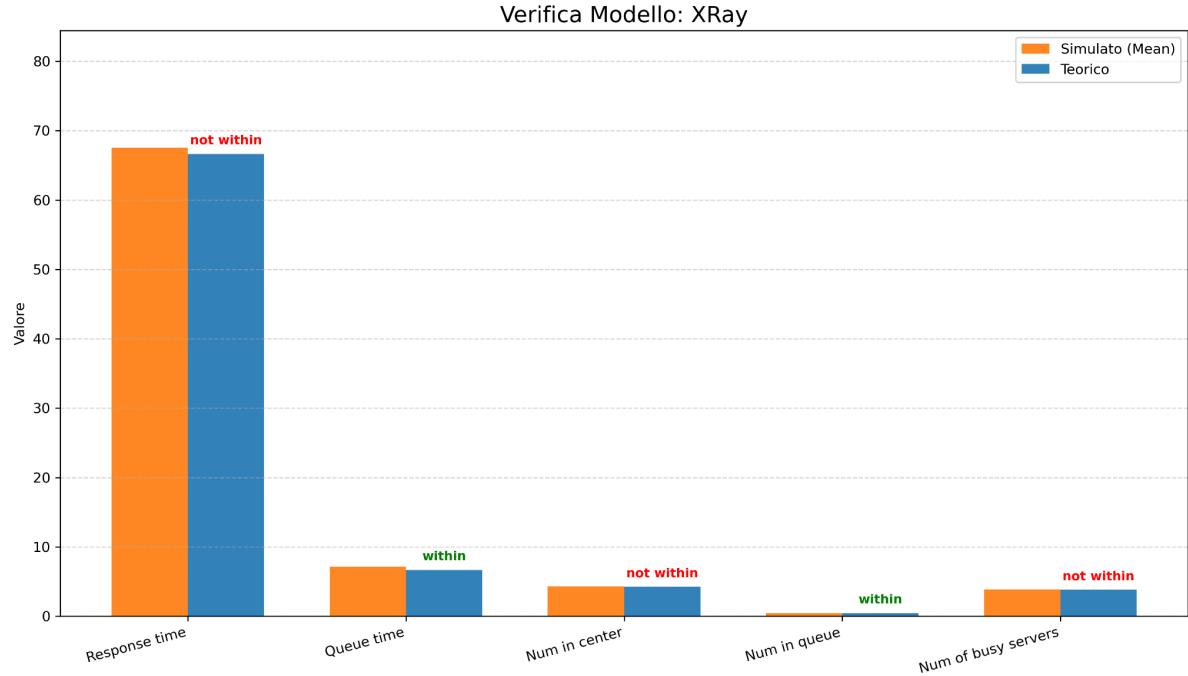
$$E(S_{i,3}) = \frac{1}{\mu_3} = 60s$$

$$E(T_{S,3}) = E(T_{Q,3}) + E(S_{i,3}) = 66,6452s$$

$$E(N_{Q,3}) = \lambda_{ver} \cdot E(T_{Q,3}) = 0,4227$$

$$E(N_{S,3}) = \lambda_{ver} \cdot E(T_{S,3}) = 4,2388$$

$$m_3 \rho_3 = 3,8162$$



| Metrica | SimMean | SimMin | SimMax | SimWidth | TheoValue | WithinInt |
|--------------------|----------|----------|----------|------------|-----------|------------|
| $E(T_{S,3})$ | 67,5508s | 66,7069s | 68,3947s | 0,84394727 | 66,6452s | not within |
| $E(T_{Q,3})$ | 7,1247s | 6,5214s | 7,7281s | 0,60333171 | 6,6452s | within |
| $E(N_{S,3})$ | 4,3092 | 4,2433 | 4,3751 | 0,06589774 | 4,2388 | not within |
| $E(N_{Q,3})$ | 0,4561 | 0,4166 | 0,4957 | 0,03958931 | 0,4227 | within |
| $m_3 \cdot \rho_3$ | 3,8530 | 3,8187 | 3,8874 | 0,03434691 | 3,8162 | not within |

6.5 - Verifica del centro trace detection M/M/1

Utilizziamo le formule per un centro M/M/1:

$$\rho_4 = \frac{\lambda_{ver} \cdot p_{check}}{\mu_4} = 0.3816$$

$$E(T_{Q,4}) = \frac{\rho_4}{\mu_4 - \lambda_{ver} \cdot p_{check}} = 37,0269s$$

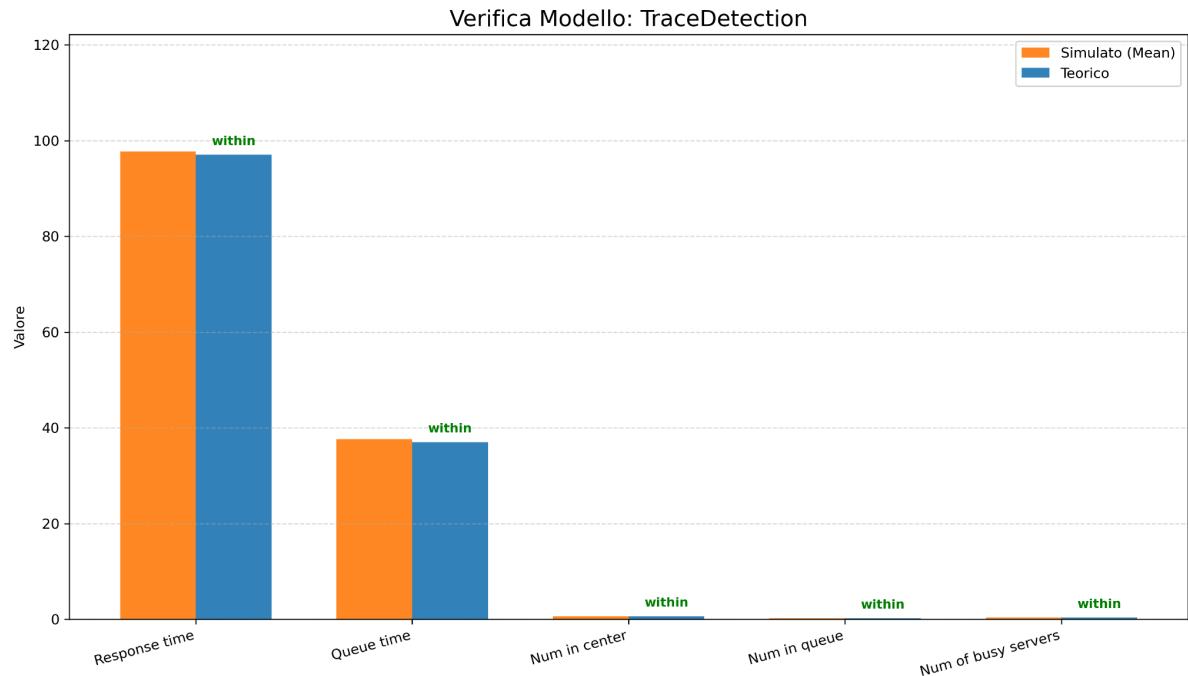
$$E(S_4) = \frac{1}{\mu_4} = 60s$$

$$E(T_{S,4}) = E(T_{Q,4}) + E(S_4) = 97,0269s$$

$$E(N_{Q,4}) = \lambda_{ver} \cdot p_{check} \cdot E(T_{Q,4}) = 0,2355$$

$$E(N_{S,4}) = \lambda_{ver} \cdot p_{check} \cdot E(T_{S,4}) = 0,6171$$

$$m_4 \cdot \rho_4 = \rho_4 = 0,3816$$



| Metrica | SimMean | SimMin | SimMax | SimWidth | TheoValue | WithinInt |
|--------------------|----------|----------|----------|------------|-----------|-----------|
| $E(T_{S,4})$ | 97,7466s | 96,4389s | 99,0542s | 1,30760265 | 97,0269s | within |
| $E(T_{Q,4})$ | 37,6507s | 36,5917s | 38,7096s | 1,05891882 | 37,0269s | within |
| $E(N_{S,4})$ | 0,6247 | 0,6145 | 0,6348 | 0,01015174 | 0,6171 | within |
| $E(N_{Q,4})$ | 0,2408 | 0,2334 | 0,2482 | 0,00739381 | 0,2355 | within |
| $m_4 \cdot \rho_4$ | 0,3839 | 0,3803 | 0,3875 | 0,00361438 | 0,3816 | within |

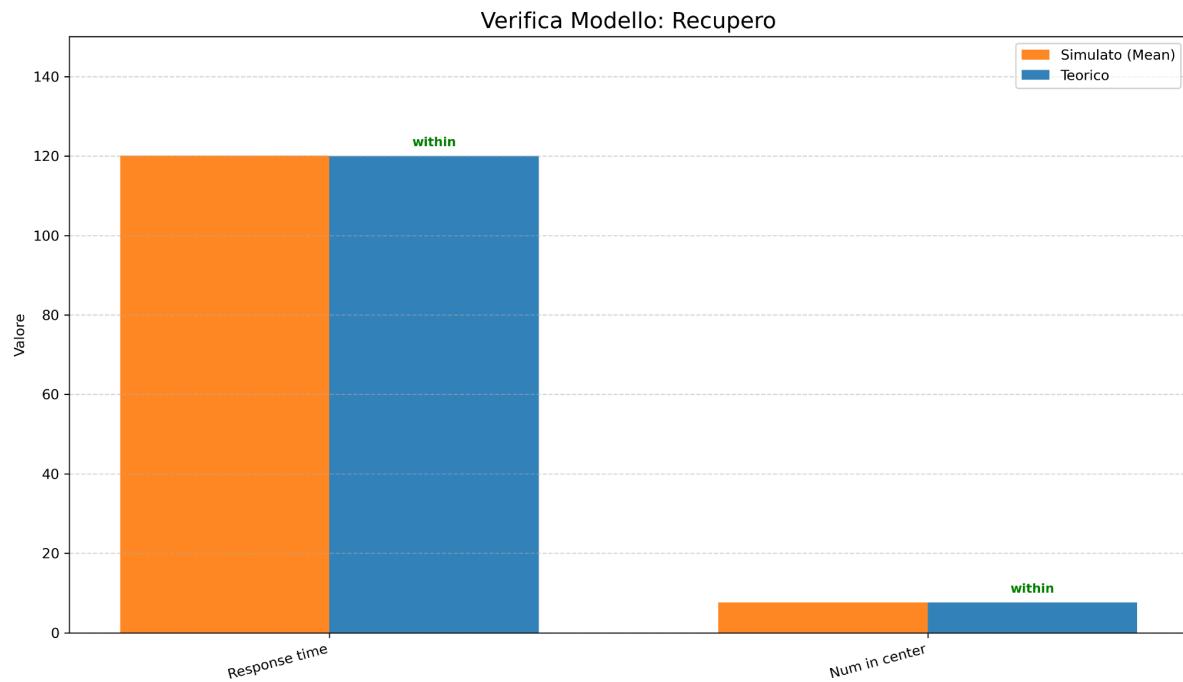
6.6 - Verifica del centro recupero M/M/∞

Utilizziamo le formule per un centro M/M/∞:

$$E(S_5) = \frac{1}{\mu_5} = 120s$$

$$E(T_{S,5}) = E(S_5) = 120s$$

$$E(N_{S,5}) = (\lambda_{ver} \cdot p_{standard} + \lambda_{ver} \cdot p_{check} \cdot p_{success}) \cdot E(T_{S,5}) = 7,6322$$



| Metrica | SimMean | SimMin | SimMax | SimWidth | TheoValue | WithinInt |
|--------------|-----------|-----------|-----------|------------|-----------|-----------|
| $E(T_{S,5})$ | 120,0579s | 119,3489s | 120,7669s | 0,70896407 | 120s | within |
| $E(N_{S,5})$ | 7,6556 | 7,5880 | 7,7231 | 0,06756742 | 7,6322 | within |

6.7 - Controlli di consistenza

I seguenti controlli di consistenza vengono rispettati:

$$E(T_{S,k}) = E(T_{Q,k}) + E(S_{i,k})$$

$$E(N_{S,k}) = E(N_{Q,k}) + m_k \cdot \rho_k$$

$$0 < \rho < 1$$

\forall centro $k = 1, 2, 3, 4, 5$

7 - Validazione

In questa fase ci assicuriamo che il modello computazionale sia conforme e consistente con il caso di studio preso in considerazione (Aeroporto di Ciampino). Per fare ciò, si confrontano le metriche ottenute dalle varie simulazioni (che utilizzano le distribuzioni effettive **Normale Troncata**) con le probabilità e i valori attesi, definiti nel modello di specifica grazie alle informazioni trovate in rete e precedentemente presentate.

7.1 - Verifica delle Probabilità di Instradamento

Per prima cosa controlliamo che la ripartizione dei passeggeri tra i vari percorsi (Check-in vs Accesso Diretto e Controlli Standard vs Approfonditi) sia coerente con quella osservata nei report reperiti online. Nello specifico, abbiamo stimato che circa il **38,72%** (p_{desk}) dei passeggeri si reca ai banchi accettazione, e che il **10%** (p_{check}) viene selezionato per controlli approfonditi. Infatti, queste percentuali sono confermate dai log della simulazione (*Figura 7a*), in cui, su un totale di **8.168** passeggeri processati nella run di riferimento, **3.181** hanno effettuato il passaggio al Centro 1 (Banchi Accettazione); mentre, su **8.168** passeggeri transitati ai raggi X, **838** sono stati instradati al Trace Detection:

```
Totali Passeggeri Processati (OUT): 8168
-----
>>> CHECK 1: Probabilità Check-In (Target: ~38.7%)
    Transiti Check-In: 3181
    Percentuale Simulata: 38,9447%
    
>>> CHECK 2: Probabilità Trace Detection (Target: ~10.0%)
    Transiti X-Ray: 8168
    Transiti Trace Det.: 838
    Percentuale Simulata: 10,2595%
```

Figura 7a

7.2 - Verifica dei Volumi di Traffico

Un altro check effettuato è quello della consistenza del generatore di arrivi rispetto ai parametri di input. Nello scenario della precedente figura (*Figura 7a*) abbiamo simulato utilizzando il valore $\lambda_{med} = 0,127205 \text{ pax / s}$ definito nel modello di specifica (*Capitolo 4.1*), su una finestra di $T = 18 \text{ h} = 64800 \text{ s}$ (durata giornata lavorativa aeroporto). Dunque, il volume di traffico atteso è:

$$N_{atteso} = \lambda_{med} \cdot T = 8242 \text{ passeggeri}$$

Il risultato della simulazione riporta un totale di **8.168** passeggeri processati, cioè lo scarto è di 74 passeggeri, pari a un errore relativo dello **0,9%**. Tale scostamento è accettato per via della natura stocastica del processo degli arrivi, ovvero un processo di Poisson, in cui la varianza coincide con il valore atteso. Di conseguenza, la deviazione standard attesa è di $\sigma = \sqrt{N_{atteso}} \simeq 91$ passeggeri, cioè siamo perfettamente entro i limiti ($74 < 91$).

7.3 - Verifica dei Parametri di Servizio

Un ulteriore livello di validazione consiste nel verificare se il motore di simulazione genera dei tempi di servizio che siano in accordo con le distribuzioni teoriche presentate nel modello di specifica (*Capitolo 4.4*). Di seguito si confrontano i tempi medi di servizio attesi con quelli effettivamente misurati durante la simulazione ad orizzonte finito:

| Centro di Servizio | Distribuzione Input | Media Teorica (E[S]th) | Media Simulata (E[S]sim) | Scostamento |
|---------------------------|---------------------|------------------------|--------------------------|-------------|
| Banchi Check-in | Norm. Troncata | 120,00 s | 119,6297 s | - 0,31% |
| Varchi Elettronici | Norm. Troncata | 15,00 s | 18,8157 s | +25,44% |
| Controlli Raggi X | Norm. Troncata | 60,00 s | 51,0605 s | -14,90% |
| Trace Detection | Norm. Troncata | 60,00 s | 59,8300 s | -0,28% |
| Recupero Oggetti | Norm. Troncata | 120,00 s | 119,8666 s | -0,11% |

Tabella 7b

Come si evince dalla *Tabella 7.2*, per i centri **Banchi Check-in**, **Trace Detection** e **Recupero Oggetti**, i tempi medi simulati coincidono quasi perfettamente con i valori teorici (scostamenti inferiori allo 0,5%), confermando la bontà del generatore di numeri casuali.

Si osservano invece scostamenti significativi, per gli altri due centri, ma con delle motivazioni statistiche:

1. **Varchi Elettronici (+25,44%)**: Il valore simulato (18,82s) è superiore alla media teorica (15s). Questo accade perché la deviazione standard impostata è molto ampia ($\sigma=15$) rispetto alla media, e il Lower Bound (LB=10) è molto vicino alla media, quindi tale taglio a sinistra sposta il valore medio della distribuzione verso destra (valori più alti).
2. **Controlli Raggi X (-14,90%)**: Qui osserviamo il fenomeno opposto: la media simulata (51,06s) è inferiore a quella teorica (60s), stavolta a causa dell'Upper Bound vicino alla media.

7.4 - Valutazione Macroscopica (Legge di Little)

Infine, applichiamo la Legge di Little ($N = \lambda T$) considerando l'intero aeroporto come una *black box* in condizione stabile, dunque, a tal fine, consideriamo i dati ottenuti dallo scenario di un'esecuzione steady state con carico dimezzato ($\lambda_{med} / 2$). Tale necessità, ricordiamo, è dovuta al fatto che, se utilizziamo il valore λ_{med} , il sistema non converge mai ad una condizione di stazionarietà, in quanto l'utilizzazione degli XRay è maggiore di 1 ($\rho > 1$).

| | | | | | |
|-------------------|--|----------------|------------|--------------|-----------------------|
| Nq_CheckIn | | 0,0046 +/- | 0,0006 [| 0,0040 ... | 0,0052] AC: 0,002 |
| Nq_Recupero | | 0,0000 +/- | 0,0000 [| 0,0000 ... | 0,0000] AC: 0,000 |
| Nq_TraceDetection | | 0,1232 +/- | 0,0027 [| 0,1205 ... | 0,1259] AC: -0,016 |
| Nq_Varchi | | 0,0102 +/- | 0,0007 [| 0,0096 ... | 0,0109] AC: 0,112 |
| Nq_XRay | | 0,0947 +/- | 0,0071 [| 0,0876 ... | 0,1017] AC: -0,032 |
| Ns_CheckIn | | 2,9591 +/- | 0,0179 [| 2,9412 ... | 2,9770] AC: -0,058 |
| Ns_Recupero | | 7,6458 +/- | 0,0501 [| 7,5957 ... | 7,6958] AC: 0,089 |
| Ns_TraceDetection | | 0,5053 +/- | 0,0047 [| 0,5006 ... | 0,5101] AC: 0,006 |
| Ns_Varchi | | 1,2103 +/- | 0,0082 [| 1,2021 ... | 1,2186] AC: 0,098 |
| Ns_XRay | | 3,3526 +/- | 0,0259 [| 3,3267 ... | 3,3785] AC: 0,043 |
| S_CheckIn | | 119,7077 +/- | 0,2008 [| 119,5068 ... | 119,9085] AC: 0,048 |
| S_Recupero | | 119,9057 +/- | 0,1600 [| 119,7457 ... | 120,0658] AC: 0,080 |
| S_TraceDetection | | 59,8301 +/- | 0,0908 [| 59,7333 ... | 59,9269] AC: -0,006 |
| S_Varchi | | 18,8199 +/- | 0,0346 [| 18,7854 ... | 18,8545] AC: -0,068 |
| S_XRay | | 51,0937 +/- | 0,0722 [| 51,0215 ... | 51,1660] AC: -0,061 |
| Tq_CheckIn | | 0,1851 +/- | 0,0230 [| 0,1621 ... | 0,2081] AC: 0,006 |
| Tq_Recupero | | 0,0000 +/- | 0,0000 [| 0,0000 ... | 0,0000] AC: 0,000 |
| Tq_TraceDetection | | 19,2603 +/- | 0,3571 [| 18,9032 ... | 19,6173] AC: -0,023 |
| Tq_Varchi | | 0,1601 +/- | 0,0104 [| 0,1498 ... | 0,1705] AC: 0,108 |
| Tq_XRay | | 1,4763 +/- | 0,1038 [| 1,3725 ... | 1,5801] AC: -0,029 |
| Ts_CheckIn | | 119,8927 +/- | 0,2020 [| 119,6908 ... | 120,0947] AC: 0,058 |
| Ts_Recupero | | 119,9057 +/- | 0,1600 [| 119,7457 ... | 120,0658] AC: 0,080 |
| Ts_TraceDetection | | 79,0906 +/- | 0,3731 [| 78,7175 ... | 79,4637] AC: 0,004 |
| Ts_Varchi | | 18,9801 +/- | 0,0371 [| 18,9430 ... | 19,0172] AC: -0,009 |
| Ts_XRay | | 52,5701 +/- | 0,1379 [| 52,4322 ... | 52,7080] AC: -0,028 |
| X_CheckIn | | 2,9545 +/- | 0,0177 [| 2,9368 ... | 2,9723] AC: -0,052 |
| X_Recupero | | 7,6458 +/- | 0,0501 [| 7,5957 ... | 7,6958] AC: 0,089 |
| X_TraceDetection | | 0,3822 +/- | 0,0026 [| 0,3796 ... | 0,3847] AC: 0,029 |
| X_Varchi | | 1,2001 +/- | 0,0079 [| 1,1922 ... | 1,2080] AC: 0,084 |
| X_XRay | | 3,2579 +/- | 0,0210 [| 3,2369 ... | 3,2790] AC: 0,060 |

Figura 7c

Dai risultati ottenuti (Figura 7c) osserviamo che:

- **Numero medio utenti nel sistema ($E[Ns]$):** Calcolato come somma delle popolazioni medie di tutti i centri:

$$E[Ns] = 2,9591 + 1,2103 + 3,3526 + 0,5053 + 7,6458 = 15,6731 \text{ passeggeri}$$

- **Tempo medio di risposta ($E[Ts]$):** Calcolato come somma pesata dei tempi di soggiorno in base alle probabilità di routing:

$$E[Ts] = (119,8927 \cdot p_{desk}) + 18,9801 + 52,5701 + (79,0906 \cdot p_{check}) + 119,9057 \\ = 245,79 \text{ secondi}$$

Dunque:

$$E[N_s]_{teorico} = \frac{\lambda_{med}}{2} \cdot E[T_s] = \frac{0,127205 \text{ passeggeri/s}}{2} \cdot 245,79 \text{ s} = 15,6329 \text{ passeggeri}$$

Confrontando il valore simulato (15,6731) con quello teorico (15,6329), lo scostamento è irrilevante (errore < 0,26%), confermando la consistenza matematica del simulatore.

7.5 - Verifiche di Consistenza

A completamento della validazione, verifichiamo che, per ogni centro il tempo medio di risposta ($E[T_s]$) corrisponde alla somma tra il tempo medio di attesa in coda ($E[T_Q]$) e il tempo medio di servizio ($E[S]$):

$$E[T_s] = E[T_Q] + E[S]$$

A tal proposito utilizziamo sempre i risultati dello scenario SteadyState (*Figura 7c*) che confermano il rispetto della formula precedente. Riportiamo l'esempio di calcolo per il centro più delicato, ovvero quello dei controlli a raggi X. Dai log della simulazione:

- Tempo medio di servizio ($E[S]_{XRay}$): **51,0937 s**
- Tempo medio di attesa ($E[T_Q]_{XRay}$): **1,4763 s**
- Tempo medio di risposta ($E[T_s]_{XRay}$): **52,5701 s**

L'uguaglianza è verificata: **1,4763 s + 51,0937 s = 52,5700 s**.

In maniera analoga si può testare la seguente uguaglianza:

$$E[N_s] = E[N_Q] + m\rho$$

Anche qui dai log dello scenario SteadyState otteniamo conferme, in particolare, leggiamo per gli XRay:

- Numero medio server attivi ($E[X]_{XRay} = m_3 \rho_3$): **3,2579**
- Numero medio job in coda ($E[N_Q]_{XRay}$): **0,0947**
- Numero medio job nel centro, coda + servizio ($E[N_s]_{XRay}$): **3,3526**

L'uguaglianza è verificata: **0,0947 + 3,2579 = 3,3526**.

8 - Design degli esperimenti

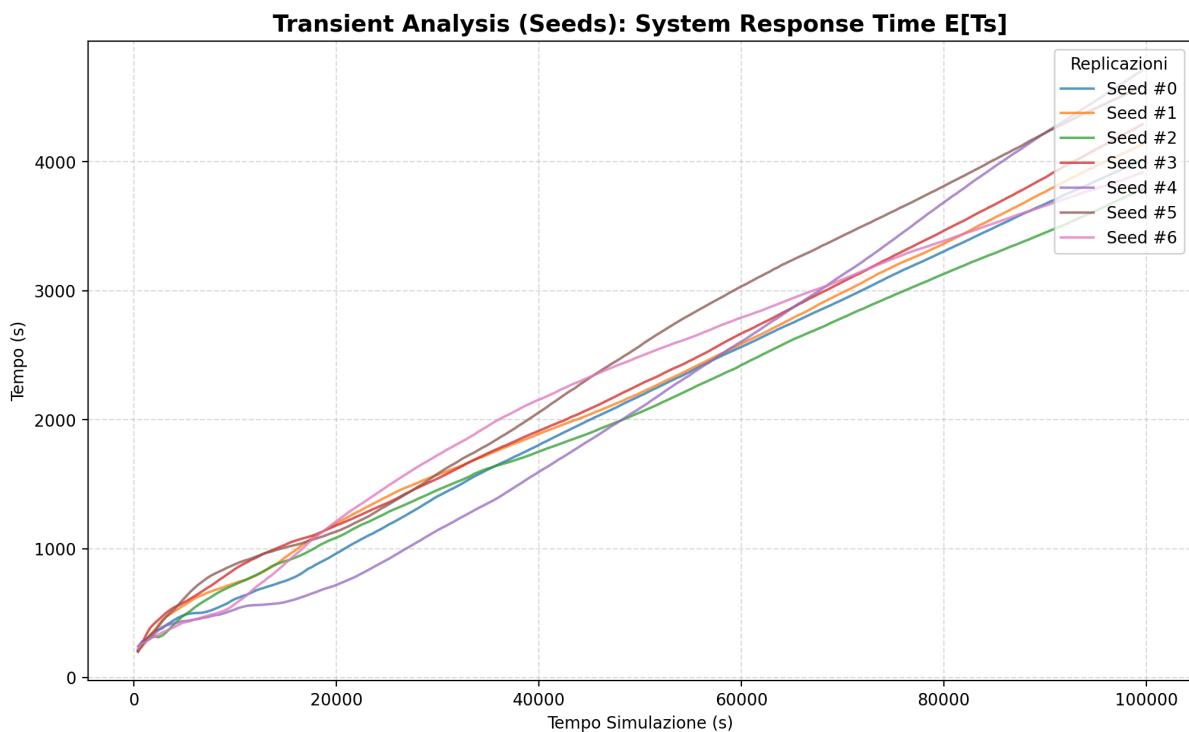
La fase di progettazione degli esperimenti ha l'obiettivo di configurare correttamente i parametri della simulazione (durata e warm-up) e di analizzare i risultati ottenuti.

