

Simulazione dell'aeroporto di Ciampino e valutazione della sua efficienza

Progetto di Performance Modeling of Computer Systems and Networks, a.a. 2024/2025

Università degli Studi di Roma “Tor Vergata”

Dennis Mariani (0365494)

Belli Stefano (0350116)

Agenda

1. Introduzione
2. Obiettivi
3. Modello base
4. Modello migliorativo

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)**.



- L'aeroporto di Ciampino rappresenta uno scalo primario per il traffico **low-cost** e **turistico**
- Questo comporta la possibilità di una forte **congestione del sistema**

Obiettivi dello studio

- Il tempo medio di attesa in coda ai controlli a raggi X non deve superare i **15 min**
- Il numero di passeggeri in prossimità dei controlli di sicurezza non deve superare il limite di **240 persone**
- Il tempo medio totale di completamento del percorso minore di **80 min**

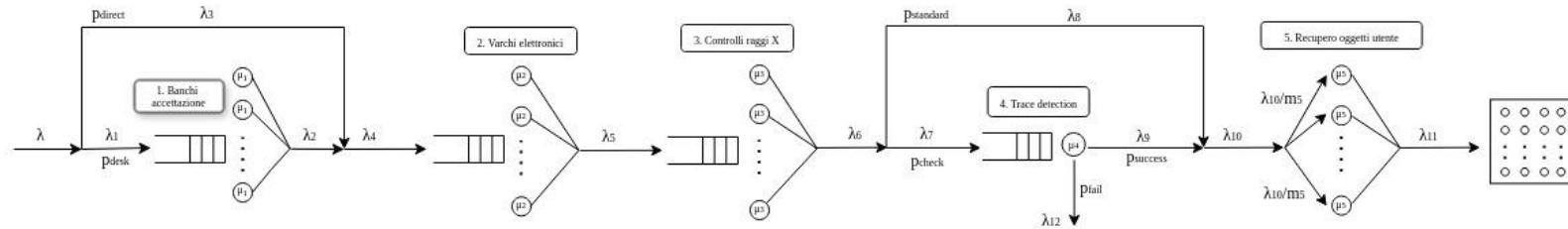
Gli obiettivi si basano su documenti ufficiali di AdR

Modello concettuale

Diagramma del sistema

Diagramma del sistema

Modello concettuale



Le probabilità di routing sono: p_{desk} p_{direct} $p_{standard}$ p_{check} $p_{success}$ p_{fail}

Lo stato è identificato da: $S(t) = (N_1, N_2, N_3, N_4, N_5)$

Gli eventi possibili sono: di tipo arrival, departure e sampling per ogni centro

Modello di specifica

1. Matrice di routing e equazioni di traffico
2. Modellazione dei singoli centri

Matrice di routing e equazioni di traffico

Modello di specifica

- **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

Le equazioni di traffico:

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

$$\lambda_2 = \lambda_1$$

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

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

$$\lambda_5 = \lambda_4$$

$$\lambda_6 = \lambda_5$$

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

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

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

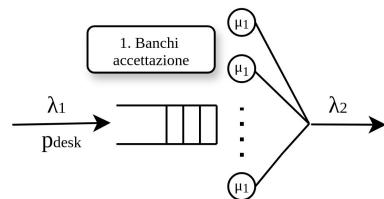
$$\lambda_{10} = \lambda_8 + \lambda_9$$

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

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

Modellazione dei singoli centri

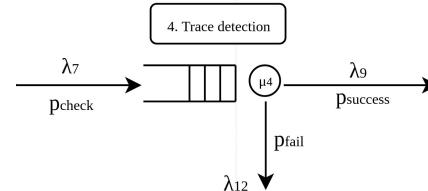
Modello di specifica



MSSQ, tempo di servizio modellato con una **normale troncata** di parametri:

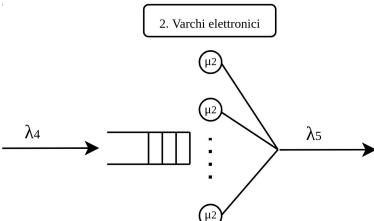
- Media: 120s
- DevStd: 60s
- LB: 60s
- UB: 180s

8 server



SSSQ, tempo di servizio modellato con una **normale troncata** di parametri:

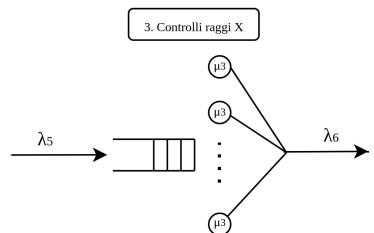
- Media: 60s
- DevStd: 20s
- LB: 30s
- UB: 90s



MSSQ, tempo di servizio modellato con una **normale troncata** di parametri:

- Media: 15s
- DevStd: 15s
- LB: 10s
- UB: 30s

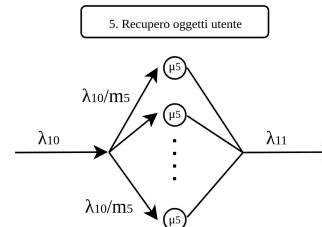
4 server



MSSQ, tempo di servizio modellato con una **normale troncata** di parametri:

- Media: 60s
- DevStd: 30s
- LB: 30s
- UB: 70s

6 server



IS, tempo di servizio modellato con una **normale troncata** di parametri:

- Media: 120s
- DevStd: 30s
- LB: 60s
- UB: 180s

Modello computazionale

1. Eventi, event queue e simulation clock
2. Implementazione dei centri

Eventi, event queue e simulation clock

Modello computazionale

Gli eventi sono rappresentati dalla seguente classe Java:

```
public final class Event implements Comparable<Event> {
    private final double t;
    private final EventType type;
    private final Job job;
    private final Center targetCenter;
    private final Object args;
```

- **t** è il tempo in cui avverrà l'evento
- **type** è una enum che può assumere valori ARRIVAL, DEPARTURE o SAMPLING,
- **job** è l'oggetto che ha causato l'evento
- **targetCenter** è il centro interessato dall'evento (es. arrivo al centro x)
- **args** sono argomenti opzionali

```
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) {
        /*...*/
        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;
    }
}
```

- Gli eventi sono contenuti in una priority queue, che estrae basandosi sul tempo di accadimento **t** più basso.
- Il metodo **add()** aggiunge un evento alla pq.
- **pop()** rimuove il più imminente evento dalla pq
- Il “**currentClock**” definito come attributo di istanza della classe è il simulation clock

Il *simulation clock* ha come valore corrente il **t** del *last processed event*

Implementazione dei centri

Modello computazionale

```
public class SingleServerSingleQueue extends Center {  
    private boolean activeServer;  
    private Queue<Job> jobQueue = new LinkedList<>();  
  
    public SingleServerSingleQueue(  
        int id,  
        String name,  
        ServiceProcess serviceProcess,  
        NetworkRoutingPoint networkRoutingPoint,  
        StatCollector statCollector,  
        SampleCollector sampleCollector,  
        BatchCollector batchCollector) {  
        /*...*/  
    }  
  
    @Override  
    public void onArrival(Event event, EventQueue eventQueue) {  
        double now = eventQueue.getCurrentClock();  
  
        /*...*/  
    }  
  
    @Override  
    public void onDeparture(Event event, EventQueue eventQueue) {  
        double now = eventQueue.getCurrentClock();  
  
        /*...*/  
    }  
  
    private void scheduleDepartureEvent(  
        double now, Job job, EventQueue eventQueue) {  
  
        /*...*/  
  
        double svc = serviceProcess.getService();  
        Event departureEvent = new Event(  
            now + svc, EventType.DEPARTURE, this, job, null);  
  
        eventQueue.add(departureEvent);  
    }  
}
```

I centri sono implementati in classi concrete (`mbpmcsn.center.SingleServerSingleQueue`, ...) che ereditano dalla classe astratta `mbpmcsn.center.Center` e implementano i metodi `onArrival`, `onDeparture` e `onSampling` invocati tramite l'event handler, quando un evento destinato al centro si verifica, quindi lo gestisce.

Il metodo `scheduleDepartureEvent` mostra esempio di come viene creato un nuovo evento e aggiunto alla event queue.

Verifica

1. Problema dell'utilizzazione maggiore di 1
2. Metodologia
3. Risultati della verifica

Problema dell'utilizzazione maggiore di 1

Verifica

- Vogliamo eseguire la verifica del sistema appena costruito, ovvero: *abbiamo costruito il modello correttamente?*
- Tuttavia, osserviamo come il centro di controllo a raggi X abbia un'utilizzazione maggiore di 1 (come previsto, è un bottleneck per il sistema): i teoremi delle reti di code