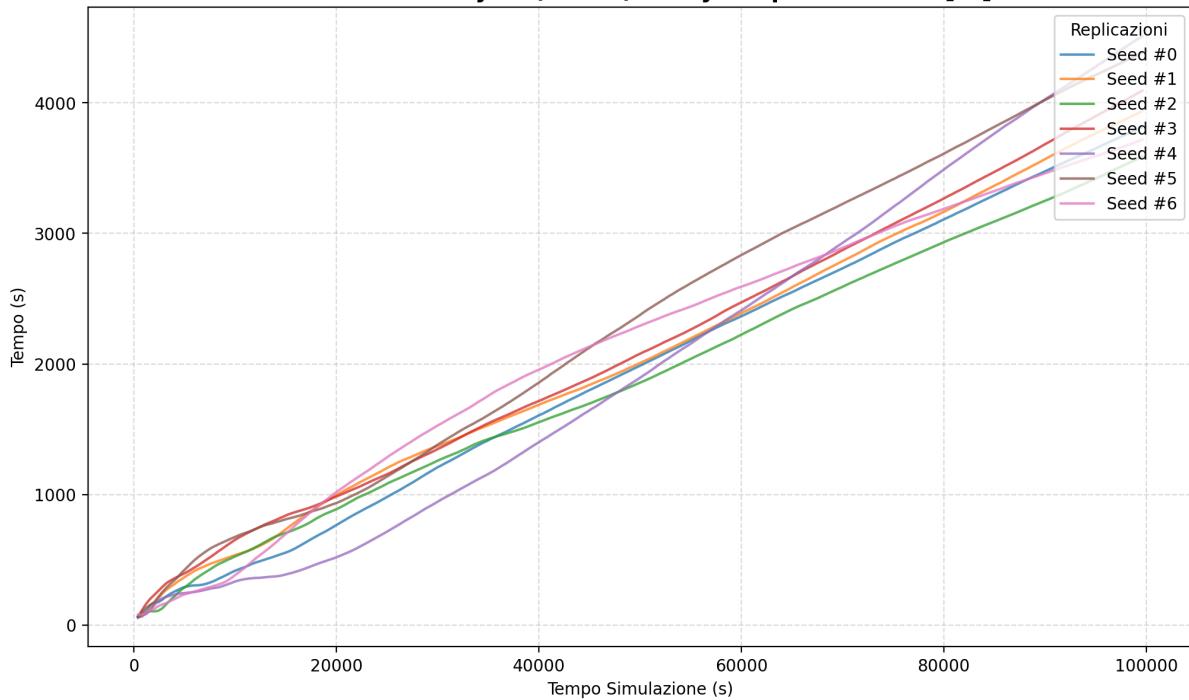
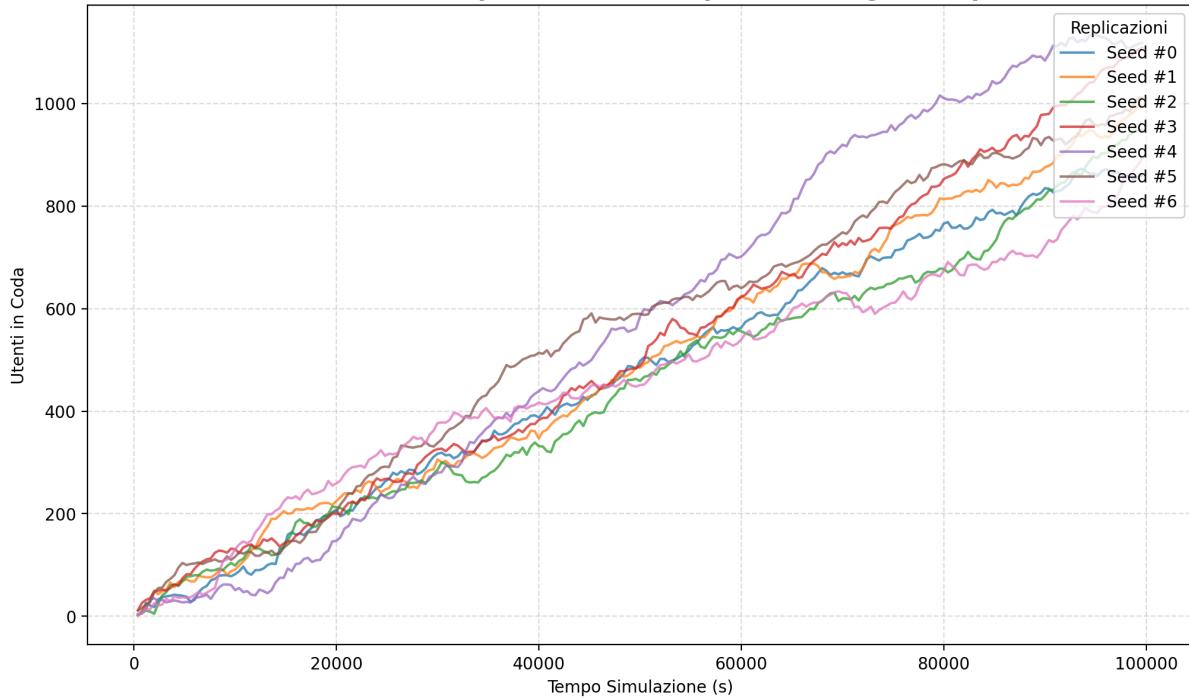
8.1 - Analisi transitorio

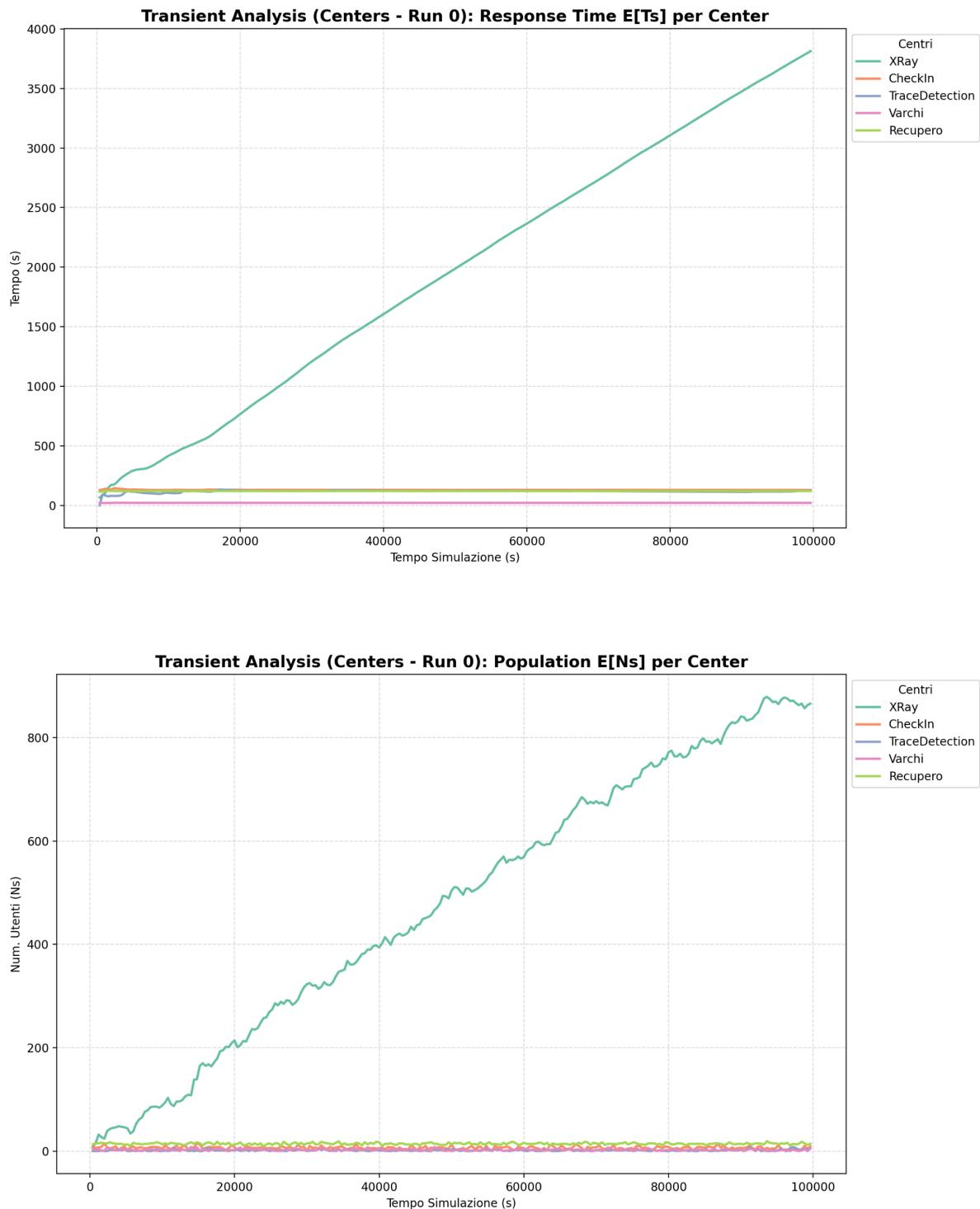
Per prima cosa siamo andati a studiare il transitorio iniziale per individuare il momento in cui il sistema raggiunge condizioni di regime stazionario o, eventualmente, identificare condizioni di instabilità.

8.1.1 - Analisi con carico effettivo (λ_{med})

Il primo tentativo effettuato è stato quello di analizzare il comportamento del sistema sottoposto al carico di lavoro stimato nel *Capitolo 4* ($\lambda_{med} = 0,127205 \text{ pax / s}$). L'analisi dell'andamento del tempo medio di risposta del sistema ($E[T_{sys}]$), del tempo medio di risposta del centro a raggi X ($E[T_{S_{XRay}}]$) e del numero medio di job in coda nel centro a raggi X ($E[N_{S_{XRay}}]$), identificato già nel modello concettuale come candidato principale per il collo di bottiglia del sistema, su una run molto lunga (*TRANSIENT_DURATION* = 100000.0) mostra che il sistema non raggiunge mai una condizione di stazionarietà:



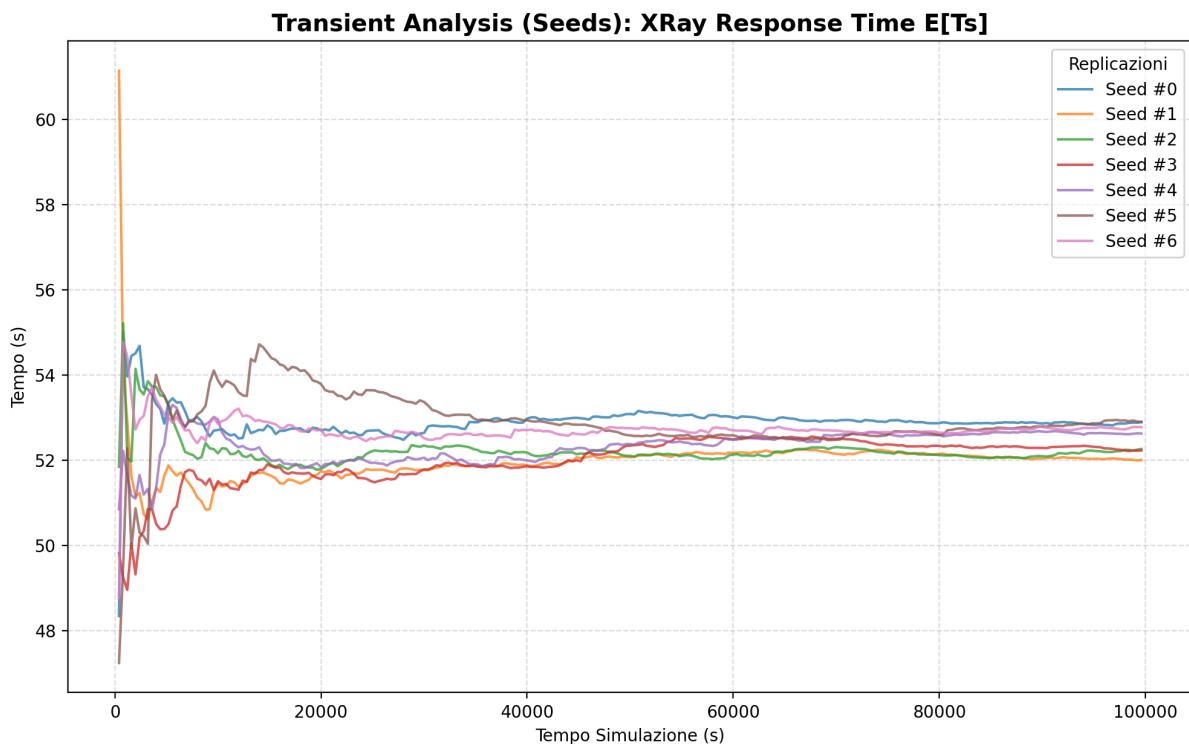
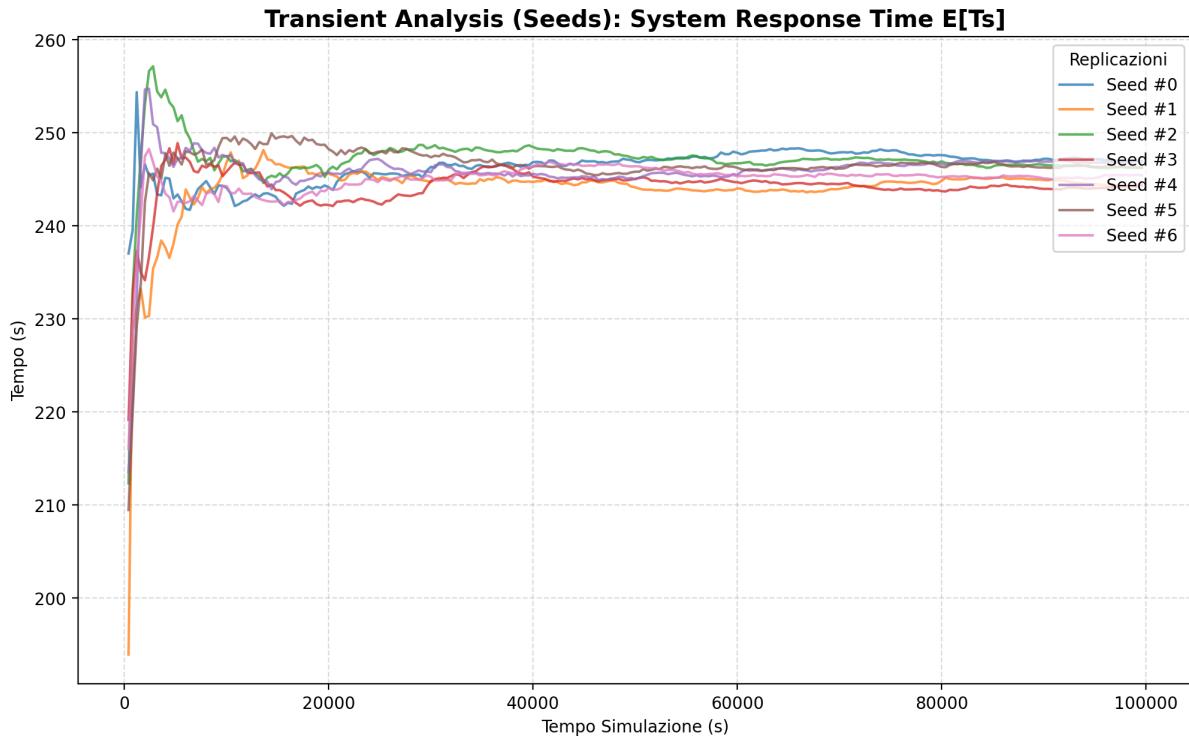
Transient Analysis (Seeds): XRay Response Time E[Ts]**Transient Analysis (Seeds): XRay Queue Length E[Nq]**

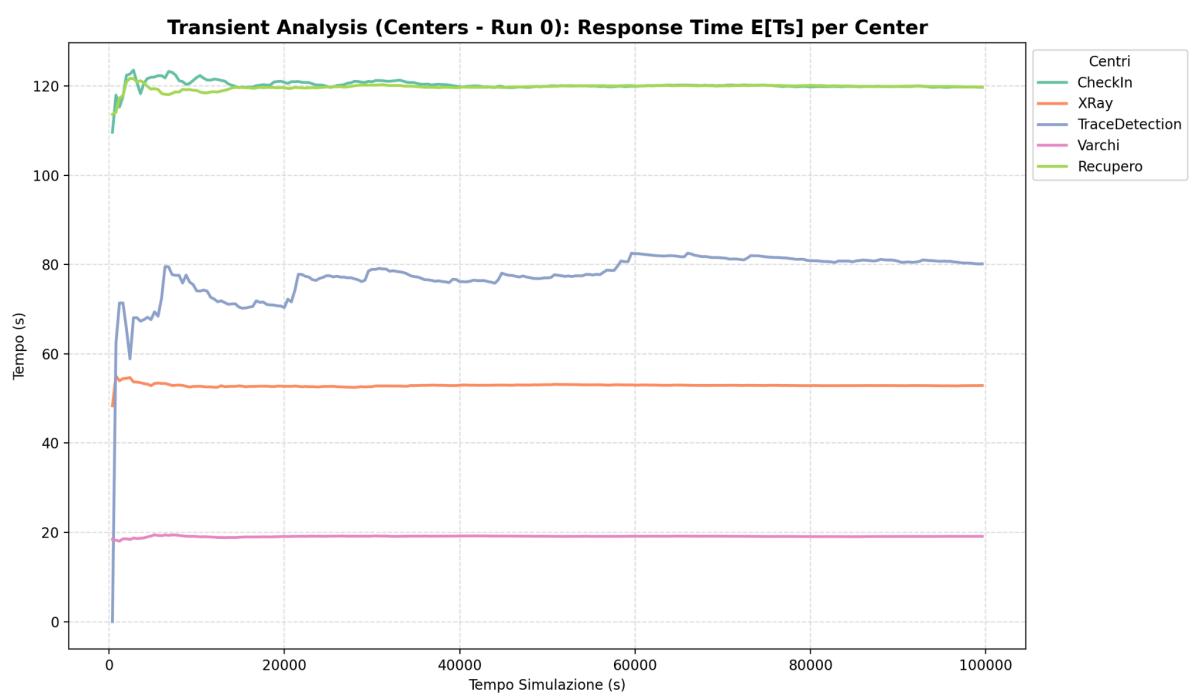
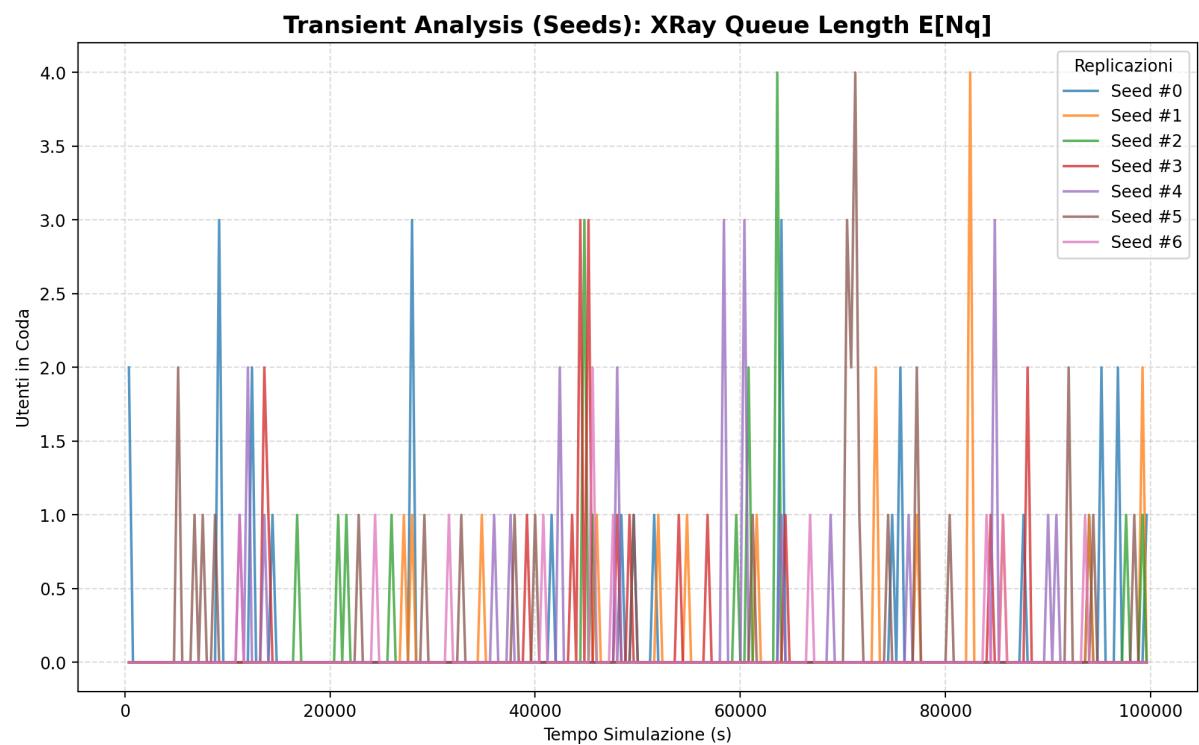


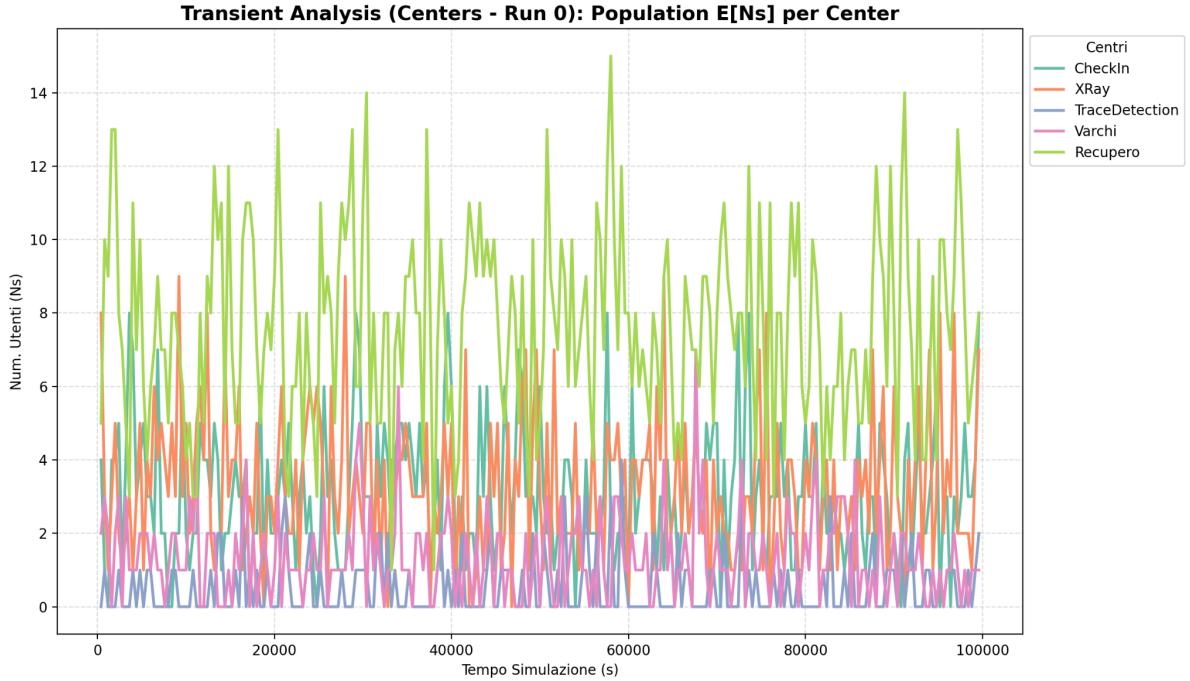
Come si evince dalle figure, la curva cresce linearmente senza stabilizzarsi. Questo indica che, con l'attuale configurazione delle risorse, il sistema opera in condizioni di saturazione ($\rho \approx 1$) nei picchi.

8.1.2 - Determinazione del warm-up a carico ridotto

Per poter comunque verificare e validare matematicamente il modello (come fatto nei *Capitoli 6 e 7*) e identificare un periodo di *warm-up* (T_{cut}) utile per le analisi statistiche, abbiamo ripetuto l'analisi del transitorio riducendo il carico al 50%, osservando i seguenti grafici:







In queste condizioni, osserviamo che il sistema converge a uno stato stazionario dopo circa **60000 secondi** (la coda negli XRay rimane sempre abbastanza stabile entro un determinato range). Di conseguenza, per le simulazioni SteadyState e per la Verification è stato fissato un tempo di taglio **TIME_WARMUP = 60000**, scartando i dati raccolti prima di tale soglia per rimuovere il bias iniziale.

8.2 - Simulazione ad orizzonte finito

8.2.1 - Analisi del collo di bottiglia

Prima di procedere con la simulazione, effettuiamo un'analisi operazionale statica per identificare quale centro costituisce il collo di bottiglia del sistema. Definiamo la **domanda di servizio scalata per server**, come:

$$\frac{D_i}{m_i} = \frac{V_i \cdot E[S_i]}{m_i}$$

In particolare, per un centro con m_i server identici, l'utilizzazione è definita come il prodotto tra il traffico in ingresso al centro λ_i e la capacità totale del centro $E[S] = \frac{E[S_i]}{m_i}$, quindi, affinché il centro non diventi saturo, c'è bisogno che:

$$\frac{\lambda_i \cdot E[S_i]}{m_i} < 1$$

A questo punto il tasso di arrivo locale al centro (λ_i) dipende dal tasso totale di arrivi all'aeroporto (λ) moltiplicato per il numero di visite (V_i) in quel centro:

$$\frac{\lambda V_i \cdot E[S_i]}{m_i} < 1$$

$$\frac{D_i}{m_i} < \frac{1}{\lambda}$$

Dunque, le domande di servizio per server devono essere minori del tempo di inter-arrivo, pari a:

$$\frac{1}{\lambda_{med}} = 7,86 \text{ s}$$

Ora dal Capitolo 4, richiamiamo le seguenti configurazioni scelte:

- **Banchi Accettazione:** $m_1 = 8$, $E[S_1] = 120 \text{ s}$
- **Varchi Elettronici:** $m_2 = 4$, $E[S_2] = 15 \text{ s}$
- **Controllo X-Ray:** $m_3 = 6$, $E[S_3] = 60 \text{ s}$
- **Trace Detection:** $m_4 = 1$, $E[S_4] = 60 \text{ s}$

In base alle equazioni di traffico, definite sempre nel *Capitolo 4*, abbiamo:

- $V_1 = \frac{\lambda_2}{\lambda} = \frac{\lambda \cdot p_{desk}}{\lambda} = p_{desk} = 0,3872$
- $V_2 = \frac{\lambda_5}{\lambda} = \frac{\lambda_2 + \lambda_3}{\lambda} = \frac{\lambda \cdot p_{desk} + \lambda \cdot p_{direct}}{\lambda} = 1$
- $V_3 = \frac{\lambda_6}{\lambda} = \frac{\lambda_2 + \lambda_3}{\lambda} = \frac{\lambda \cdot p_{desk} + \lambda \cdot p_{direct}}{\lambda} = 1$
- $V_4 = \frac{\lambda_9}{\lambda} = \frac{\lambda_7 \cdot p_{success}}{\lambda} = \frac{(\lambda_6 \cdot p_{check}) \cdot p_{success}}{\lambda} = p_{check} \cdot p_{success} \approx 0,1$

Quindi:

- **Banchi Accettazione:** $\frac{0,3872 \cdot 120}{8} \text{ s} \approx 5,81 \text{ s}$
- **Varchi Elettronici:** $\frac{1 \cdot 15}{4} \text{ s} \approx 3,75 \text{ s}$
- **Controllo X-Ray:** $\frac{1 \cdot 60}{6} \text{ s} \approx 10,00 \text{ s}$
- **Trace Detection:** $\frac{0,1 \cdot 60}{1} \text{ s} \approx 6,00 \text{ s}$
- **Recupero Oggetti:** è modellato come Infinite Server, pertanto il suo carico per server tende a 0 e non può fisicamente rappresentare un collo di bottiglia.

Confrontando i valori, notiamo che il **Centro 3 (Controlli Raggi X)** presenta il carico per server più elevato (10 s), superiore al tempo di inter-arrivo (7,86 s), identificandosi come il collo di bottiglia principale del sistema. Anche il Trace Detection presenta un carico significativo (6,00 s), seppur sotto il limite di stabilità, e probabilmente, anche nella fase di simulazione non presenterà problemi a causa della saturazione dei Raggi X, che limita il flusso in arrivo al Trace Detection.

8.2.2 - Risultati della simulazione operativa

Confermata l'esistenza teorica di un collo di bottiglia, si è proceduto con la simulazione utilizzando un **Orizzonte Finito** di 18 ore (*WORK_DAY* = 64800), corrispondente all'effettiva giornata lavorativa dell'aeroporto, applicando il carico $\lambda_{med} = 0,127205$ pax/s.

Per ottenere risultati statisticamente rilevanti è stato utilizzato il **metodo delle Replicazioni Indipendenti**. Sono state eseguite **64 replicazioni** ($n=64$) con semi (seed) diversi per il generatore di numeri casuali, garantendo che le osservazioni campionarie (le medie di ogni run) fossero indipendenti e identicamente distribuite (i.i.d.).

L'analisi dell'errore è stata condotta calcolando gli **Intervalli di Confidenza (IC) al 95%**. Il calcolo, implementato nella classe `IntervalEstimation`, si basa sulla distribuzione **t di Student**. L'ampiezza dell'intervallo (w) è calcolata secondo la formula:

$$w = t_{n-1, 1-\frac{\alpha}{2}} \cdot \frac{S}{\sqrt{n}}$$

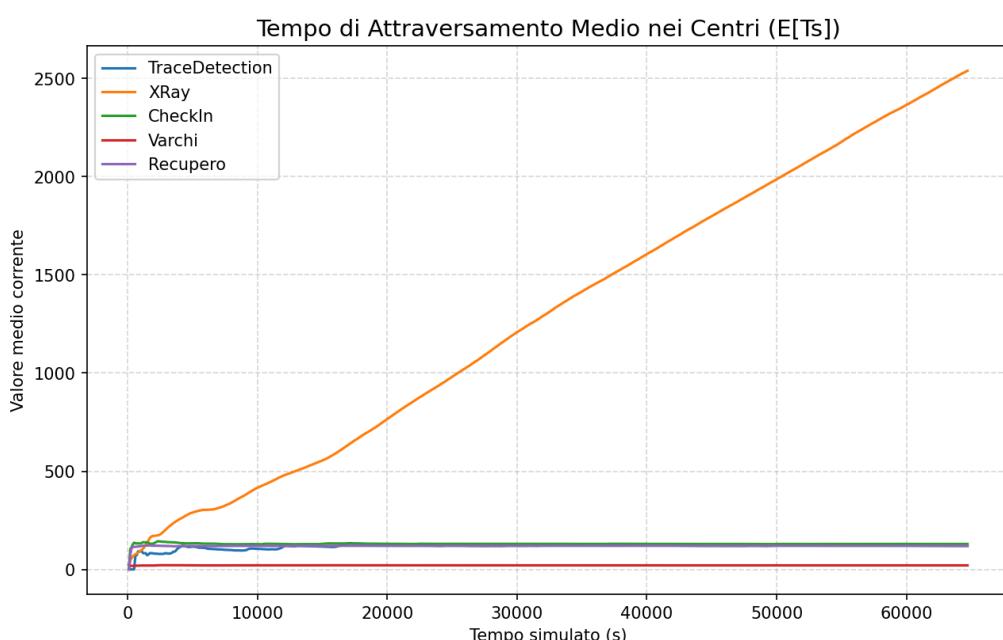
dove:

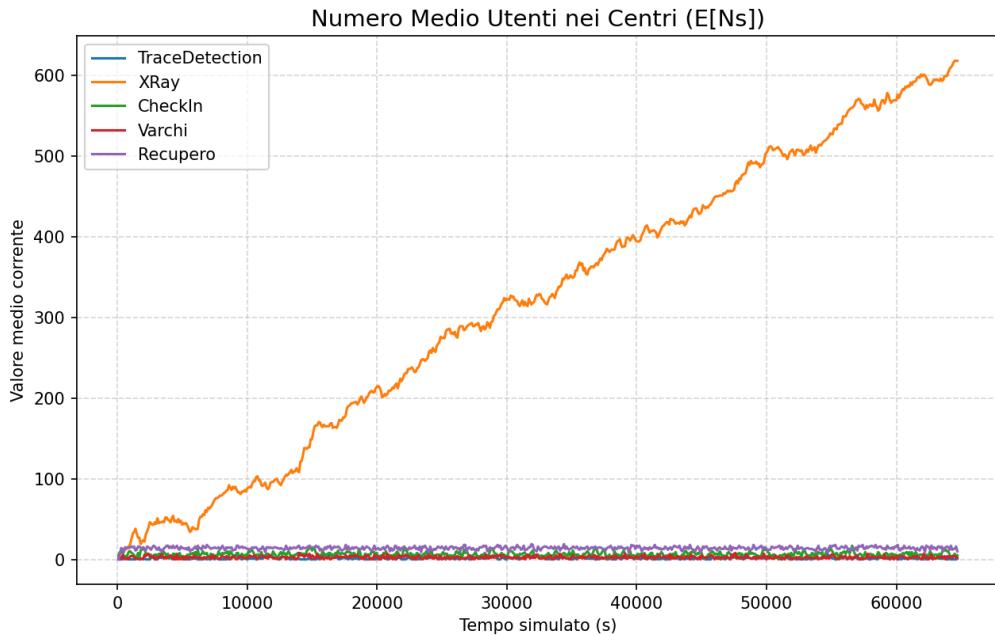
- n è la dimensione del campione (numero di replicazioni);
- S è la deviazione standard campionaria;
- $t_{n-1, 1-\frac{\alpha}{2}}$ è il valore critico (quantile) della distribuzione **t di Student** con $n-1$ gradi di libertà (calcolato nel codice mediante la funzione `idfStudent()` della libreria `Rvms`).

Gli estremi dell'intervallo sono dunque:

$$\bar{x} \pm w$$

Di seguito vengono presentati i risultati ottenuti, riportando nelle tabelle, per ogni metrica, la media stimata e il relativo intervallo di confidenza:





Nello specifico, mentre i centri Check-in, Varchi e Trace Detection mostrano code trascurabili, il centro **Controllo X-Ray**, accumula una coda media di **318,61 passeggeri**, rappresentando un rischio sia per le prestazioni sia per la sicurezza, in quanto viola il requisito presentato nei capitoli precedenti (*Capitolo 2*), visto che il numero medio di job nel centro raggiunge il valore di circa **324,60**:

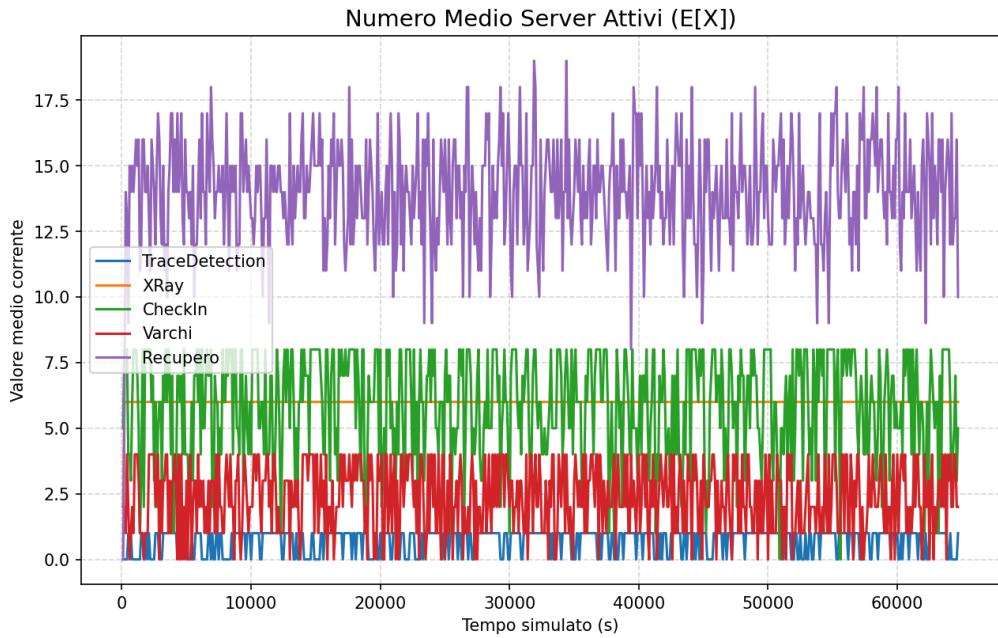
Numero Utenti in Coda (Nq) - Confidenza 95%

| Center | Mean | Width | Min | Max |
|-----------------------|------------|-----------|------------|------------|
| CheckIn | 0.511258 | 0.024614 | 0.486645 | 0.535872 |
| Recupero | 0.0 | 0.0 | 0.0 | 0.0 |
| TraceDetection | 0.8034 | 0.041767 | 0.761632 | 0.845167 |
| Varchi | 0.226247 | 0.004779 | 0.221468 | 0.231027 |
| XRay | 318.613026 | 12.165288 | 306.447738 | 330.778314 |

Numero Utenti nel Sistema (Ns) - Confidenza 95%

| Center | Mean | Width | Min | Max |
|-----------------------|------------|-----------|------------|------------|
| CheckIn | 6.384689 | 0.04882 | 6.335869 | 6.433509 |
| Recupero | 14.015745 | 0.011812 | 14.003933 | 14.027558 |
| TraceDetection | 1.507305 | 0.045729 | 1.461576 | 1.553034 |
| Varchi | 2.614096 | 0.01063 | 2.603467 | 2.624726 |
| XRay | 324.600013 | 12.165358 | 312.434655 | 336.765371 |

Interessante osservare anche il grafico riguardante il numero medio di server utilizzati, dove possiamo vedere che per il centro a raggi X sono costantemente in uso tutti e 6 i server:



Ciò impatta pesantemente sui tempi di risposta dei vari centri:

Tempo di Attesa in Coda (Tq) - Confidenza 95%

| Center | Mean | Width | Min | Max |
|-----------------------|-------------|------------|-------------|-------------|
| CheckIn | 10.390779 | 0.467931 | 9.922848 | 10.85871 |
| Recupero | 0.0 | 0.0 | 0.0 | 0.0 |
| TraceDetection | 68.077271 | 3.142358 | 64.934913 | 71.219629 |
| Varchi | 1.78194 | 0.034701 | 1.747238 | 1.816641 |
| XRay | 2717.398551 | 103.866805 | 2613.531746 | 2821.265356 |

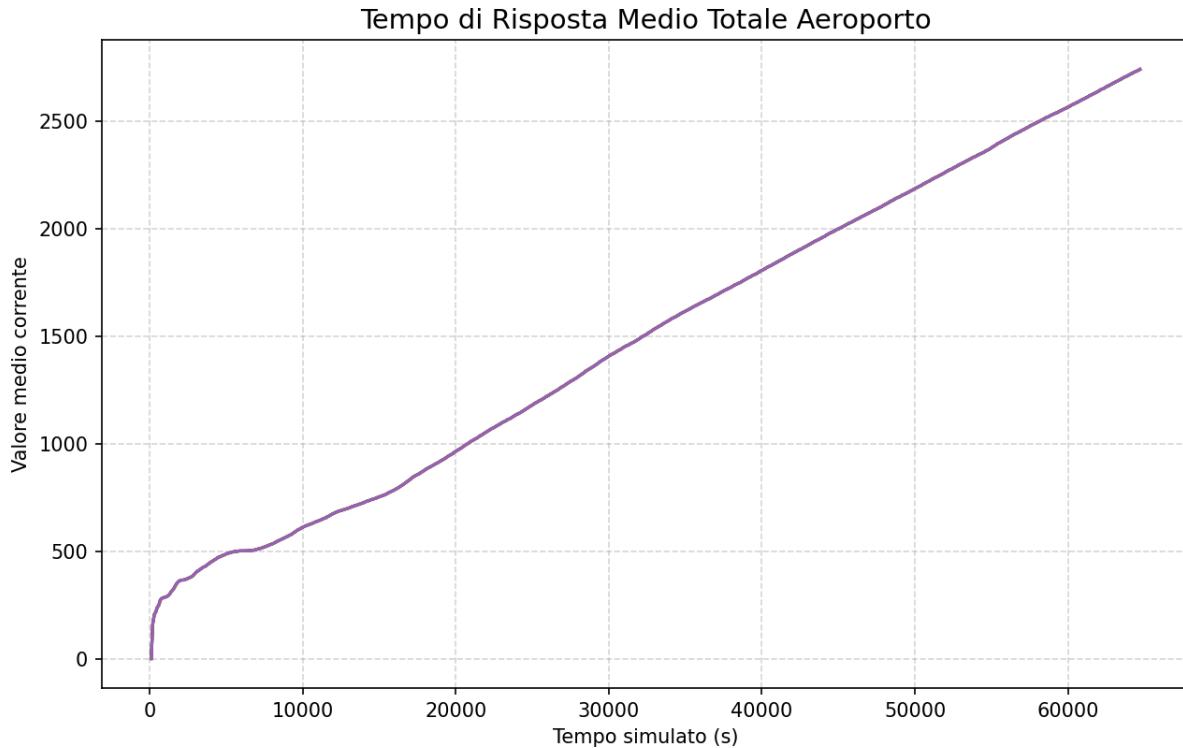
Tempo di Risposta (Wait+Svc) (Ts) - Confidenza 95%

| Center | Mean | Width | Min | Max |
|-----------------------|-------------|------------|-------------|-------------|
| CheckIn | 130.020525 | 0.506225 | 129.5143 | 130.52675 |
| Recupero | 119.866589 | 0.062239 | 119.80435 | 119.928828 |
| TraceDetection | 127.907289 | 3.173391 | 124.733897 | 131.08068 |
| Varchi | 20.59759 | 0.0428 | 20.55479 | 20.640391 |
| XRay | 2768.459076 | 103.870342 | 2664.588734 | 2872.329418 |

Di conseguenza, il tempo medio di attraversamento del sistema ammonta a circa **49,5 min**:

Tempo Totale Attraversamento (SystemResponseTime) - Confidenza 95%

| Center | Mean | Width | Min | Max |
|----------------|-------------|------------|-------------|-------------|
| Success | 2972.078868 | 103.961222 | 2868.117646 | 3076.040089 |



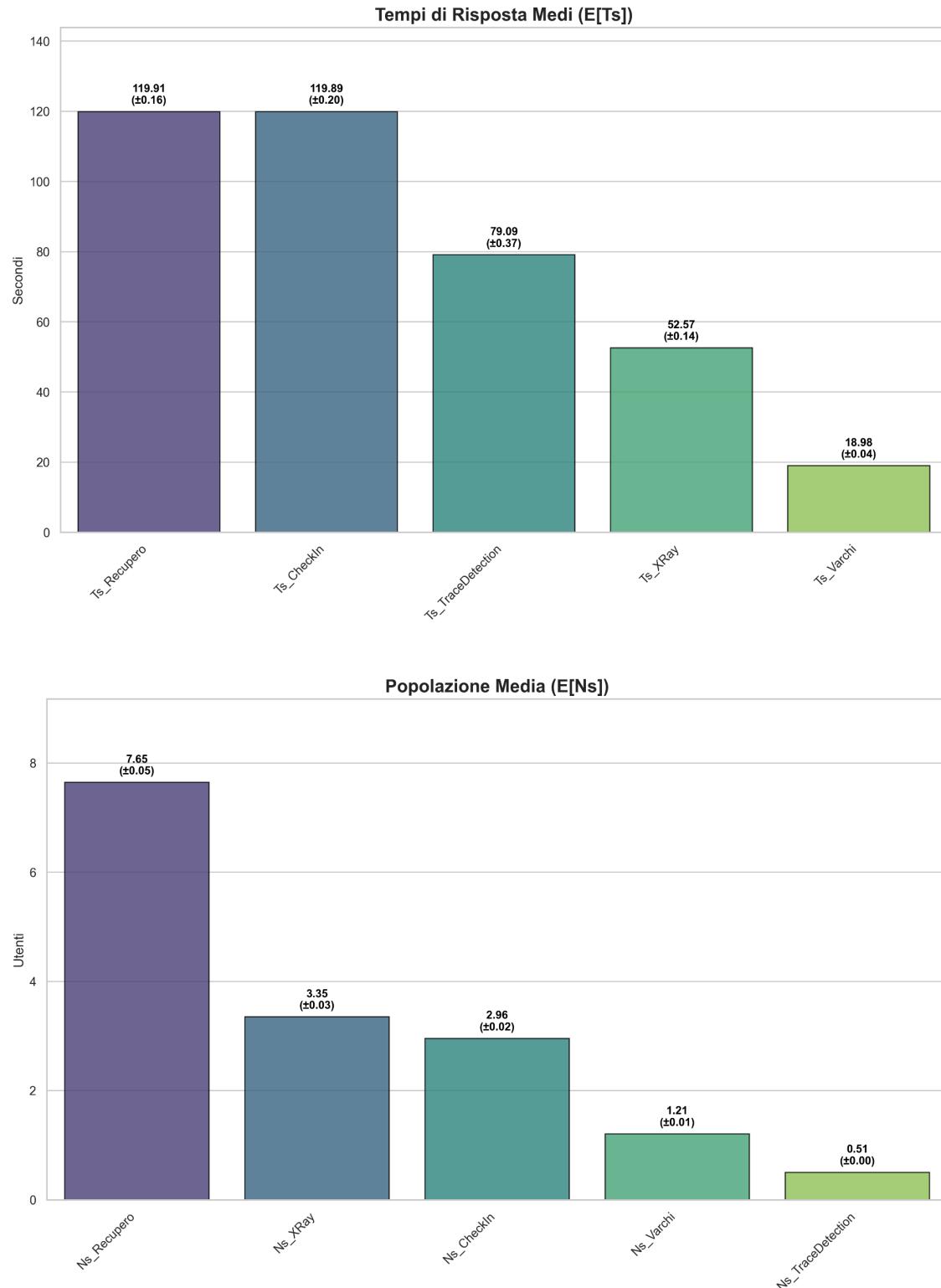
Seppur tale tempo rientra nei margini a disposizione del passeggero (considerando un arrivo in aeroporto 120 minuti prima del volo), questo valore è indice di una pessima gestione. Infatti, vengono spesi in media **45,5 minuti** ($2730,01 \text{ s} = p_{desk} E[T_Q]_{\text{Banchi}} + E[T_Q]_{\text{Varchi}} + E[T_Q]_{\text{XRay}} + p_{check} E[T_Q]_{\text{Trace}}$) nelle varie code, sui 49,5 min totali per completare l'intero percorso. Il fatto che quasi il **92% del tempo** di permanenza in aeroporto sia speso in coda ai controlli di sicurezza a raggi X (45,3 min su 49,5 min totali) conferma la necessità di interventi migliorativi su quest'ultimo centro.

8.3 - Simulazione ad orizzonte infinito

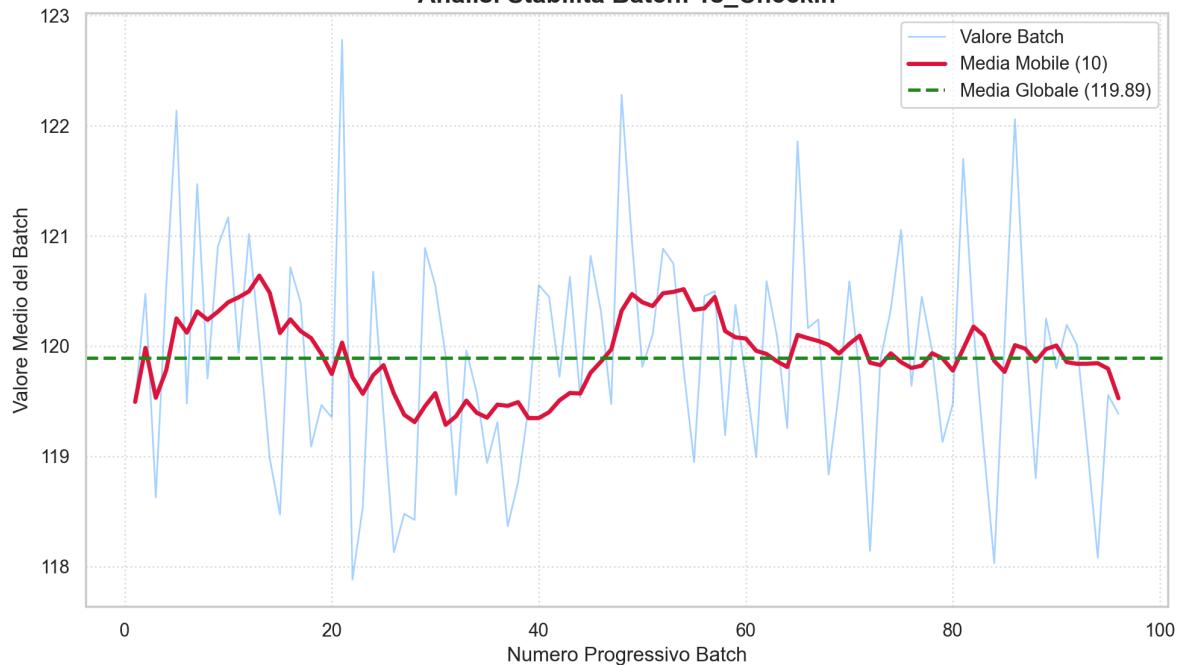
L'ultima parte del design degli esperimenti riguarda la simulazione a orizzonte infinito, ma è bene specificare che questa configurazione, implementata nella classe `SteadyStateRunner`, non è utilizzata per simulare le dinamiche operative dell'aeroporto, bensì è stata progettata ai fini di Verifica e Validazione (come spiegato nei capitoli precedenti). Dunque, in questa sezione, ci concentreremo sull'elencare le caratteristiche salienti del runner a Orizzonte Infinito. In particolare, per garantire la validità statistica dei risultati, è stato implementato il metodo delle **Batch Means**. Tale tecnica consiste nell'eseguire una singola simulazione molto lunga; dividere la sequenza dei dati generati in k blocchi consecutivi (batches), ciascuno di dimensione m ; calcolare la media del batch; e infine, trattare queste medie come se fossero osservazioni indipendenti del comportamento steady-state del sistema. La configurazione determinata è la seguente:

- **Lambda:** $\lambda_{med} / 2$
- **Tempo di Taglio (T_{cut}):** 60.000 secondi (warm-up per rimuovere il *bias* iniziale).
- **Dimensione Batch (b):** 1080 osservazioni per batch.
- **Numero di Batch (k):** 96 batch.

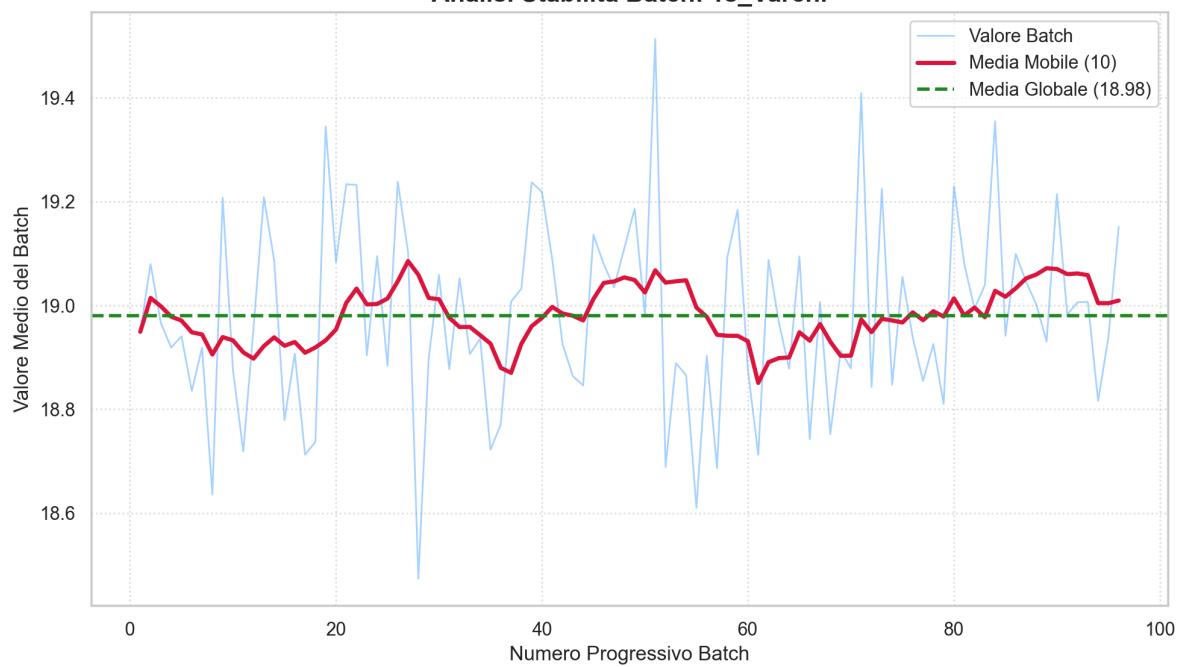
Tali parametri sono stati scelti misurando l'**autocorrelazione** tra le medie dei batch, la quale è risultata trascurabile ($< |0, 2|$ per tutte le metriche), confermando che i dati raccolti sono indipendenti e identicamente distribuiti e quindi adatti per la stima degli intervalli di confidenza. I risultati dello SteadyState con la frequenza degli arrivi dimezzata mostrano un sistema in grado di stabilizzarsi, confermando le osservazioni dell'analisi del transitorio:



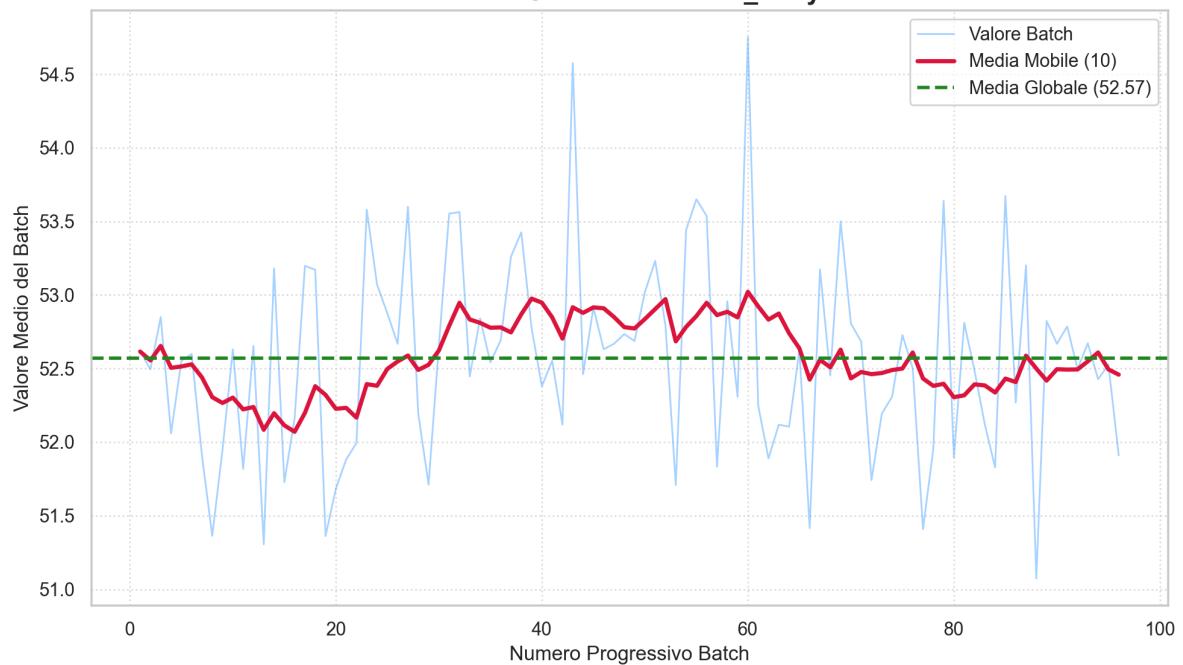
Analisi Stabilità Batch: Ts_CheckIn



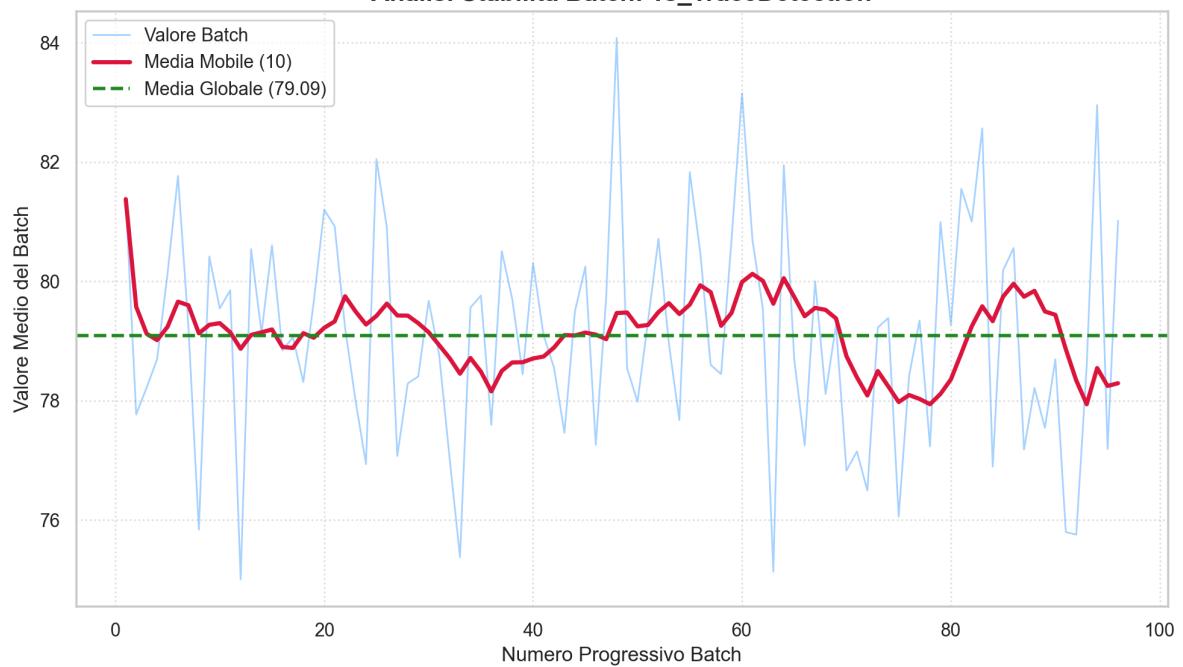
Analisi Stabilità Batch: Ts_Varchi



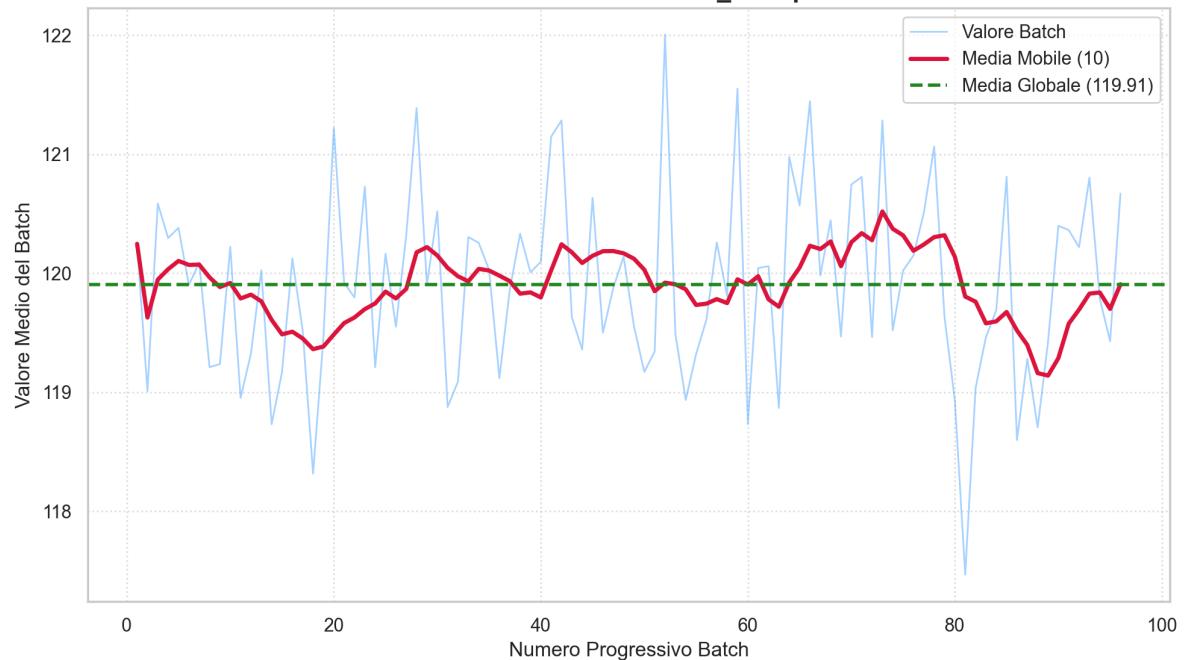
Analisi Stabilità Batch: Ts_XRay



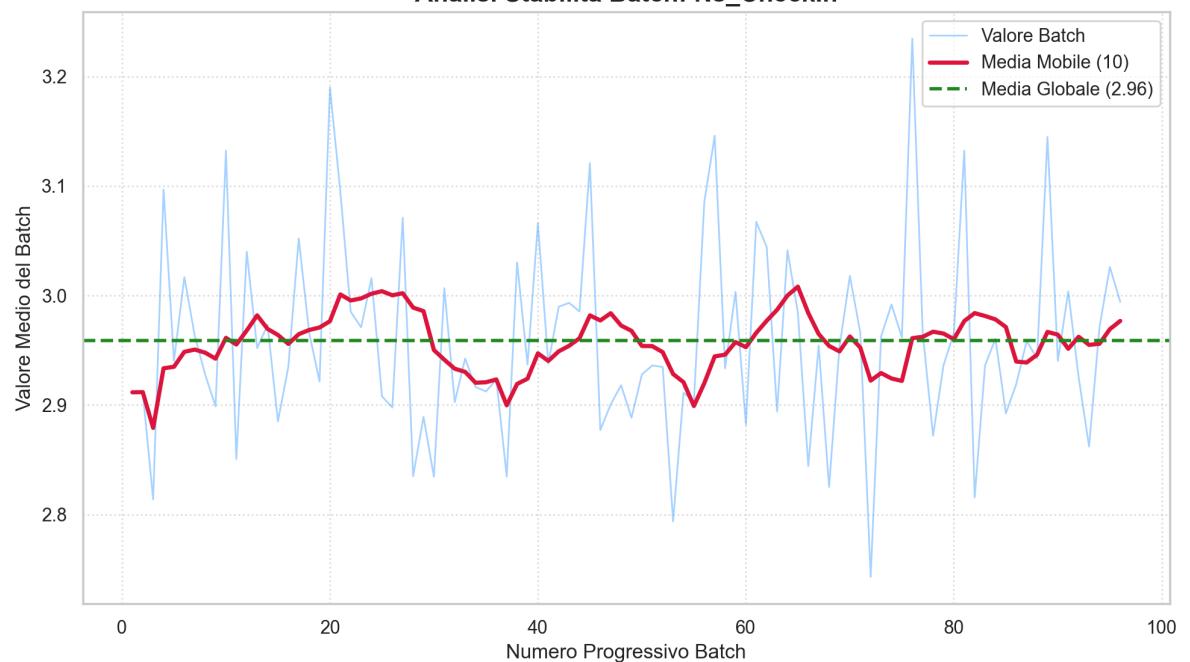
Analisi Stabilità Batch: Ts_TraceDetection

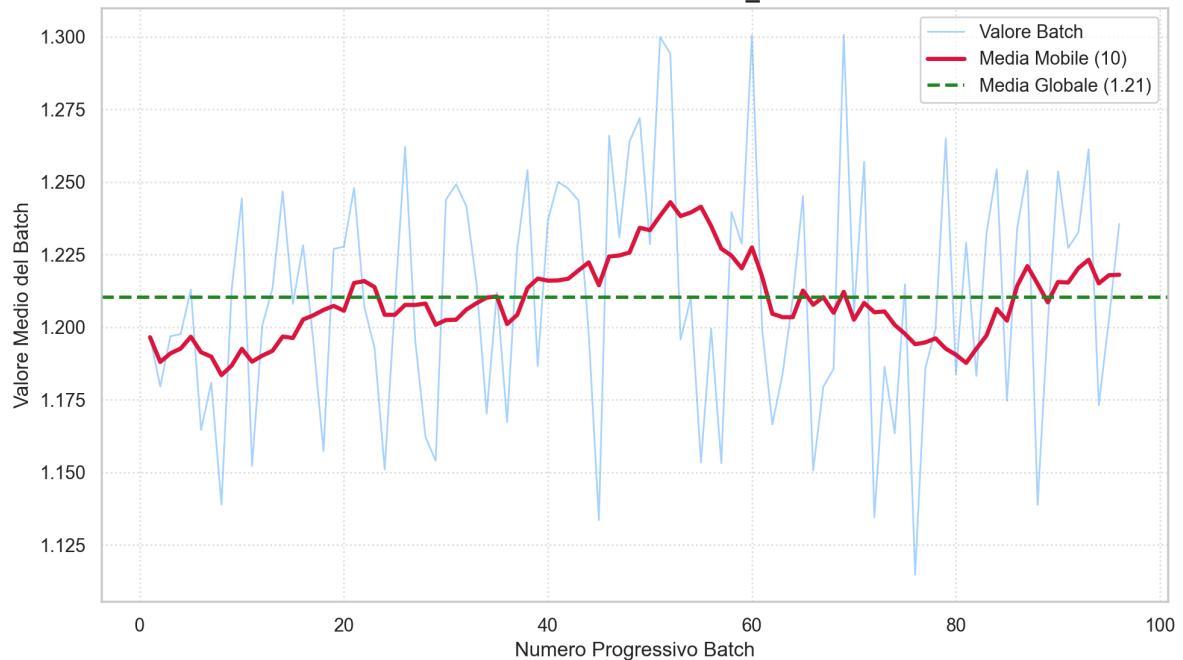


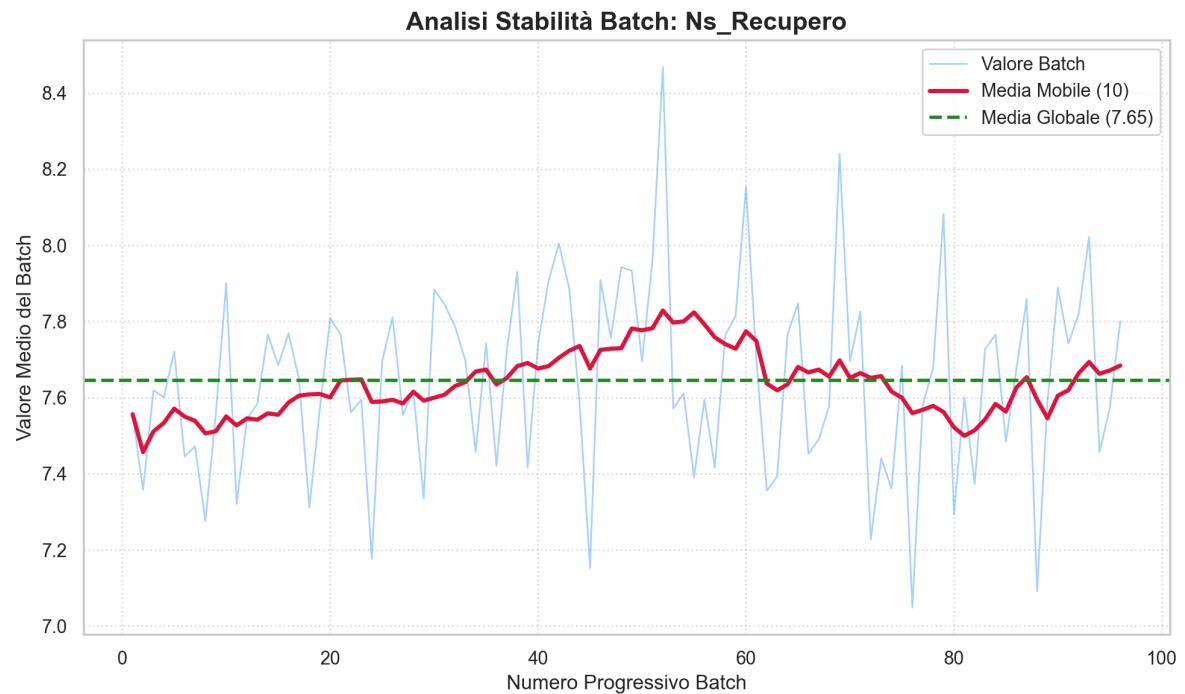
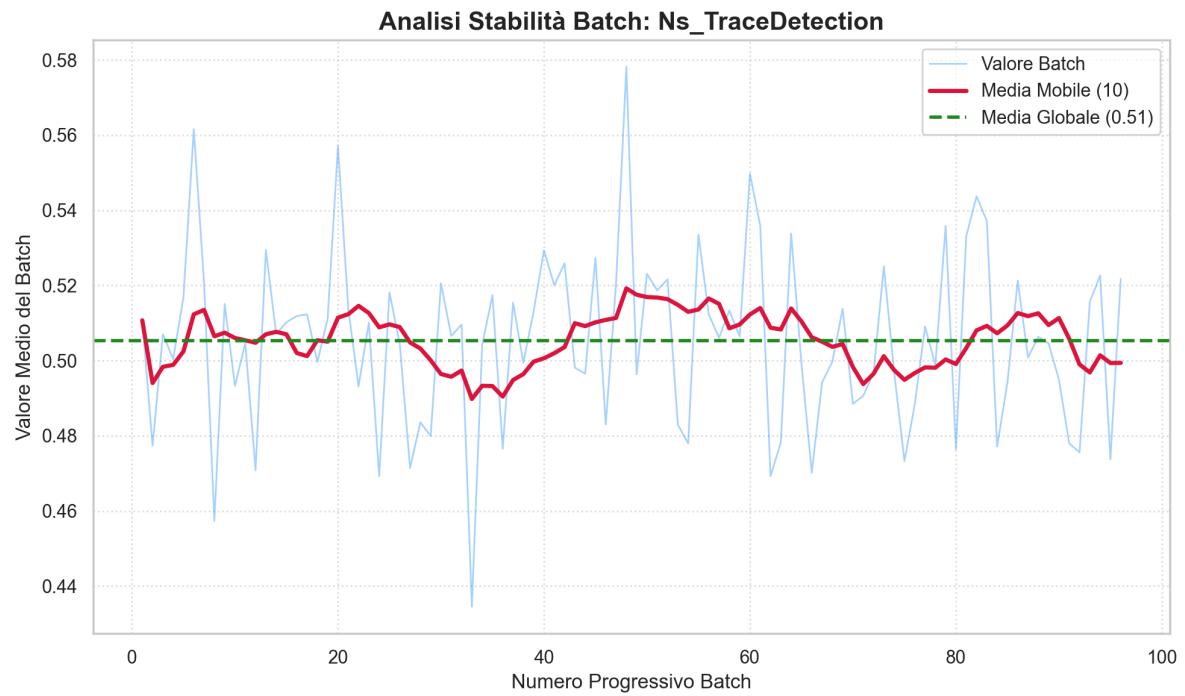
Analisi Stabilità Batch: Ts_Recupero



Analisi Stabilità Batch: Ns_CheckIn



Analisi Stabilità Batch: Ns_Varchi**Analisi Stabilità Batch: Ns_XRay**



Questo ci permette di affermare con certezza che le criticità emerse negli altri scenari, non sono dovute a errori di programmazione, ma derivano dalla conformazione fisica dei centri.

8.4 - Conclusioni

L'analisi sperimentale condotta in questo capitolo, ci ha permesso di validare e verificare la robustezza logica e matematica del simulatore sviluppato, e allo stesso tempo di mostrare le mancanze strutturali dell'aeroporto nella gestione dei flussi di passeggeri.

Il collo di bottiglia identificato presso i **Controlli a Raggi X**, rende il sistema attuale incapace di soddisfare tutti i requisiti di Qualità del Servizio (QoS) e di Sicurezza prefissati. In particolare:

1. Il primo obiettivo secondo cui il tempo medio di attesa in coda ai Raggi X non deve superare i 15 min ($E[T_{Q3}] \leq 15$ minuti) è stato violato. Nel nostro caso $E[T_{Q3}]$ è di circa **45,3 minuti**.
2. Così come è stato violato il secondo obiettivo secondo cui il numero di persone nella zona in prossimità dei controlli di sicurezza dovrebbe essere minore di 240 ($E[N_{S_controlli}] \leq 240$). Nel nostro caso è pari a circa **325 persone** solo negli XRay.
3. Invece, risulta soddisfatto l'ultimo requisito nel quale ci chiediamo se i passeggeri siano in grado di completare l'intero percorso (dall'ingresso fino all'imbarco) in meno di 80 minuti, che è la media tra il tempo raccomandato minimo (STD-40') e massimo (STD-120'). Infatti, come abbiamo visto, servono circa **49,5 min**.

Alla luce di questi risultati, il prossimo capitolo (*Capitolo 9*) sarà dedicato all'identificazione e al test di scenari alternativi, con l'obiettivo di riportare i tempi di attesa in coda agli XRay sotto i 15 min, e il numero di persone in prossimità dei controlli entro la soglia di sicurezza delle 240 persone.

9 - Modello migliorativo

Risulta evidente, come previsto, che il centro di controllo a raggi X è il collo di bottiglia del sistema, che addirittura causa il fatto che il sistema non raggiunge mai uno stato steady state, quindi, per elaborare un modello migliorativo ci si concentra sul migliorare le prestazioni del centro suddetto.

Il problema è che non si possono effettuare cambiamenti radicali all'interno dell'aeroporto, quindi, la soluzione più immediata e meno "impattante" sul sistema complessivo è quella di introdurre il *"fast track"*, che è un servizio a pagamento che offre un accesso dedicato ai controlli di sicurezza, permettendo, a chi usufruisce del servizio, di saltare le lunghe file. E' importante notare che gli utenti che utilizzano il servizio fast track sono i frequent flyer, o business. Offrono questo tipo di servizio gestori aeroportuali e compagnie aeree.

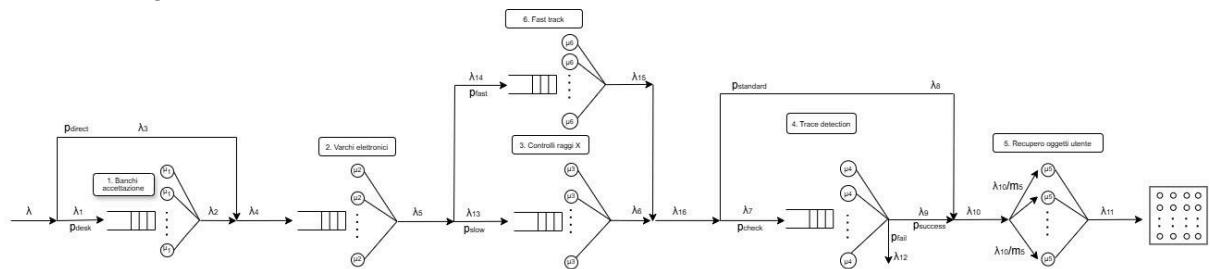
In realtà, in molti aeroporti è già presente, **ma poco utilizzato**, e pertanto, trascurato nel modello di base. Il nostro obiettivo è dimostrare che incentivare l'utilizzo da parte delle compagnie aeree (sconti, offerte, ecc.), porterà ad eliminare il collo di bottiglia, perché allevierà la pressione sul centro dei controlli a raggi X, e quindi al rispetto degli importanti QoS.

Dunque, in questo capitolo andremo a presentare esclusivamente le differenze introdotte rispetto al sistema base, per quanto riguarda il modello concettuale, di specifica e computazionale, senza essere ridondanti nella trattazione delle porzioni rimaste invariate all'interno della rete. Procederemo poi con le fasi di verifica, validazione e presentazione degli esperimenti condotti e dei risultati ottenuti.

9.1 - Modello concettuale

9.1.1 - Diagramma

Il nuovo diagramma con le modifiche:



9.1.2 - Descrizione

Le modifiche apportate al modello concettuale per lo scenario migliorativo sono due e mirano a rendere più fluidi i controlli di sicurezza.

1. In primo luogo, all'uscita dai varchi elettronici (Centro 2) è stato aggiunto un nuovo centro di servizio, identificato come **Centro 6 - Fast Track**, che opera in parallelo rispetto al collo di bottiglia individuato precedentemente, ovvero il Centro 3 - Controlli a Raggi X. In particolare, questo nuovo centro è modellato come un **MSSQ (Multi Server Single Queue)**, dove la coda viene gestita con una disciplina **FIFO**.
 - Dunque, a differenza del modello base, il flusso di passeggeri in uscita dai varchi elettronici subisce una biforcazione: una quota (p_{fast}) viene instradata verso il percorso prioritario (Centro 6), mentre la restante parte (p_{slow}) prosegue verso il percorso tradizionale (Centro 3).
 - È fondamentale notare che il Fast Track non sostituisce o elimina i controlli di sicurezza a raggi X, ma ne offre una via alternativa generalmente più rapida, in quanto caratterizzata da un minor numero di utenti e molto spesso da viaggiatori frequenti abituati a tali procedure.
 - Tuttavia i vincoli di sicurezza rimangono invariati e di conseguenza i due gruppi di passeggeri sono soggetti alla stessa probabilità p_{check} di essere selezionati per i controlli approfonditi presso il Trace Detection (Centro 4).
 2. In secondo luogo, è stato potenziato il centro di Trace Detection (Centro 4), che è

Single Queue). Questo perché l'eliminazione del collo di bottiglia ai Raggi X comporta un aumento del flusso di uscita verso il Centro 4 che rischierebbe di saturarlo. Tale modifica è perfettamente ammissibile in quanto non necessita dell'installazione di nuovi macchinari, bensì della semplice aggiunta di personale dotato di dispositivi portatili per il rilevamento di tracce di esplosivi e droghe.

9.1.3 - Sintesi

Ricapitolando, i centri di servizio sono i seguenti:

- Centro 1 - Banchi accettazione: MSSQ
- Centro 2 - Varchi elettronici: MSSQ
- Centro 3 - Controlli a raggi X: MSSQ
- Centro 4 - Trace detection: MSSQ
- Centro 5 - Recupero oggetti utente: IS
- Centro 6 - Fast track: MSSQ

Mentre le probabilità:

- p_{desk} = probabilità che i passeggeri si dirigano al centro 1 per l'accettazione.
- p_{direct} = probabilità che i passeggeri saltino il centro 1.
- $p_{standard}$ = probabilità che il passeggero non venga selezionato per il controllo aggiuntivo a campione.
- p_{check} = probabilità che il passeggero venga selezionato per il controllo a campione.
- p_{fail} = probabilità che il passeggero venga espulso dal sistema.
- p_{fast} = probabilità che il passeggero usufruisca del fast track.
- p_{slow} = probabilità che il passeggero affronti il percorso tradizionale, passando per gli X-Rays, senza fast track.

9.1.4 - Stato

$$S(t) = (N_1, N_2, N_3, N_4, N_5, N_6)$$

Le uniche differenze rispetto allo stato precedente sono le seguenti:

- N_4 (**Stato Centro 4 - MSSQ**): Un numero intero che rappresenta il numero totale di utenti presenti al trace detection (utenti in servizio + utenti in coda).
- N_6 (**Stato Centro 6 - MSSQ**): Un numero intero che rappresenta il numero totale di utenti presenti al fast track (utenti in servizio + utenti in coda).

9.1.5 - Eventi

I nuovi eventi e gli eventi modificati sono riassunti di seguito:

1. **Fine Servizio Varchi Elettronici (end_service_2):** Il passeggero attraversa il tornello. Si libera la risorsa specifica tra i molteplici server e l'utente si dirige verso il Centro 3 o verso il Centro 6, in base all'esito del routing probabilistico (p_{fast} vs p_{slow}).
2. **Arrivo Servizio Trace Detection (arrival_service_4):** Rappresenta l'ingresso di un nuovo passeggero nel servizio trace detection. Possono essere soggetti a questi controlli approfonditi sia i passeggeri standard sia quelli del servizio di fast track.
3. **Fine Servizio Trace Detection (end_service_4):** Il passeggero termina il controllo aggiuntivo e si libera una delle molteplici risorse. Si verifica la condizione di espulsione (p_{fail}): l'utente può uscire definitivamente dal sistema (fallimento) o rientrare nel flusso verso l'area Airside (successo) passando prima per le banchine per il recupero degli oggetti personali.
4. **Arrivo Servizio Fast Track (arrival_service_6):** Rappresenta l'ingresso di un nuovo passeggero nel servizio fast track.
5. **Fine Servizio Fast Track (end_service_6):** Il passeggero termina il servizio del fast track e si dirige verso le banchine per il recupero degli oggetti personali (Centro 5) oppure verso il controllo approfondito (Centro 4).

9.2 - Modello specifica

9.2.1 - Probabilità di routing

Le probabilità p_{desk} , p_{direct} , p_{check} , $p_{standard}$, p_{fail} , $p_{success}$ rimangono invariate. Si aggiungono le probabilità di entrare o meno in fast track da parte di un passeggero qualunque: p_{fast} e p_{slow} . La nostra proposta è di ripartire i passeggeri nel seguente modo:

- $p_{fast} = 0.33 = \frac{1}{3}$
- $p_{slow} = 1 - p_{fast} = 0.67 = \frac{2}{3}$

Ci aspettiamo che il centro XRay non sia più collo di bottiglia per il sistema, dato che il flusso degli arrivi è il 67% dell'originale (da $\lambda_{med} \rightarrow \lambda_{med} \cdot 0.67$).

9.2.2 - Matrice di routing

Definiamo i nodi del grafo $N = \{0, 1, 2, 3, 4, 5, 6\}$

- **6:** Centro 6 - Fast track
- Il resto, rimane uguale al modello base

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|----------|------------|------------|--------------|------------|-------------|----------------|------------|
| 0 | 0 | p_{desk} | p_{direct} | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | p_{slow} | 0 | 0 | p_{fast} |
| 3 | 0 | 0 | 0 | 0 | p_{check} | $p_{standard}$ | 0 |
| 4 | p_{fail} | 0 | 0 | 0 | 0 | $p_{success}$ | 0 |
| 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | p_{check} | $p_{standard}$ | 0 |

9.2.3 - Equazioni di traffico

Le equazioni che descrivono il traffico nella rete di code sono le seguenti: sono scritte con un colore più scuro le nuove equazioni del modello migliorativo. Gli indici vanno dal 13 al 16 per i nuovi flussi.

$$\lambda_1 = \lambda \cdot p_{desk}$$

$$\lambda_2 = \lambda_1$$

$$\lambda_3 = \lambda \cdot p_{direct}$$

$$\lambda_4 = \lambda_2 + \lambda_3$$

$$\lambda_5 = \lambda_4$$

$$\lambda_{13} = \lambda_5 \cdot p_{slow}$$

$$\lambda_6 = \lambda_{13}$$

$$\lambda_{14} = \lambda_5 \cdot p_{fast}$$

$$\lambda_{15} = \lambda_{14}$$

$$\lambda_{16} = \lambda_6 + \lambda_{15}$$

$$\lambda_7 = \lambda_{16} \cdot p_{check}$$

$$\lambda_8 = \lambda_{16} \cdot p_{standard}$$

$$\lambda_9 = \lambda_7 \cdot p_{success}$$

$$\lambda_{10} = \lambda_8 + \lambda_9$$

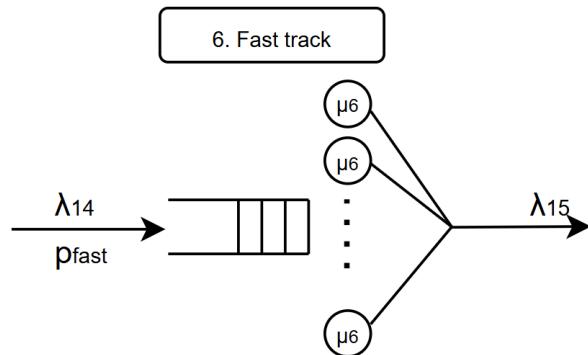
$$\lambda_{11} = \lambda_{10}$$

$$\lambda_{12} = \lambda_7 \cdot p_{fail}$$

9.2.4 - Modellazione centri

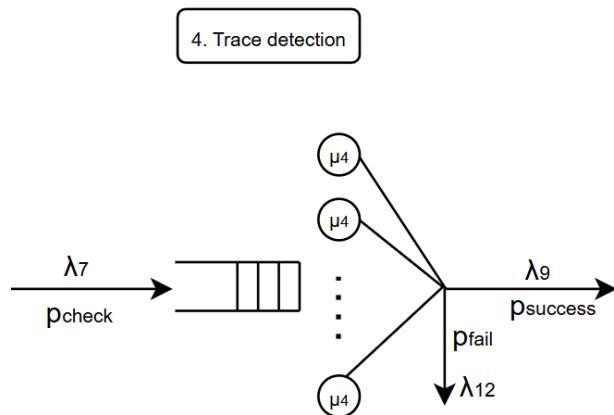
In questa fase indichiamo i nuovi centri introdotti nel migliorativo, o centri già esistenti, ma cambiati, come al solito la scelta della distribuzione e dei parametri riflette la realtà.

Centro 6: Fast track (nuovo)



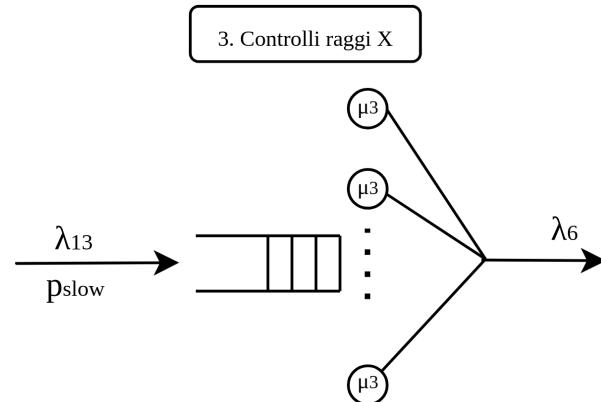
- **Tipologia Centro e Processo:** Modellato come un **MSSQ (Multi Server Single Queue)** con **$m_6 = 3$ server**. Un server altro non è che una macchina raggi X, e i tempi di servizio sono **simili** al centro dei controlli a raggi X perchè comunque *tutti* gli utenti (anche chi, appunto, usufruisce del *fast track*), per questioni di sicurezza, devono sottoporsi ai controlli a raggi X, è inevitabile. Tuttavia, ricordiamo che nella modellazione del centro dei controlli a raggi X, nei tempi di servizio, erano **inclusi** i tempi di preparazione dei passeggeri. Assumiamo che stavolta i passeggeri, essendo *frequent flyer*, siano più "svelti" rispetto alla media, quindi **riduciamo leggermente i tempi**.
- **Distribuzione:** Si sceglie la distribuzione **Normale Troncata**.
- **Parametri:**
 - **Media ($E[S_6]$):** 50 secondi.
 - **Deviazione Standard (σ_6):** 20 secondi.
 - **Minimo (LB_6):** 20 secondi.
 - **Massimo (UB_6):** 60 secondi.

Centro 4: Trace Detection (modificato)



Cambiamenti rispetto al modello base: il numero di server passa da **$m_4 = 1$ server (SSQ)** a **$m_4 = 2$ server (MSSQ)**. Aggiungere un servente al trace detection significa aggiungere un operatore umano. E' necessario perché avendo notevolmente ridotto il collo di bottiglia, ora l'operatore nel trace detection dovrà effettuare analisi approfondite su più passeggeri, a parità di tempo.

Centro 3: Controlli a Raggi X (modificato)



Cambiamenti rispetto al modello base: il flusso degli arrivi è $\lambda_{med} \cdot p_{slow}$, dato che l'altra parte va nel fast track ($\lambda_{med} \cdot p_{fast}$), riducendo il collo di bottiglia dovuto ai controlli a raggi X.

9.3 - Modello computazionale

Le modifiche al codice sorgente, per l'implementazione del modello migliorativo e l'esecuzione degli esperimenti sono state minime.

9.3.1 - Jobs che attraversano la rete

In `mbpmcsn.entity.Job`, ora c'è un attributo in più:

```
private boolean fastTrackBeingUsed;
```

con un getter e un setter per l'attributo:

```
public boolean isFastTrackBeingUsed() {
    return fastTrackBeingUsed;
}

public void setFastTrackBeingUsed(boolean fastTrackBeingUsed) {
    this.fastTrackBeingUsed = fastTrackBeingUsed;
}
```

9.3.2 - Nuovo routing point

Implementato un nuovo NetworkRoutingPoint per decidere se il job va al fast track o segue il percorso classico (verso gli X-rays): mbpmcsn.routing.VarchiRouting

```
public final class VarchiRouting implements NetworkRoutingPoint {

    private final Center fastTrack;
    private final Center xRayStandard;
    private final int streamIndex;

    public VarchiRouting(Center fastTrack, Center xRayStandard, int streamIndex) {
        this.fastTrack = fastTrack;
        this.xRayStandard = xRayStandard;
        this.streamIndex = streamIndex;
    }

    @Override
    public Center getNextCenter(Rngs r, Job job) {
        r.selectStream(streamIndex);

        boolean isFastTrackBeingUsed = r.random() < IMPROVED_P_FAST_TRACK;
        job.setFastTrackBeingUsed(isFastTrackBeingUsed);

        return isFastTrackBeingUsed ? fastTrack : xRayStandard;
    }
}
```

9.3.3 - Improved simulation model

E' stato anche semplice implementare il nuovo (improved) simulation model, creando una nuova classe mbpmcsn.core.ImprovedSimulationModel: si riportano di seguito, i cambiamenti rispetto al modello base.

Si istanzia il nuovo rvg per il fast track, con i suoi parametri:

```
rvgFastTrack = new TruncatedNormalGenerator(IMPROVED_MEAN_S6,      IMPROVED_STD_S6,
IMPROVED_LB6, IMPROVED_UB6);
```

Si istanziano il nuovo centro MSSQ per il fast track, con il suo service process associato, basato sull'rvg appena creato:

```
// --- 6. Fast Track ---
ServiceProcess sp6 = new ServiceProcess(rvgFastTrack, rngs, STREAM_S6_SERVICE);
fastTrack = new MultiServerSingleQueue(
    IMPROVED_ID_FAST_TRACK,
    "FastTrack",
```

```
/*...*/
IMPROVED_M6
);
```

Il nuovo NetworkRoutingPoint per decidere se il job/passeggero va in fast track o segue il percorso classico, in base alle probabilità di routing:

```
NetworkRoutingPoint routingVarchi = new VarchiRouting(fastTrack, centerXRay,
STREAM_S2_ROUTING);
```

E infine, la modifica al centro trace detection, che diventa un MSSQ:

```
centerTrace = new MultiServerSingleQueue(
    ID_TRACE_DETECTION,
    "TraceDetection",
    /*...*/
    IMPROVED_M4
);
```

9.4 - Verifica

Effettuiamo la verifica con l'approssimazione dei tempi di servizio dei vari centri a tempi esponenziali. Stavolta il sistema raggiunge lo stato stazionario e tutti i $\rho_k < 1 \forall \text{ centro } k = 1, \dots, 6$. Il tasso di ingressi al sistema è $\lambda = \lambda_{med} = 0,127205 \text{ pax/s}$.

Confronteremo i risultati generati dal simulatore nel caso stazionario, con simulazione ad orizzonte infinito. Le approssimazioni sopra elencate, sono attuate anche per il simulatore. L'intervallo di confidenza è del 95%. Confronteremo centro-per-centro.

Si ricorda che gli indici dei vari centri rimangono invariati, bisogna solo aggiungere il nuovo centro fast track con indice 6.

9.4.1 - Verifica del centro check-in M/M/8

Utilizziamo le formule per un centro M/M/8:

$$\rho_1 = \frac{\lambda_{med} \cdot p_{desk}}{m_1 \cdot \mu_1} = 0,7388$$

$$p_{0,1} = \left(\sum_{i=0}^{m_1-1} \frac{(m_1 \rho_1)^i}{i!} + \frac{(m_1 \rho_1)^{m_1}}{m_1! \cdot (1-\rho_1)} \right)^{-1} = 0.002379$$

$$P_{Q,1} = \frac{(m_1 \rho_1)^{m_1}}{m_1! \cdot (1-\rho_1)} \cdot p_{0,1} = 0.336340$$

$$E(T_{Q,1}) = \frac{P_{Q,1}}{m_1 \cdot \mu_1 - \lambda_{med} \cdot p_{desk}} = 19,3129s$$

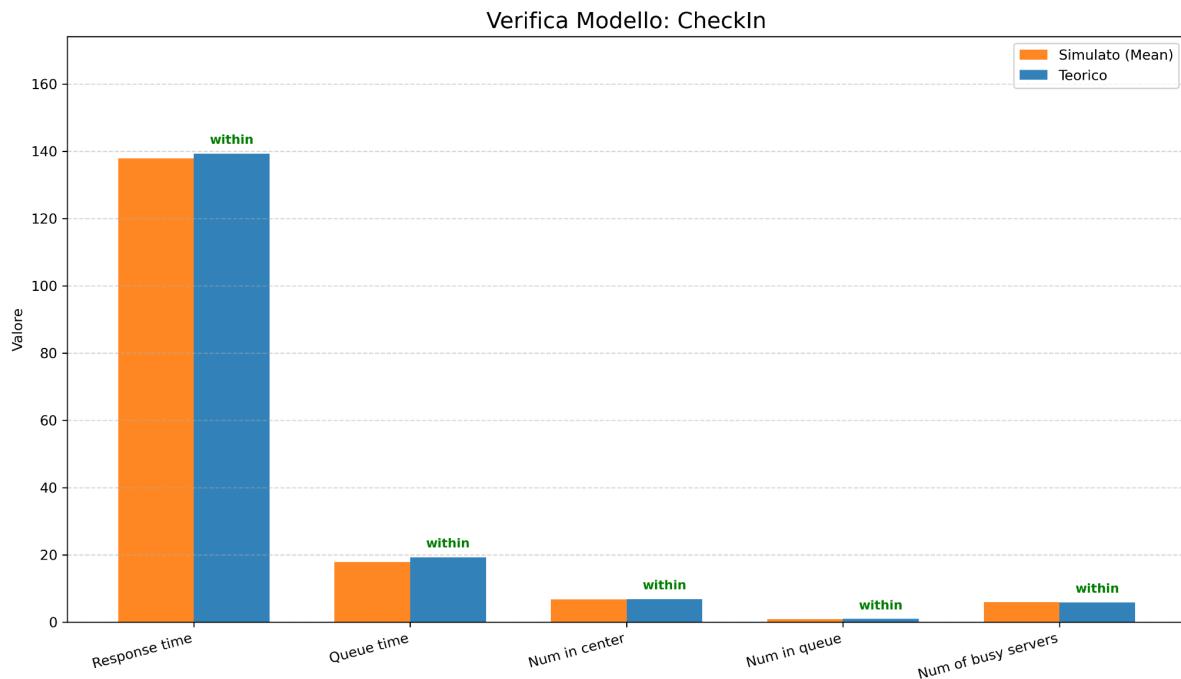
$$E(S_{i,1}) = \frac{1}{\mu_1} = 120s$$

$$E(T_{S,1}) = E(T_{Q,1}) + E(S_{i,1}) = 139,3129s$$

$$E(N_{Q,1}) = \lambda_{med} \cdot p_{desk} \cdot E(T_{Q,1}) = 0,9512$$

$$E(N_{S,1}) = \lambda_{med} \cdot p_{desk} \cdot E(T_{S,1}) = 6,8614$$

$$m_1 \rho_1 = 5,9102$$



| Metrica | SimMean | SimMin | SimMax | SimWidth | TheoValue | WithinInt |
|--------------------|-----------|-----------|-----------|------------|-----------|-----------|
| $E(T_{S,1})$ | 137,9006s | 135,7853s | 140,0160s | 2,11535519 | 139,3129s | within |
| $E(T_{Q,1})$ | 17,8713s | 16,3482s | 19,3944s | 1,52309928 | 19,3129s | within |
| $E(N_{S,1})$ | 6,8108 | 6,6900 | 6,9316 | 0,12077353 | 6,8614 | within |
| $E(N_{Q,1})$ | 0,8858 | 0,8075 | 0,9642 | 0,07836096 | 0,9512 | within |
| $m_1 \cdot \rho_1$ | 5,9250 | 5,8736 | 5,9763 | 0,05136753 | 5,9102 | within |

9.4.2 - Verifica del centro varchi elettronici M/M/4

Utilizziamo le formule per un centro M/M/4:

$$\rho_2 = \frac{\lambda_{med}}{m_2 \cdot \mu_2} = 0,47702$$

$$p_{0,2} = \left(\sum_{i=0}^{m_2-1} \frac{(m_2 \rho_2)^i}{i!} + \frac{(m_2 \rho_2)^{m_2}}{m_2! \cdot (1-\rho_2)} \right)^{-1} = 0.144044$$

$$P_{Q,2} = \frac{(m_2 \rho_2)^{m_2}}{m_2! \cdot (1-\rho_2)} \cdot p_{0,2} = 0.152118$$

$$E(T_{Q,2}) = \frac{P_{Q,2}}{m_2 \cdot \mu_2 - \lambda_{med}} = 1,0908s$$

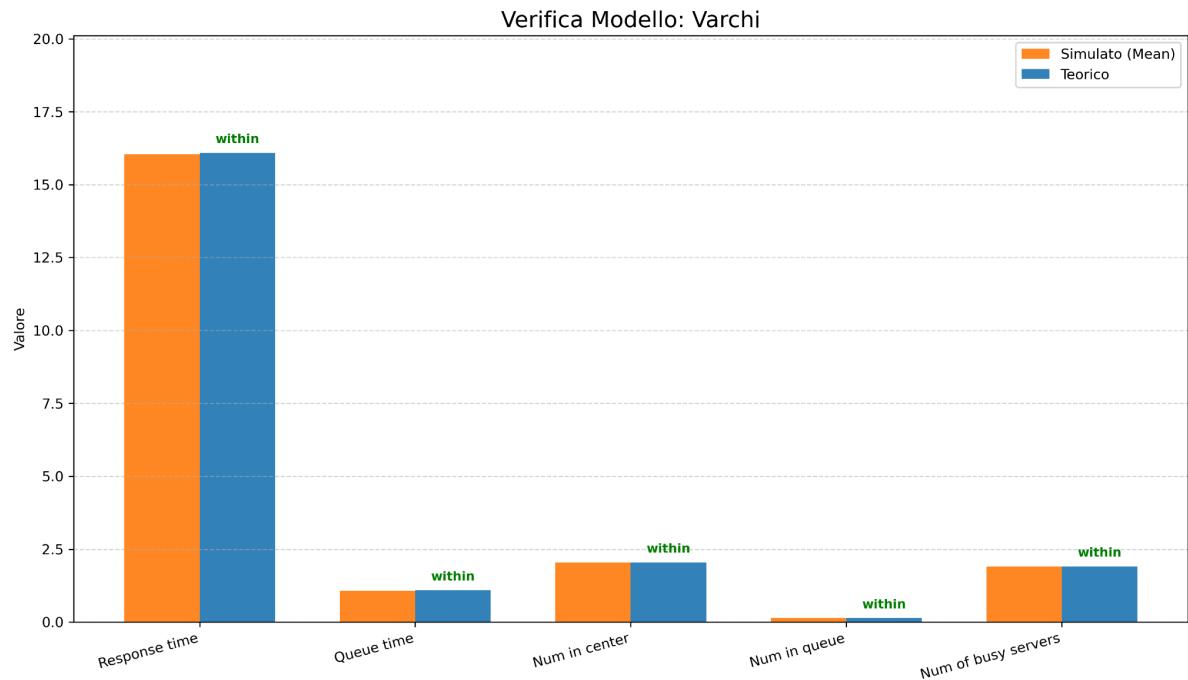
$$E(S_{i,2}) = \frac{1}{\mu_2} = 15s$$

$$E(T_{S,2}) = E(T_{Q,2}) + E(S_{i,2}) = 16,0908s$$

$$E(N_{Q,2}) = \lambda_{med} \cdot E(T_{Q,2}) = 0,1387$$

$$E(N_{S,2}) = \lambda_{med} \cdot E(T_{S,2}) = 2,0468$$

$$m_2 \rho_2 = 1,9081$$



| Metrica | SimMean | SimMin | SimMax | SimWidth | TheoValue | WithinInt |
|--------------------|----------|----------|----------|------------|-----------|-----------|
| $E(T_{S,2})$ | 16,0448s | 15,9104s | 16,1792s | 0,13436749 | 16,0908s | within |
| $E(T_{Q,2})$ | 1,0669s | 0,9991s | 1,1346s | 0,06775763 | 1,0908s | within |
| $E(N_{S,2})$ | 2,0497 | 2,0257 | 2,0736 | 0,02392825 | 2,0468 | within |
| $E(N_{Q,2})$ | 0,1367 | 0,1277 | 0,1457 | 0,00900086 | 0,1387 | within |
| $m_2 \cdot \rho_2$ | 1,9130 | 1,8953 | 1,9307 | 0,01768490 | 1,9081 | within |

9.4.3 - Verifica del centro x-rays M/M/6

Utilizziamo le formule per un centro M/M/6:

$$\rho_3 = \frac{\lambda_{med} \cdot p_{slow}}{m_3 \cdot \mu_3} = 0,8523$$

$$p_{0,3} = \left(\sum_{i=0}^{m_3-1} \frac{(m_3 \rho_3)^i}{i!} + \frac{(m_3 \rho_3)^{m_3}}{m_3! \cdot (1-\rho_3)} \right)^{-1} = 0.003742$$

$$P_{Q,3} = \frac{(m_3 \rho_3)^{m_3}}{m_3! \cdot (1-\rho_3)} \cdot p_{0,3} = 0.629118$$

$$E(T_{Q,3}) = \frac{P_{Q,3}}{m_3 \cdot \mu_3 - \lambda_{med} \cdot p_{slow}} = 42,5867s$$

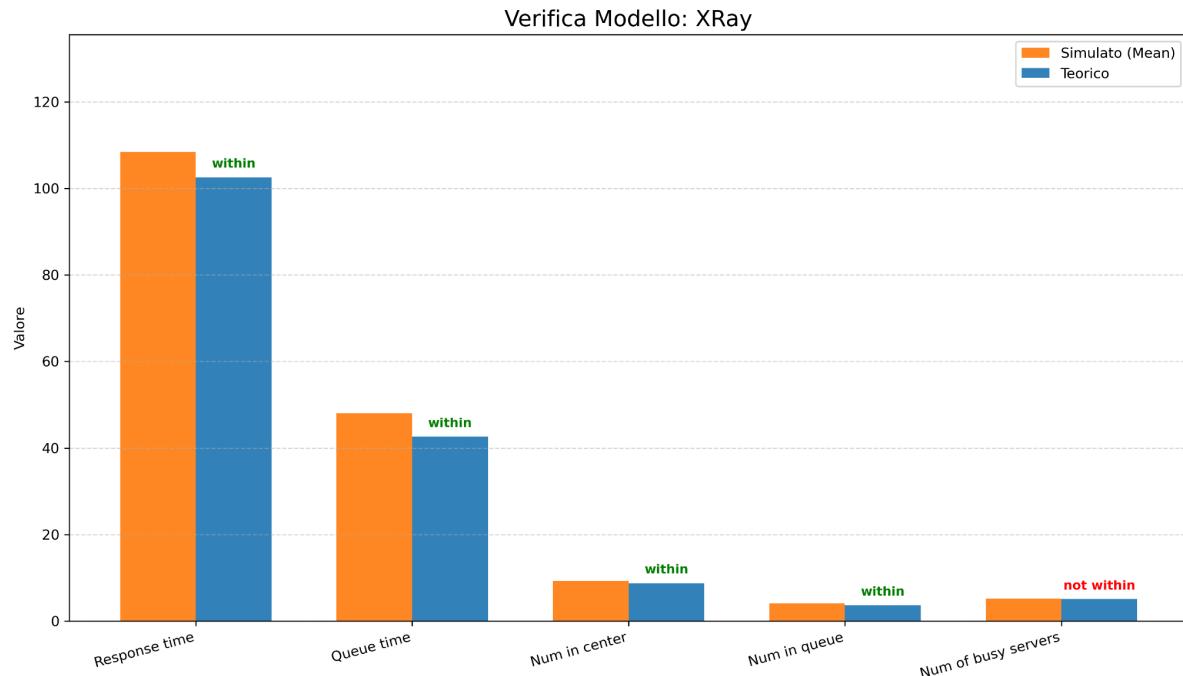
$$E(S_{i,3}) = \frac{1}{\mu_3} = 60s$$

$$E(T_{S,3}) = E(T_{Q,3}) + E(S_{i,3}) = 102,5867s$$

$$E(N_{Q,3}) = \lambda_{med} \cdot p_{slow} \cdot E(T_{Q,3}) = 3,6296$$

$$E(N_{S,3}) = \lambda_{med} \cdot p_{slow} \cdot E(T_{S,3}) = 8,7432$$

$$m_3 \rho_3 = 5,1136$$



| Metrica | SimMean | SimMin | SimMax | SimWidth | TheoValue | WithinInt |
|--------------|-----------|-----------|-----------|------------|-----------|-----------|
| $E(T_{S,3})$ | 108,4452s | 101,7233s | 115,1672s | 6,72192025 | 102,5867s | within |
| $E(T_{Q,3})$ | 47,9985s | 41,4666s | 54,5304s | 6,53185485 | 42,5867s | within |

| | | | | | | |
|--------------------|--------|--------|--------|------------|--------|------------|
| $E(N_{S,3})$ | 9,3033 | 8,6755 | 9,9311 | 0,62780889 | 8,7432 | within |
| $E(N_{Q,3})$ | 4,1414 | 3,5474 | 4,7355 | 0,59406487 | 3,6296 | within |
| $m_3 \cdot \rho_3$ | 5,1619 | 5,1138 | 5,2099 | 0,04803635 | 5,1136 | not within |

9.4.4 - Verifica del centro fast track M/M/3

Utilizziamo le formule per un centro M/M/3:

$$\rho_6 = \frac{\lambda_{med} \cdot p_{fast}}{m_6 \cdot \mu_6} = 0,6996$$

$$p_{0,6} = \left(\sum_{i=0}^{m_6-1} \frac{(m_6 \rho_6)^i}{i!} + \frac{(m_6 \rho_6)^{m_6}}{m_6! \cdot (1-\rho_6)} \right)^{-1} = 0.095859$$

$$P_{Q,6} = \frac{(m_6 \rho_6)^{m_6}}{m_6! \cdot (1-\rho_6)} \cdot p_{0,6} = 0.491798$$

$$E(T_{Q,6}) = \frac{P_{Q,6}}{m_6 \cdot \mu_6 - \lambda_{med} \cdot p_{fast}} = 27,2883s$$

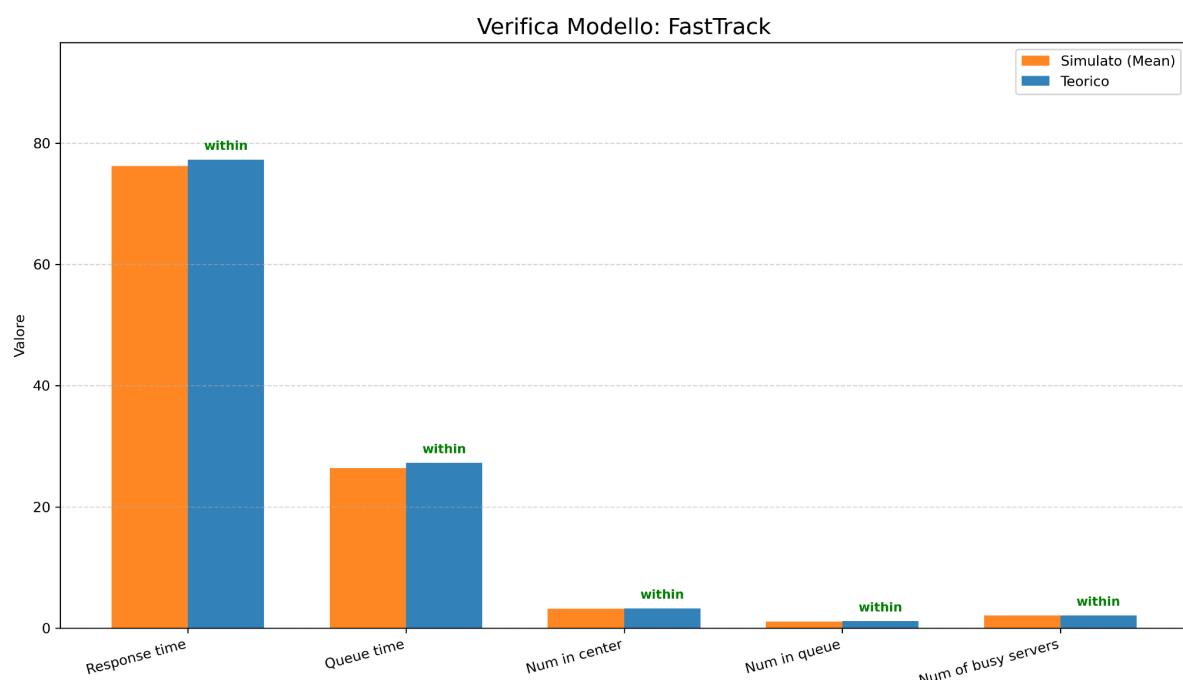
$$E(S_{i,6}) = \frac{1}{\mu_6} = 50s$$

$$E(T_{S,6}) = E(T_{Q,6}) + E(S_{i,6}) = 77,2883s$$

$$E(N_{Q,6}) = \lambda_{med} \cdot p_{fast} \cdot E(T_{Q,6}) = 1,1455$$

$$E(N_{S,6}) = \lambda_{med} \cdot p_{fast} \cdot E(T_{S,6}) = 3,2444$$

$$m_6 \rho_6 = 2,0989$$



| Metrica | SimMean | SimMin | SimMax | SimWidth | TheoValue | WithinInt |
|--------------------|----------|----------|----------|------------|-----------|-----------|
| $E(T_{S,6})$ | 76,2598s | 74,5868s | 77,9328s | 1,67298414 | 77,2883s | within |
| $E(T_{Q,6})$ | 26,3992s | 24,8725s | 27,9258s | 1,52664580 | 27,2883s | within |
| $E(N_{S,6})$ | 3,2033 | 3,1243 | 3,2823 | 0,07897483 | 3,2444 | within |
| $E(N_{Q,6})$ | 1,1111 | 1,0438 | 1,1785 | 0,06732807 | 1,1455 | within |
| $m_6 \cdot \rho_6$ | 2,0921 | 2,0756 | 2,1087 | 0,01653246 | 2,0989 | within |

9.4.5 - Verifica del centro trace detection M/M/2

Utilizziamo le formule per un centro M/M/2:

$$\rho_4 = \frac{\lambda_{med} \cdot p_{check}}{m_4 \cdot \mu_4} = 0,3816$$

$$p_{0,4} = \left(\sum_{i=0}^{m_4-1} \frac{(m_4 \rho_4)^i}{i!} + \frac{(m_4 \rho_4)^{m_4}}{m_4! \cdot (1-\rho_4)} \right)^{-1} = 0.447581$$

$$P_{Q,4} = \frac{(m_4 \rho_4)^{m_4}}{m_4! \cdot (1-\rho_4)} \cdot p_{0,4} = 0.210811$$

$$E(T_{Q,4}) = \frac{P_{Q,4}}{m_4 \cdot \mu_4 - \lambda_{med} \cdot p_{check}} = 10,2272s$$

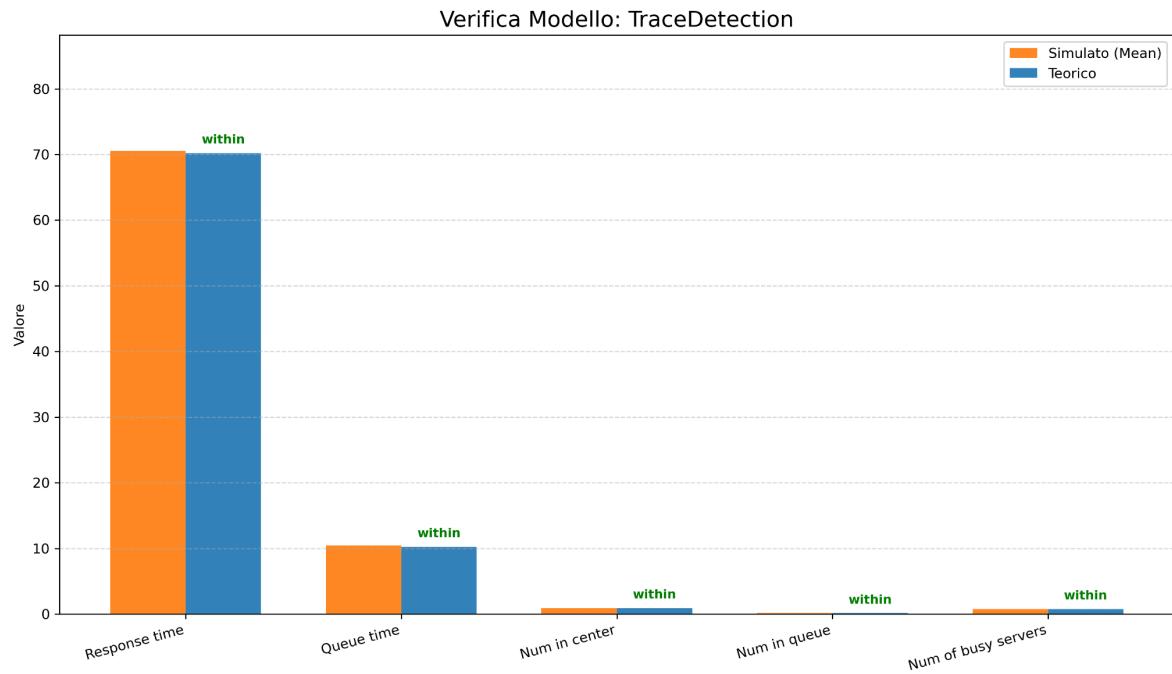
$$E(S_{i,4}) = \frac{1}{\mu_4} = 60s$$

$$E(T_{S,4}) = E(T_{Q,4}) + E(S_{i,4}) = 70,2272s$$

$$E(N_{Q,4}) = \lambda_{med} \cdot p_{check} \cdot E(T_{Q,4}) = 0,1301$$

$$E(N_{S,4}) = \lambda_{med} \cdot p_{check} \cdot E(T_{S,4}) = 0,8933$$

$$m_4 \rho_4 = 0,7632$$



9.4.6 - Verifica del centro recupero M/M/ ∞

Utilizziamo le formule per un centro M/M/ ∞ :

$$E(S_5) = \frac{1}{\mu_5} = 120s$$

$$E(T_{S,5}) = E(S_5) = 120s$$

$$E(N_{S,5}) = (\lambda_{med} \cdot p_{standard} + \lambda_{med} \cdot p_{check} \cdot p_{success}) \cdot E(T_{S,5}) = 15,2644$$



9.4.7 - Controlli di consistenza

I seguenti controlli di consistenza vengono rispettati:

$$E(T_{S,k}) = E(T_{Q,k}) + E(S_{i,k})$$

$$E(N_{S,k}) = E(N_{Q,k}) + m_k \cdot \rho_k$$

$$0 < \rho < 1$$

$$\forall \text{ centro } k = 1, 2, 3, 4, 5, 6$$

9.5 - Validazione

In questa fase ci assicuriamo che anche il modello computazionale implementato per il caso migliorativo sia conforme e consistente con il caso di studio analizzato.

9.5.1 - Verifica delle Probabilità di Instradamento

Analogamente a quanto fatto per il modello base, andiamo a verificare la correttezza della nuova logica di routing introdotta, ovvero la biforcazione all'uscita dei Varchi Elettronici (Centro 2), dove il flusso deve dividersi tra il Fast Track (Centro 6) e i controlli standard (Centro 3). Dai requisiti del modello migliorativo abbiamo impostato che, un terzo dei passeggeri ($p_{fast} = 0.33$) usufruisca del percorso prioritario, mentre i restanti due terzi ($p_{slow} = 0.67$) proseguano verso i raggi X tradizionali. L'analisi dei log conferma che il generatore di numeri casuali e la logica di instradamento funzionano correttamente, infatti su un totale di **8169 passeggeri transitati dai varchi**: **2705** sono stati instradati al **Fast Track** (0,3311), mentre **5464** sono stati instradati ai **Raggi X** (0,6689):

```
>>> CHECK 3: Split Fast Track vs X-Ray Standard (SCENARIO IMPROVED)
      Totale Uscita Varchi: 8169
      -----
      [FAST TRACK] Target: ~33.0% | Simulato: 33,1130% (Pax: 2705)
      [STANDARD]   Target: ~67.0% | Simulato: 66,8870% (Pax: 5464)
```

9.5.2 - Verifica dei Volumi di Traffico

Come fatto per il modello base verifichiamo la consistenza del generatore di arrivi rispetto ai parametri di input, calcolando:

$$N_{atteso} = \lambda_{med} \cdot T = 8242 \text{ passeggeri}$$

dove $\lambda_{med} = 0,127205 \text{ pax / s}$ e $T = 18 \text{ h} = 64800 \text{ s}$ (durata giornata lavorativa aeroporto).

Il risultato della simulazione riporta un totale di **8.169** passeggeri processati, cioè lo scarto è di 75 passeggeri, pari a un errore relativo dello **0,9%** ed entro i limiti delimitati dalla deviazione standard $\sigma = \sqrt{N_{atteso}} \simeq 91$ passeggeri.

9.5.3 - Verifica dei Parametri di Servizio

Di seguito si confrontano i tempi medi di servizio attesi con quelli effettivamente misurati durante la simulazione ad orizzonte finito:

| Centro di Servizio | Distribuzione Input | Media Teorica (E[S]th) | Media Simulata (E[S]sim) | Scostamento |
|---------------------------|---------------------|------------------------|--------------------------|-------------|
| Banchi Check-in | Norm. Troncata | 120,00 s | 119,6297 s | - 0,31% |
| Varchi Elettronici | Norm. Troncata | 15,00 s | 18,8157 s | +25,44% |

| | | | | |
|--------------------------|----------------|----------|------------|----------|
| Controlli Raggi X | Norm. Troncata | 60,00 s | 51,0584 | -14,90% |
| Trace Detection | Norm. Troncata | 60,00 s | 59,8300 s | -0,28% |
| Recupero Oggetti | Norm. Troncata | 120,00 s | 119,8666 s | -0,11% |
| Fast Track | Norm. Troncata | 50,00 s | 42,6166 s | - 14,77% |

Anche in questo caso si notano scostamenti maggiori per i centri Varchi, Raggi X e Fast Track, dovuti all'utilizzo della distribuzione Normale Troncata con deviazione standard elevata rispetto alla media e parametri di troncamento (upper e lower bound) vicini al valore medio, che spostano la media del servizio verso destra o sinistra.

9.5.4 - Valutazione Macroscopica (Legge di Little)

Applichiamo la Legge di Little ($N = \lambda T$) considerando l'intero aeroporto come una *black box* in condizione stabile, e mostrando i risultati dell'esecuzione steady state con λ_{med} :

| | | | | | |
|-------------------|--|----------------|------------|--------------|------------------------|
| Ns_CheckIn | | 6,4370 +/- | 0,0610 [| 6,3760 ... | 6,4980] AC: -0,162 |
| Ns_FastTrack | | 2,0505 +/- | 0,0193 [| 2,0312 ... | 2,0698] AC: -0,043 |
| Ns_Recupero | | 15,3110 +/- | 0,0997 [| 15,2113 ... | 15,4107] AC: -0,071 |
| Ns_TraceDetection | | 0,8325 +/- | 0,0062 [| 0,8263 ... | 0,8387] AC: 0,001 |
| Ns_Varchi | | 2,6323 +/- | 0,0251 [| 2,6072 ... | 2,6575] AC: -0,020 |
| Ns_XRay | | 4,8436 +/- | 0,0631 [| 4,7806 ... | 4,9067] AC: -0,051 |
| | | | | | |
| Ts_CheckIn | | 130,3466 +/- | 0,6593 [| 129,6873 ... | 131,0059] AC: -0,145 |
| Ts_FastTrack | | 48,8481 +/- | 0,2535 [| 48,5945 ... | 49,1016] AC: 0,050 |
| Ts_Recupero | | 119,8990 +/- | 0,1607 [| 119,7383 ... | 120,0597] AC: 0,040 |
| Ts_TraceDetection | | 65,1799 +/- | 0,1597 [| 65,0202 ... | 65,3397] AC: 0,064 |
| Ts_Varchi | | 20,6043 +/- | 0,0898 [| 20,5145 ... | 20,6941] AC: 0,004 |
| Ts_XRay | | 56,6682 +/- | 0,3963 [| 56,2719 ... | 57,0645] AC: -0,033 |

Dai risultati ottenuti osserviamo che:

- **Numero medio utenti nel sistema ($E[Ns]$):** Calcolato come somma delle popolazioni medie di tutti i centri:

$$\begin{aligned}
 E[Ns] &= 6,4370 + 2,6323 + 4,8436 + 2,0505 + 0,8325 + 15,3110 \\
 &= 32,1069 \text{ passeggeri}
 \end{aligned}$$

- **Tempo medio di risposta ($E[Ts]$):** Calcolato come somma pesata dei tempi di soggiorno in base alle probabilità di routing:

$$\begin{aligned}
 E[Ts] &= (130,3466 \cdot p_{desk}) + 20,6043 + (48,8481 \cdot p_{fast}) + (56,6682 \cdot p_{slow}) \\
 &\quad + (65,1799 \cdot p_{check}) + 119,8990 = 251,58 \text{ secondi}
 \end{aligned}$$

Dunque:

$$E[Ns]_{teorico} = \lambda_{med} \cdot E[Ts] = 0,127205 \text{ pax / s} \cdot 251,58 \text{ s} = 32,0022 \text{ passeggeri}$$

Confrontando il valore simulato (32,1069) con quello teorico (32,0022), lo scostamento è irrilevante (errore $\approx 0,3\%$), confermando la consistenza matematica del simulatore.

9.5.5 - Verifiche di Consistenza

A completamento della validazione del modello migliorativo, verifichiamo che per il nuovo centro introdotto, il **Fast Track (Centro 6)**, siano rispettate le leggi fondamentali delle code. In particolare, controlliamo che il tempo medio di risposta corrisponda alla somma tra il tempo medio di attesa in coda e il tempo medio di servizio:

$$E[T_s] = E[T_Q] + E[S]$$

Utilizzando i risultati dello scenario Steady State, leggiamo dai log:

- Tempo medio di servizio ($E[S]_{FastTrack}$): **42,6209 s**
- Tempo medio di attesa ($E[T_Q]_{FastTrack}$): **6,2271 s**
- Tempo medio di risposta ($E[T_s]_{FastTrack}$): **48,8481 s**

L'uguaglianza è verificata: **6,2271 s + 42,6209 s = 48,8480 s**.

In maniera analoga, testiamo la relazione sulla popolazione, verificando che il numero medio di utenti nel centro sia pari alla somma tra gli utenti in coda e quelli in servizio:

$$E[N_s] = E[N_Q] + m\rho$$

Anche qui, dai log dello scenario Steady State per il Fast Track otteniamo:

- Numero medio server attivi ($E[X]_{FastTrack} = m_6 \rho_6$): **1,7885**
- Numero medio job in coda ($E[N_Q]_{FastTrack}$): **0,2620**
- Numero medio job nel centro, coda + servizio ($E[N_s]_{FastTrack}$): **2,0505**

L'uguaglianza è verificata: **0,2620 + 1,7885 = 2,0505**.

9.6 - Design degli esperimenti

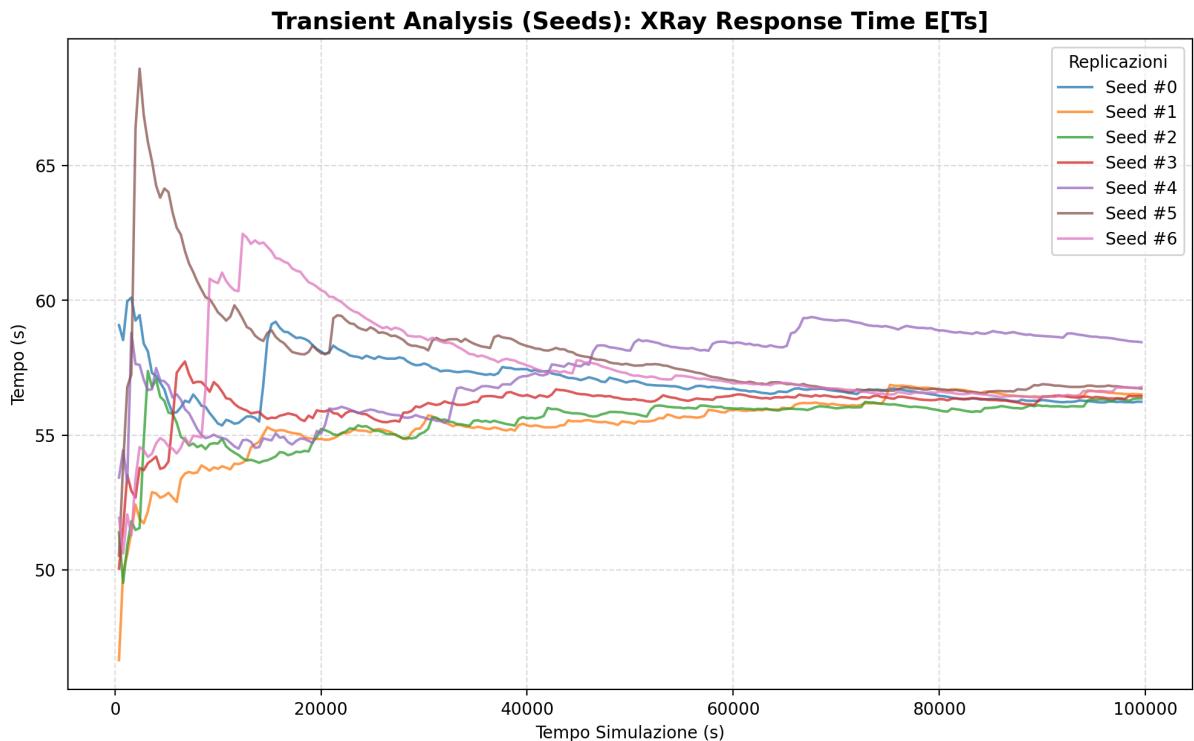
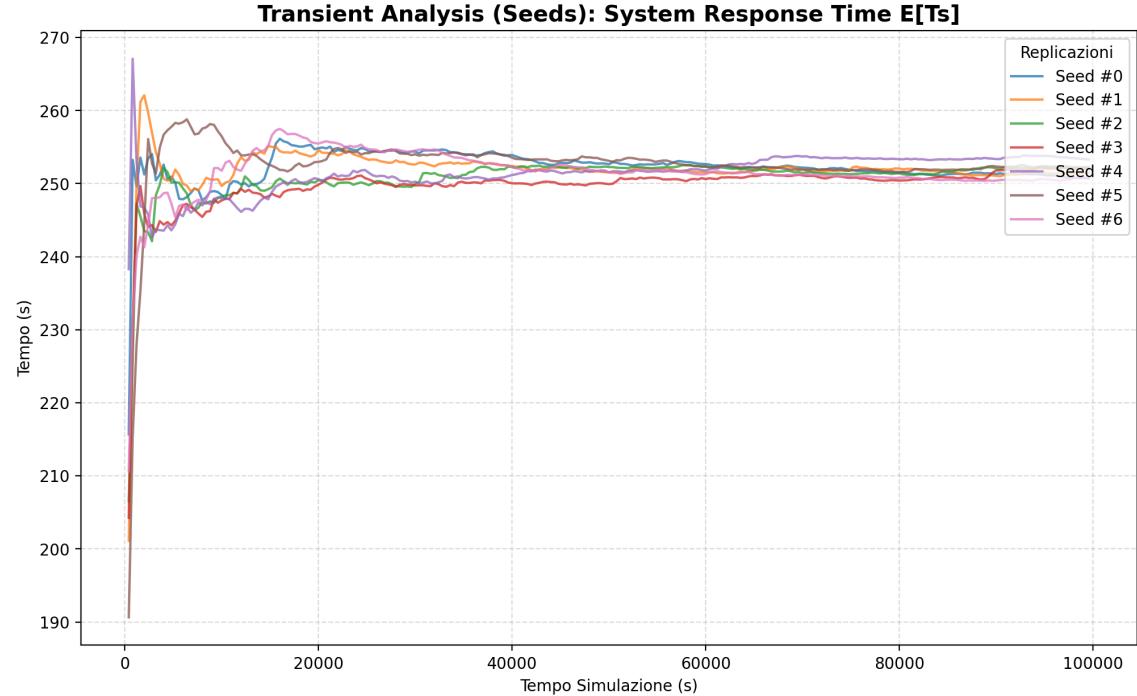
La fase di progettazione degli esperimenti ha l'obiettivo di mostrare gli esperimenti condotti e di analizzare i risultati ottenuti.

9.6.1 - Analisi transitorio

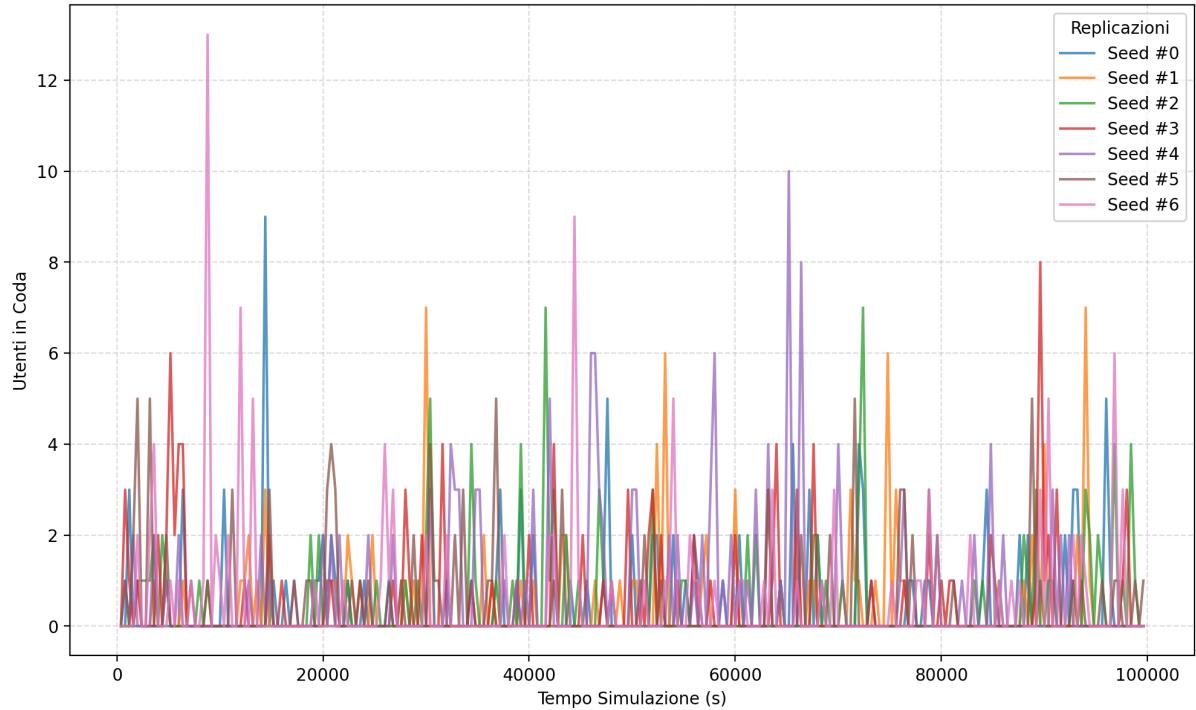
L'analisi del transitorio iniziale è stata condotta con l'obiettivo di individuare il momento in cui il sistema migliorato raggiunge le condizioni di regime stazionario, e di determinare la lunghezza del periodo di warm-up da scartare nelle successive analisi statistiche. Il sistema

è stato quindi testato utilizzando un tasso di arrivo $\lambda_{med} = 0,127205 \text{ pax / s}$ e una durata della simulazione pari a $TRANSIENT_DURATION = 100000.0$.

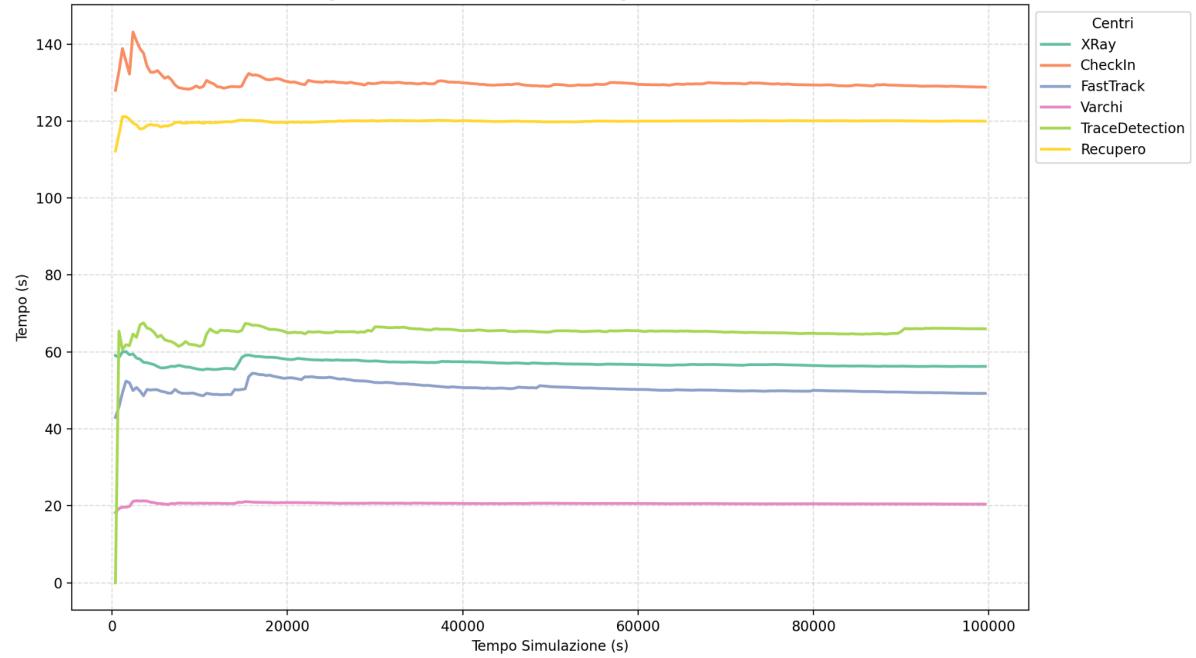
Come si evince dai grafici sottostanti le curve non presentano più una crescita indefinita, ma tendono a stabilizzarsi e ad oscillare intorno ad un valore medio costante, dopo una prima fase iniziale. Osservando l'andamento delle curve confermiamo la scelta di un tempo di taglio pari a $TIME_WARMUP = 60000$ secondi.

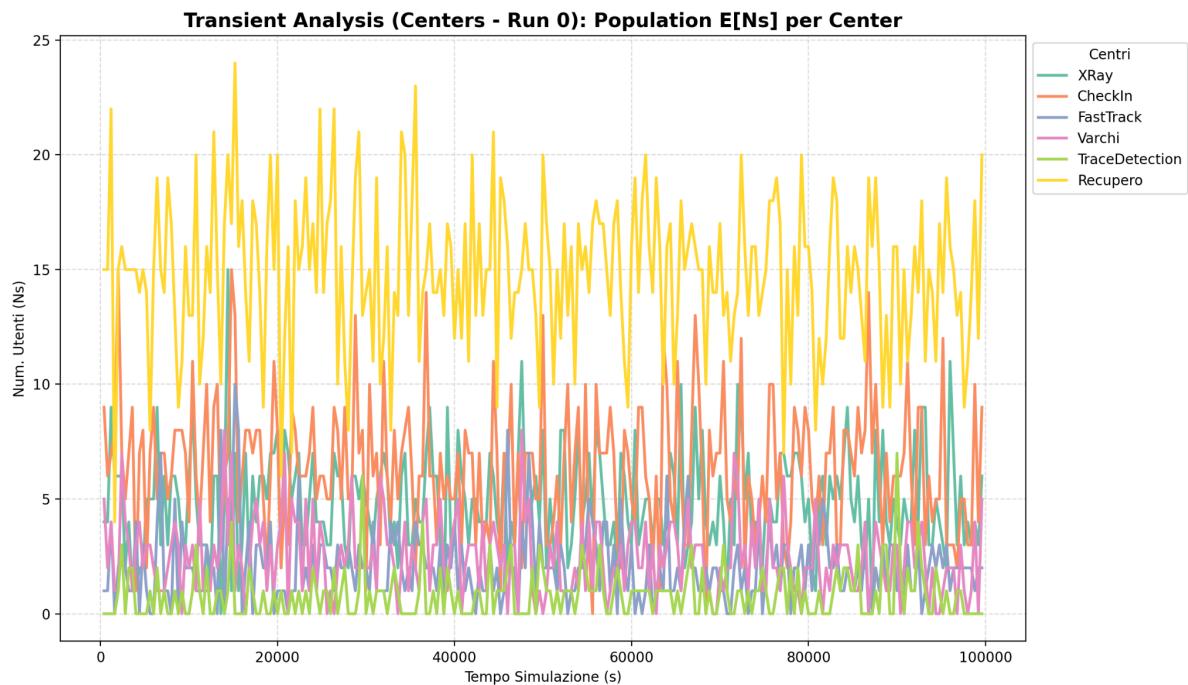


Transient Analysis (Seeds): XRay Queue Length E[Nq]



Transient Analysis (Centers - Run 0): Response Time E[Ts] per Center





9.6.2 - Simulazione ad orizzonte finito

9.6.2.1 - Analisi del collo di bottiglia

Analogamente a quanto fatto per il modello base, prima di procedere con la simulazione, effettuiamo l'analisi operazionale per verificare se, con la nuova topologia e il ridimensionamento delle risorse, il carico di lavoro per server soddisfi la condizione di stabilità.

Ricordiamo che la domanda di servizio scalata per server va calcolata come:

$$\frac{D_i}{m_i} = \frac{V_i \cdot E[S_i]}{m_i}$$

Dove:

- **Banchi Accettazione:** $m_1 = 8$, $E[S_1] = 120$ s
- **Varchi Elettronici:** $m_2 = 4$, $E[S_2] = 15$ s
- **Controllo X-Ray:** $m_3 = 6$, $E[S_3] = 60$ s
- **Trace Detection:** $m_4 = 2$, $E[S_4] = 60$ s
- **Fast Track:** $m_6 = 3$, $E[S_6] = 50$ s

In base alle equazioni di traffico abbiamo:

- $V_1 = p_{desk} = 0,3872$
- $V_2 = 1$
- $V_3 = V_2 \cdot p_{slow} = 0,67$

- $V_4 = 0,1$
- $V_6 = V_2 \cdot p_{fast} = 0,33$

Quindi:

- **Banchi Accettazione:** $\frac{0,3872 \cdot 120}{8} s \approx 5,81 s$
- **Varchi Elettronici:** $\frac{1 \cdot 15}{4} s \approx 3,75 s$
- **Controllo X-Ray:** $\frac{0,67 \cdot 60}{6} s \approx 6,70 s$
- **Trace Detection:** $\frac{0,1 \cdot 60}{2} s \approx 3,00 s$
- **Fast Track:** $\frac{0,33 \cdot 50}{3} s \approx 5,50 s$
- **Recupero Oggetti:** è modellato come Infinite Server, pertanto il suo carico per server tende a 0 e non può fisicamente rappresentare un collo di bottiglia.

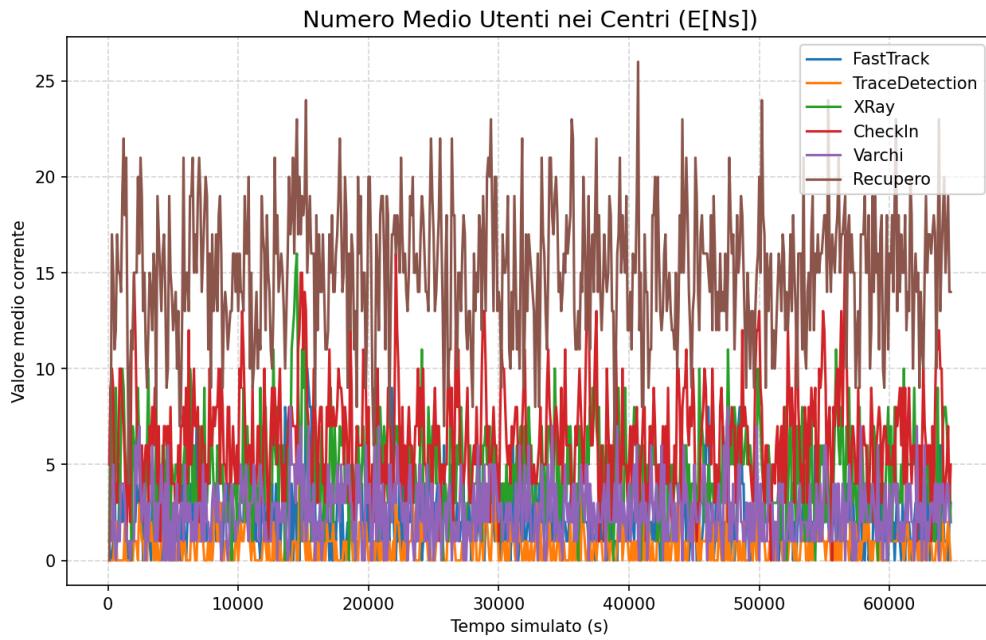
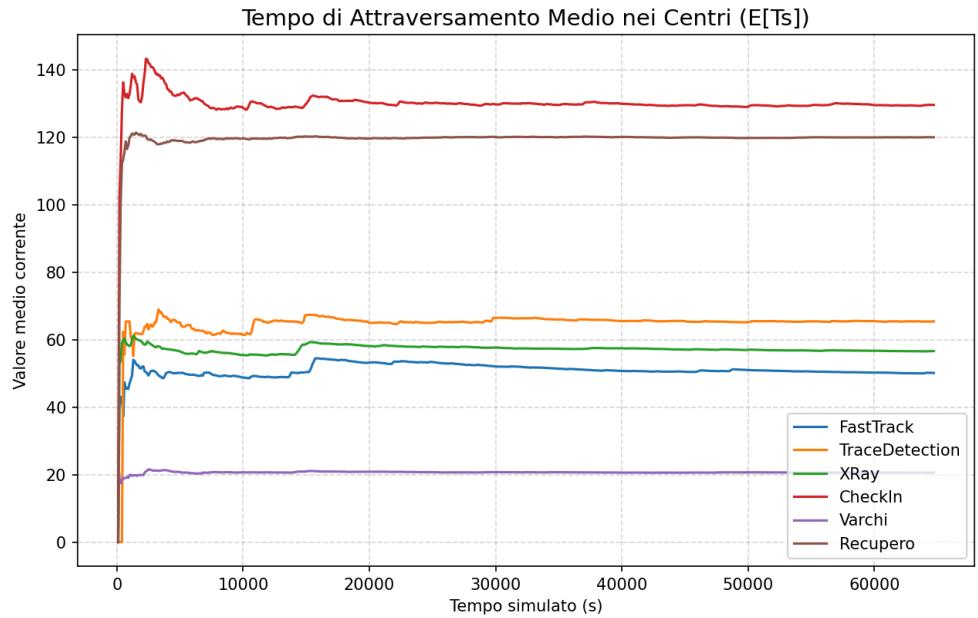
Possiamo quindi trarre le seguenti conclusioni:

- Il carico sul Centro 3 (XRay) è sceso a **6,70 s**, un valore inferiore alla soglia di stabilità ($E(r) = 1 / \lambda_{med} = 7,86 s$), dunque il centro non è più saturo.
- Con 3 server dedicati, il nuovo centro per il Fast Track presenta un carico di **5,50 s**, risultando pienamente sostenibile.
- Il raddoppio dei server per il Trace Detection ($m=2$) ha portato il carico a soli **3,00 s**, eliminando qualsiasi rischio di saturazione.

Poiché per tutti i centri vale la condizione $\frac{D_i}{m_i} \leq \frac{1}{\lambda_{med}}$, l'analisi statica prevede che il sistema sia pienamente stabile e in grado di gestire il flusso nominale di passeggeri.

9.6.2.2 - Risultati della simulazione operativa

Confermata teoricamente l'eliminazione del collo di bottiglia grazie all'analisi statica, si è proceduto con la simulazione su un Orizzonte Finito di 18 ore ($WORK_DAY = 64800$), applicando il carico $\lambda_{med} = 0,127205$ pax/s. Per ottenere risultati statisticamente rilevanti, è stato mantenuto l'approccio delle **Replicazioni Indipendenti** ($n=64$), calcolando gli Intervalli di Confidenza (IC) al 95% tramite distribuzione t-Student. La metodologia di calcolo dell'errore e dell'ampiezza dell'intervallo (w) rimane esattamente quella già esposta per lo scenario base. Di seguito vengono presentati i risultati ottenuti, che evidenziano un miglioramento delle prestazioni globali del sistema:



Nello specifico, a differenza dello scenario Base in cui il centro raggi X accumulava code ingestibili, nel modello migliorativo **tutti i centri mostrano code medie trascurabili**. In particolare, il centro X-Ray, precedentemente critico, presenta ora una coda media composta da soli 0.45 passeggeri, con un numero totale di passeggeri nel centro pari a circa 4,79. Il nuovo centro di Fast Track e il centro di Trace Detection potenziato dimostrano di assorbire in maniera efficace il flusso in arrivo.

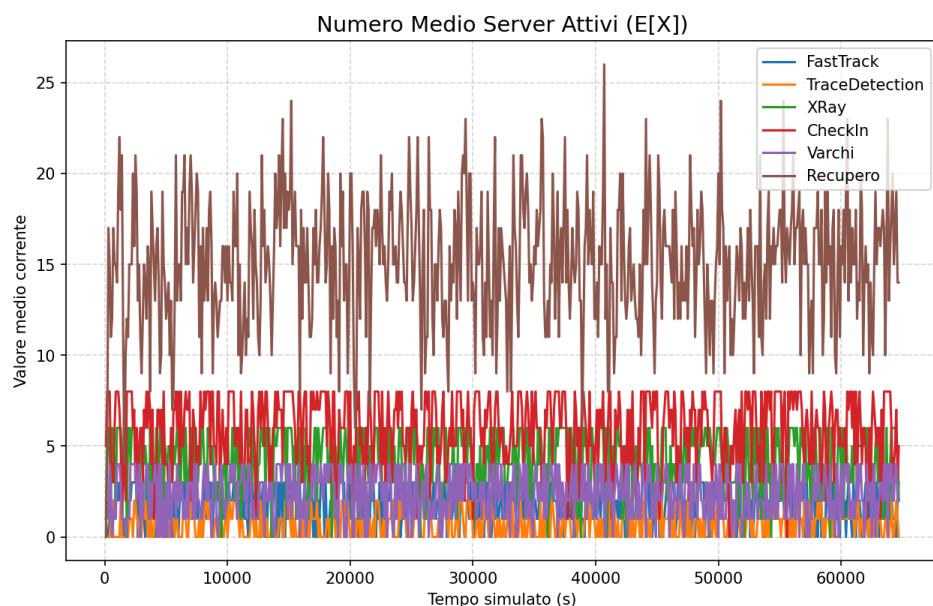
Numero Utenti in Coda (Nq) - Confidenza 95%

| Center | Mean | Width | Min | Max |
|-----------------------|----------|----------|----------|----------|
| CheckIn | 0.511386 | 0.024629 | 0.486757 | 0.536016 |
| FastTrack | 0.262473 | 0.007516 | 0.254957 | 0.269989 |
| Recupero | 0.0 | 0.0 | 0.0 | 0.0 |
| TraceDetection | 0.068154 | 0.00269 | 0.065464 | 0.070844 |
| Varchi | 0.226266 | 0.004776 | 0.221491 | 0.231042 |
| XRay | 0.454277 | 0.016346 | 0.437932 | 0.470623 |

Numero Utenti nel Sistema (Ns) - Confidenza 95%

| Center | Mean | Width | Min | Max |
|-----------------------|-----------|----------|-----------|-----------|
| CheckIn | 6.385066 | 0.04886 | 6.336206 | 6.433926 |
| FastTrack | 2.048213 | 0.012484 | 2.035729 | 2.060697 |
| Recupero | 15.169146 | 0.042496 | 15.126651 | 15.211642 |
| TraceDetection | 0.830729 | 0.008257 | 0.822472 | 0.838986 |
| Varchi | 2.614209 | 0.010596 | 2.603614 | 2.624805 |
| XRay | 4.791958 | 0.029419 | 4.762539 | 4.821377 |

L'efficienza della nuova configurazione è confermata anche dall'analisi del numero medio di server occupati, dove per gli XRay non si osserva più un valore fisso pari a 6:



Tutto questo si riflette sui tempi in coda e di risposta dei vari centri:

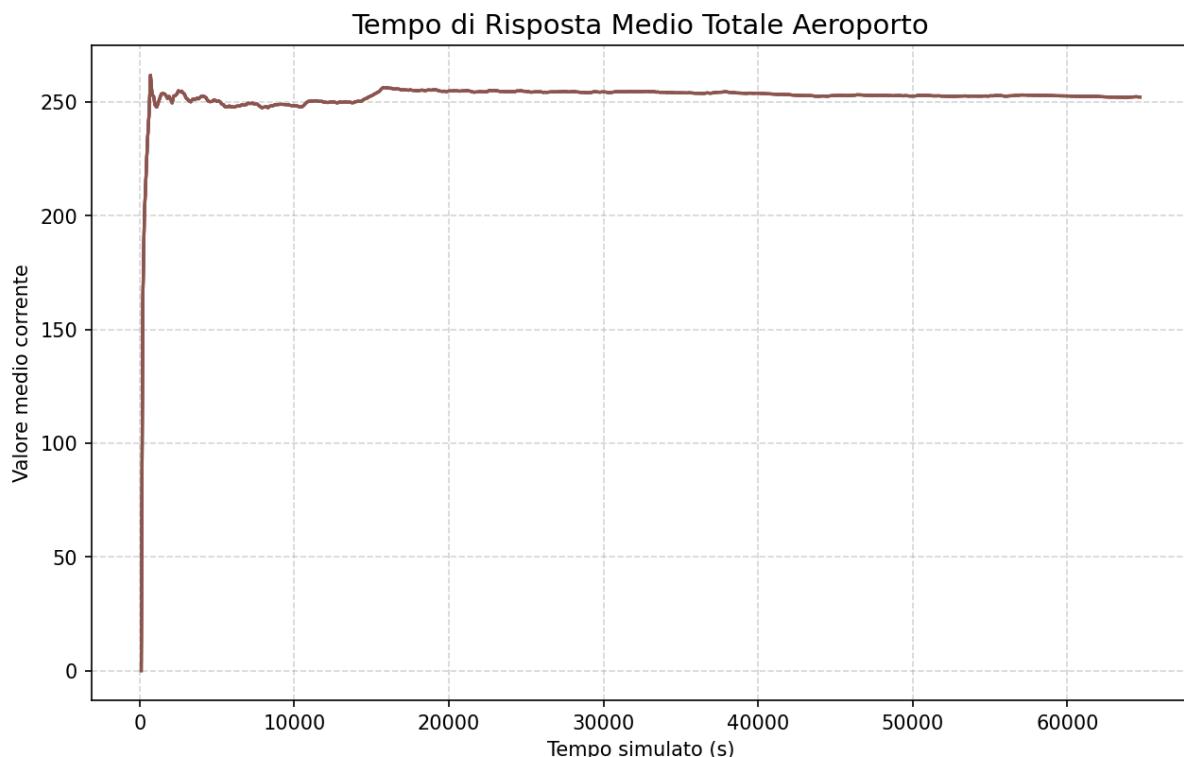
Tempo di Attesa in Coda (Tq) - Confidenza 95%

| Center | Mean | Width | Min | Max |
|-----------------------|-----------|----------|----------|-----------|
| CheckIn | 10.392918 | 0.468206 | 9.924712 | 10.861124 |
| FastTrack | 6.260389 | 0.169347 | 6.091042 | 6.429736 |
| Recupero | 0.0 | 0.0 | 0.0 | 0.0 |
| TraceDetection | 5.337109 | 0.186901 | 5.150208 | 5.52401 |
| Varchi | 1.78202 | 0.034679 | 1.747341 | 1.816698 |
| XRay | 5.341139 | 0.180178 | 5.160961 | 5.521318 |

Tempo di Risposta (Wait+Svc) (Ts) - Confidenza 95%

| Center | Mean | Width | Min | Max |
|-----------------------|------------|----------|------------|------------|
| CheckIn | 130.022651 | 0.506814 | 129.515838 | 130.529465 |
| FastTrack | 48.876974 | 0.184292 | 48.692682 | 49.061266 |
| Recupero | 119.866592 | 0.062301 | 119.804291 | 119.928892 |
| TraceDetection | 65.167126 | 0.218662 | 64.948464 | 65.385788 |
| Varchi | 20.597672 | 0.042756 | 20.554916 | 20.640428 |
| XRay | 56.399562 | 0.191353 | 56.20821 | 56.590915 |

Di conseguenza, il tempo di risposta del sistema, che nel modello base sfiorava i 50 min, ora nello scenario migliorativo crolla a una media di **4,2 min**:



Tempo Totale Attraversamento (SystemResponseTime) - Confidenza 95%

| Center | Mean | Width | Min | Max |
|---------|------------|----------|------------|------------|
| Success | 251.218328 | 0.396696 | 250.821632 | 251.615023 |

Dunque, il tempo trascorso in coda, che prima costituiva oltre il 90% del tempo totale, è ora ridotto a pochi secondi per ogni centro. Questo conferma che l'introduzione del percorso di Fast Track e l'adeguamento del Trace Detection hanno permesso di rendere il sistema capace di garantire il rispetto dei requisiti di qualità e di sicurezza.

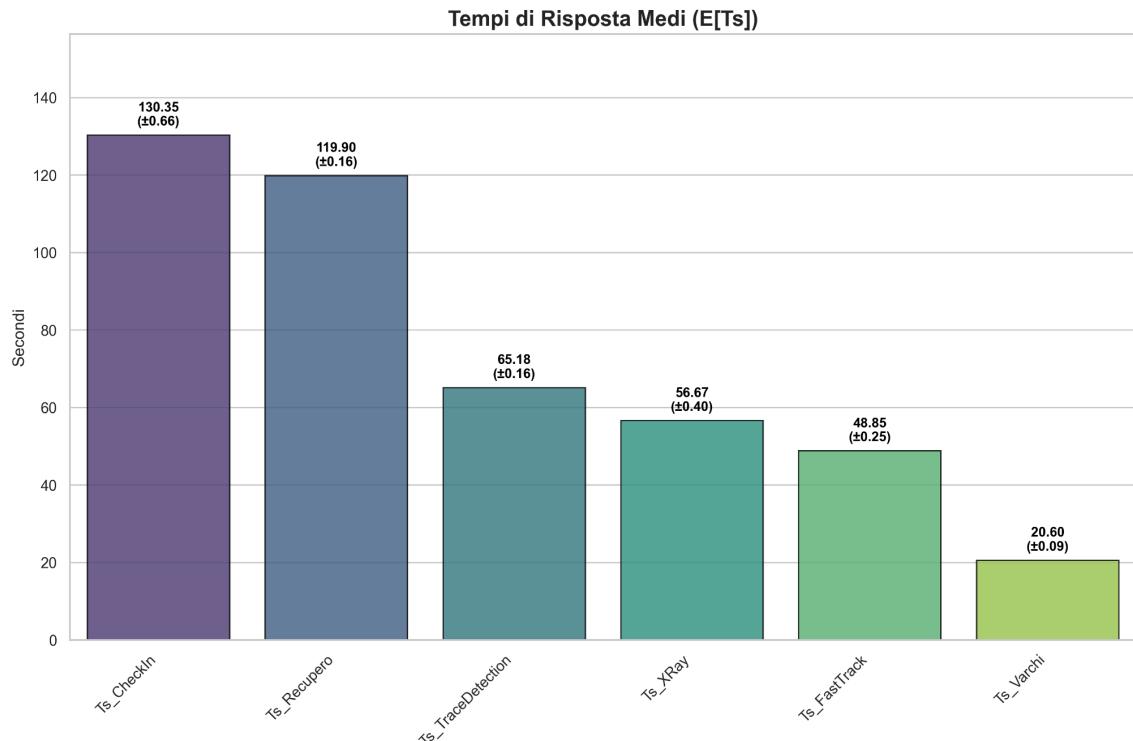
9.6.3 - Simulazione ad orizzonte infinito

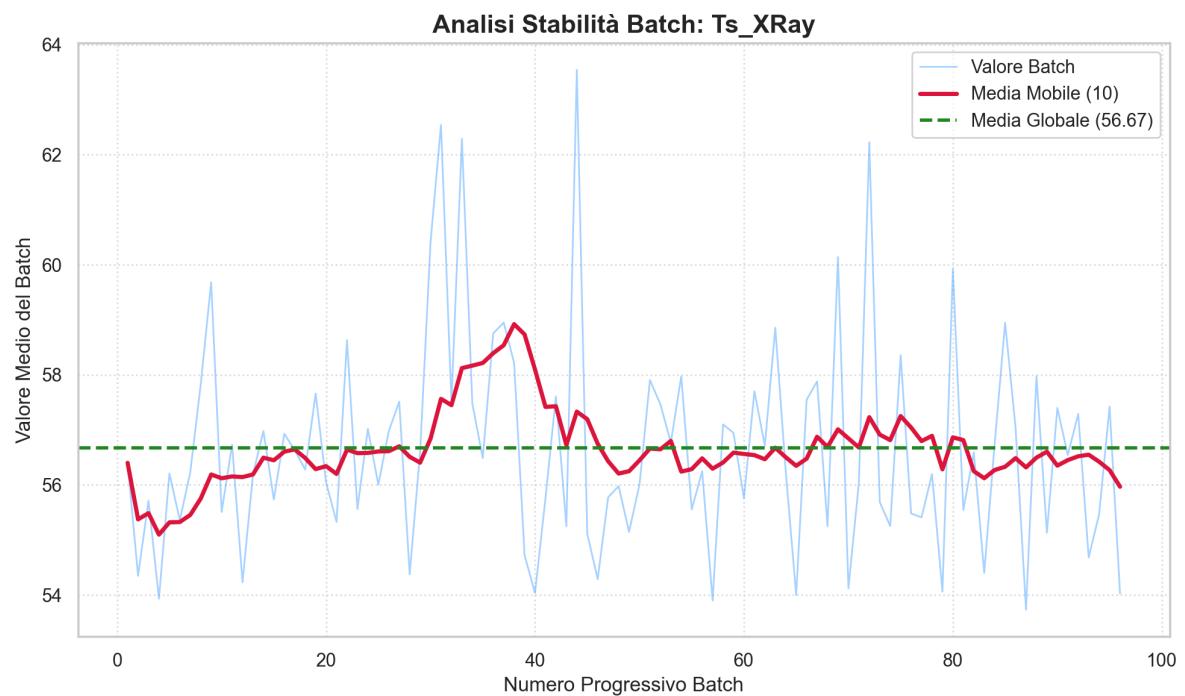
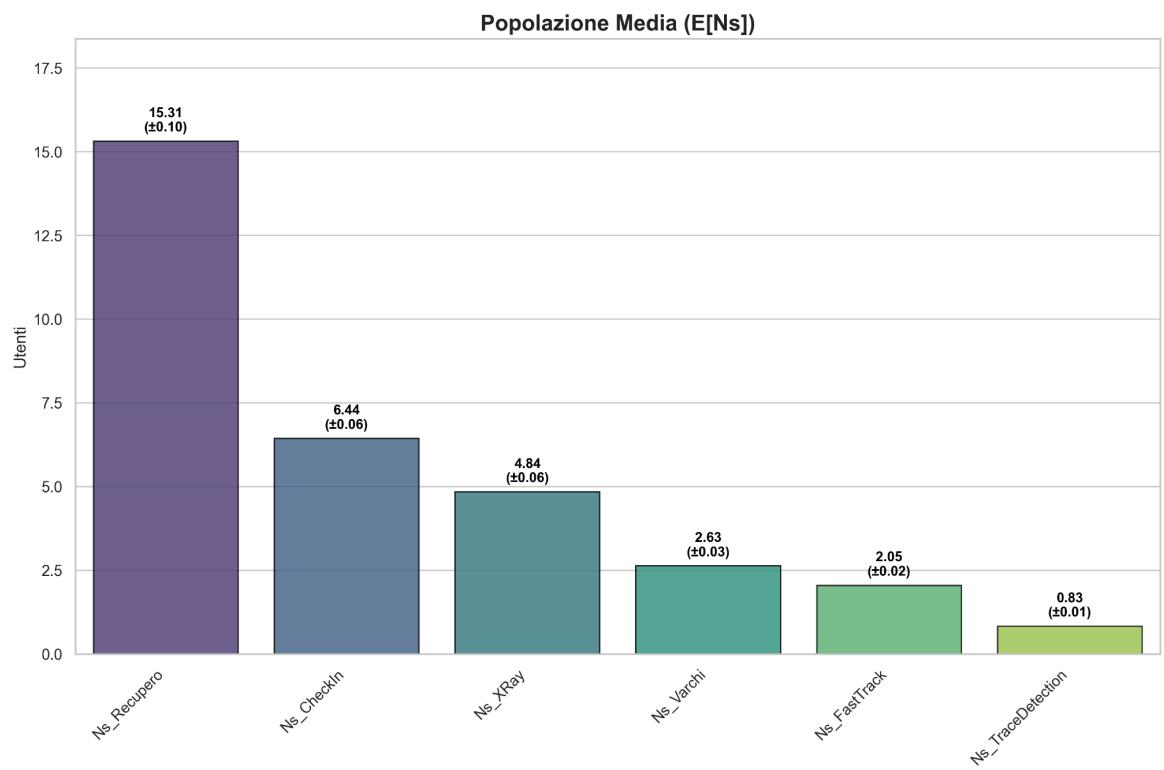
A completamento dell'analisi sperimentale, è stata eseguita la simulazione a orizzonte infinito. Come specificato nei capitoli precedenti, questa configurazione non mira a replicare il comportamento di una giornata lavorativa, ma è stata utilizzata per le fasi di Verifica e Validazione per confermare che il modello è stato costruito in maniera corretta. Tuttavia, esiste una differenza sostanziale rispetto allo scenario base, grazie agli interventi migliorativi, infatti, non è necessario dimezzare il flusso λ_{med} per ottenere un sistema stabile.

E' stato quindi applicato nuovamente il metodo delle Batch Means ottenendo valori di autocorrelazione trascurabili ($< |0, 2|$ per tutte le metriche) con i seguenti parametri:

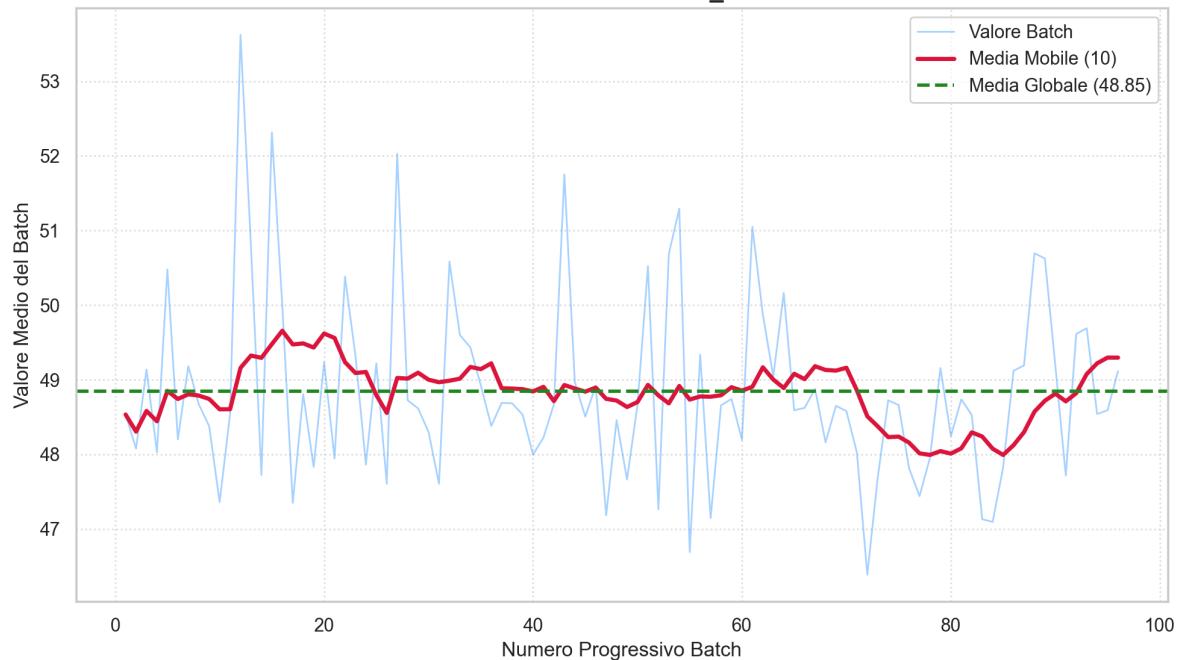
- **Lambda:** λ_{med} .
- **Tempo di Taglio (T_{cut}):** 60.000 secondi (warm-up per rimuovere il *bias* iniziale).
- **Dimensione Batch (b):** 1080 osservazioni per batch.
- **Numero di Batch (k):** 96 batch.

I risultati dello Steady State a pieno carico mostrano un sistema perfettamente stabile:

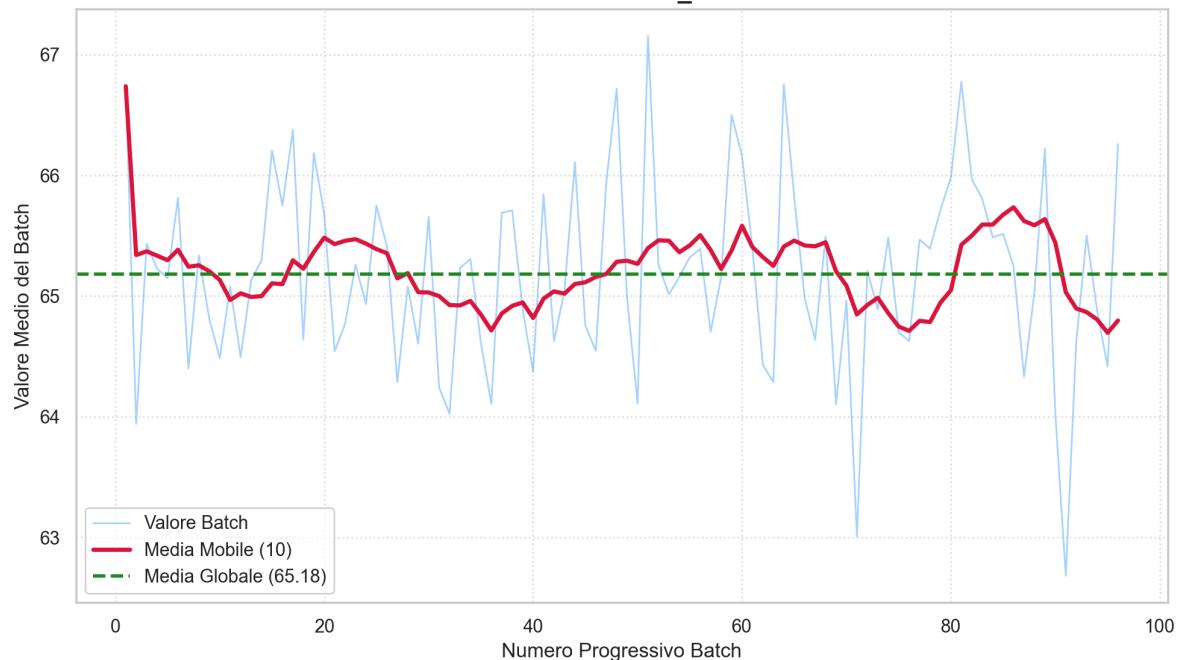




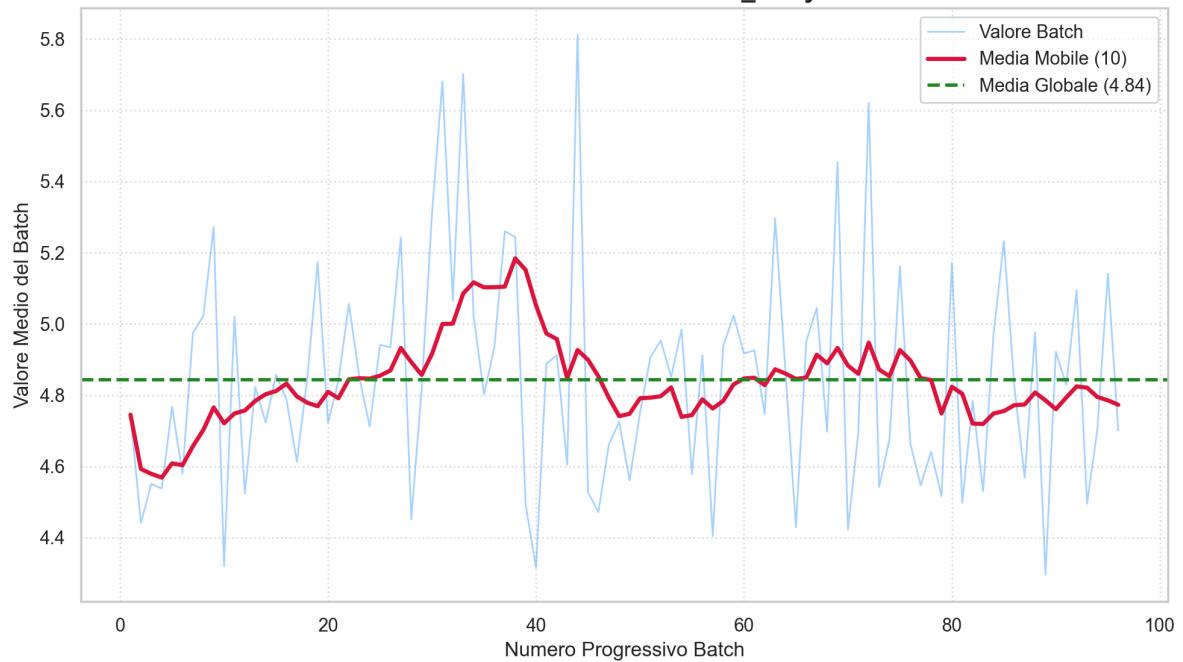
Analisi Stabilità Batch: Ts_FastTrack



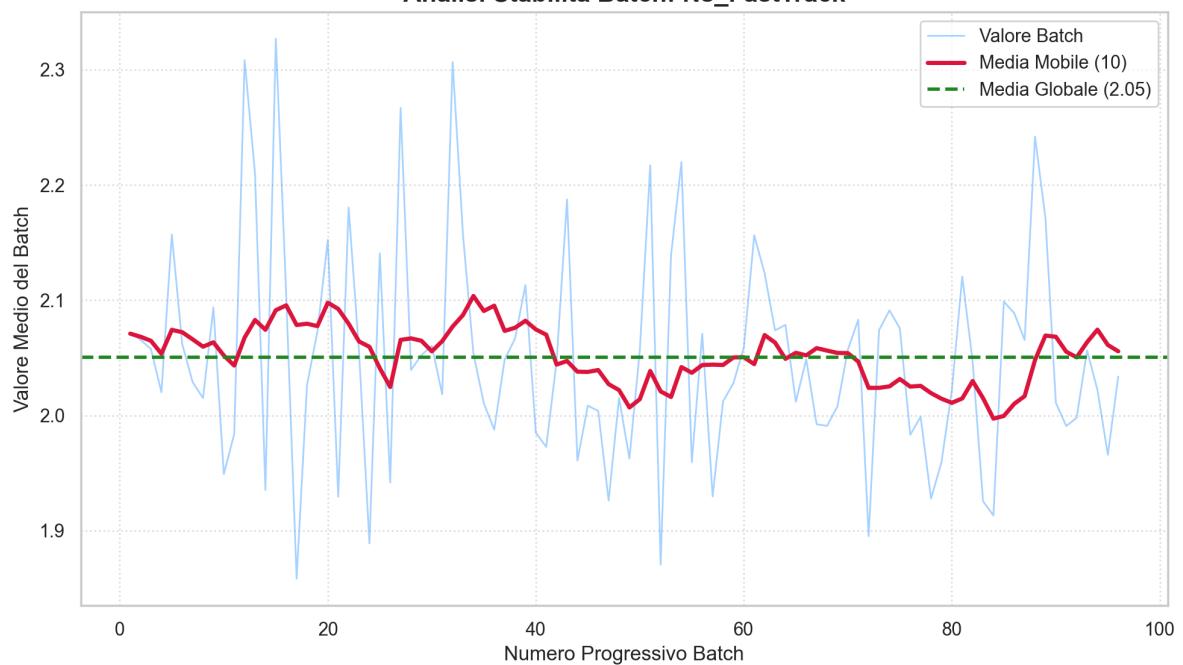
Analisi Stabilità Batch: Ts_TraceDetection

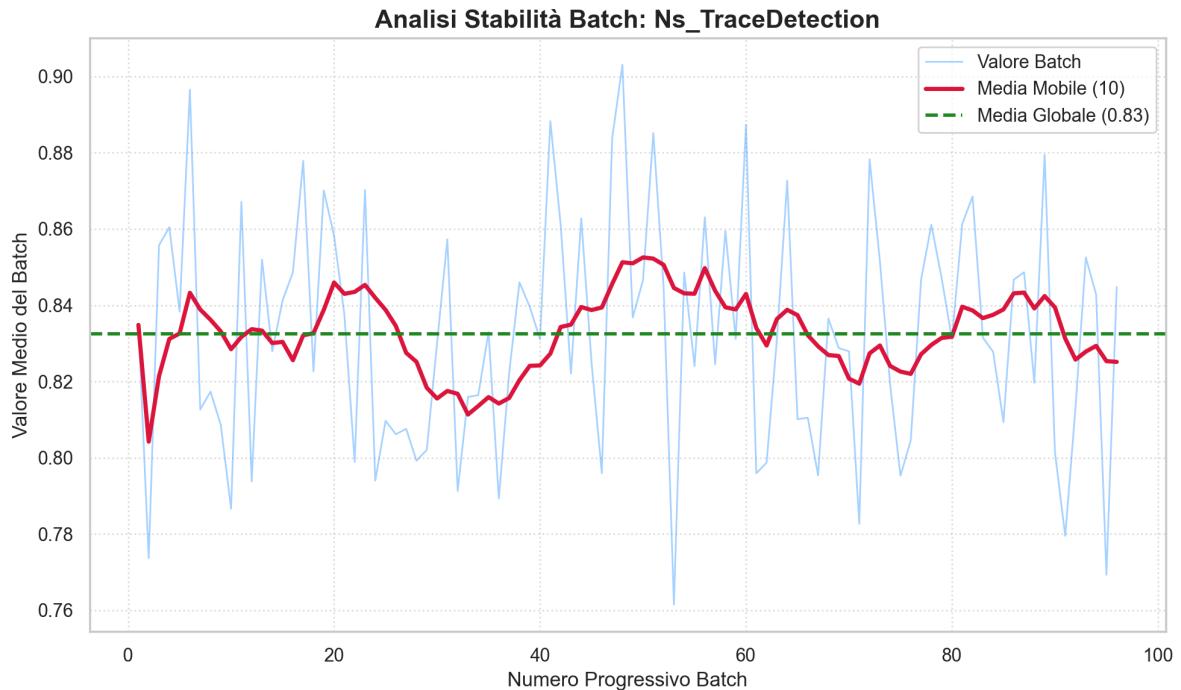


Analisi Stabilità Batch: Ns_XRay



Analisi Stabilità Batch: Ns_FastTrack





Questa simulazione conferma che il collo di bottiglia è stato rimosso e la nuova configurazione risulta robusta.

10 - Conclusioni

Le analisi condotte durante lo studio hanno permesso di validare la robustezza logica e statistica del simulatore, quantificando l'impatto degli interventi sulla gestione dei passeggeri nell'aeroporto di Ciampino.

Il confronto tra lo scenario base e lo scenario migliorativo evidenzia come le criticità iniziali siano state completamente risolte:

1. **Analisi criticità modello base:** le simulazioni condotte hanno confermato l'esistenza di un grave collo di bottiglia presso i Controlli a Raggi X. Tale configurazione rendeva il sistema incapace di gestire il flusso medio di passeggeri λ_{med} , evidenziato nel modello di specifica, come rappresentativo del carico di lavoro medio durante le giornate lavorative di alcuni mesi di picco. Questo portava alla violazione dei requisiti di qualità del servizio e di sicurezza proposti:
 - **Violazione tempi di attesa:** il tempo medio di coda agli X-Ray ($E[T_{Q3}] \approx 45,3$ minuti) superava ampiamente la soglia tollerabile di **15 minuti**.
 - **Violazione affollamento:** la popolazione media nella zona controlli ($E[N_{S_controlli}] = E[N_{S3}] + E[N_{S4}] \approx 325 + 2$ persone ≈ 327 persone) oltrepassava il limite di sicurezza fissato a **240 persone**.

- **Inefficienza globale:** sebbene il tempo totale di transito (49,5 min) rientrasse nei limiti teorici di 120 min e nell'obiettivo proposto di 80 min, oltre il 90% di tale tempo veniva speso in coda, offrendo un servizio pessimo.
2. **Efficacia delle soluzioni proposte nello scenario migliorativo:** l'introduzione del percorso Fast Track (con un instradamento del 33% dei passeggeri) e il potenziamento del Trace Detection (da 1 a 2 server) hanno trasformato le prestazioni del sistema. I risultati ottenuti confermano il soddisfacimento degli obiettivi prefissati:
- **Obiettivo 1:** il tempo di attesa ai controlli a raggi X è crollato a circa 5 secondi, ampiamente sotto i 15 minuti richiesti.
 - **Obiettivo 2:** il numero medio di persone nella zona controlli, costituita dal centro a Raggi X, il centro di Fast Track e il centro di Trace Detection, si aggira intorno al valore medio di 7,67 passeggeri ($E[N_{s3}] + E[N_{s4}] + E[N_{s6}]$).
 - **Obiettivo 3:** il tempo medio di attraversamento dell'intero sistema è sceso da 49,5 minuti a soli 4,2 minuti, garantendo un'alta efficienza.

Riferimenti Bibliografici

- [1] Aeroporti di Roma (ADR), *"Regolamento di Scalo - Aeroporto G.B. Pastine Ciampino"*, Revisione 2, Ultimo aggiornamento 16 Ottobre 2025. Disponibile online: https://www.adr.it/documents/d/guest/reg_cia_rev2_16-10-2025-pdf
- [2] Ministero dell'Interno, *"D.M. 17 luglio 2014 - Regola tecnica di prevenzione incendi per la progettazione, la costruzione e l'esercizio delle attività di aerostazioni con superficie coperta accessibile al pubblico superiore a 5.000 m²"*, pubblicato in Gazzetta Ufficiale n. 173 del 28 luglio 2014. Disponibile online: <https://mauromalizia.it/wp-content/uploads/Aerostazioni.pdf>
- [3] Aeroporti di Roma (ADR), *"Carta dei Servizi 2025 - Aeroporto Giovan Battista Pastine Ciampino"*, Guida all'Aeroporto, Maggio 2025. Disponibile online: https://www.enac.gov.it/app/uploads/2025/05/CartaServizi_CIA_2025_250513.pdf
- [4] Ryanair Group, *"Annual Report 2024"*, 2024. Disponibile online: <https://it.scribd.com/document/842783247/Ryanair-2024-Annual-Report#:~:text=The%20Ryanair%20Annual%20Report%20for,challenges%20and%20pursuing%20sustainability%20initiatives.>
- [5] Aeroporti di Roma (ADR), *"Allegato II: Previsioni di Traffico al 2044"*, Contratto di Programma 2012-2021, Ottobre 2012. Disponibile online: https://www.enac.gov.it/app/uploads/2024/04/Allegato_11_Previsioni_di_traffico_al_2044_1.pdf
- [6] IATA, *"Annual Review 2023"*, Safety & Security Statistics, p. 15, 2024. Disponibile online: <https://www.iata.org/contentassets/c81222d96c9a4e0bb4ff6ced0126f0bb/annual-review-2023.pdf>
- [7] ENAC, *"Safety Review 2024"*, Sezione Unruly Passengers, Roma, Maggio 2025. Disponibile online: <https://www.enac.gov.it/news/safety-review-2025/>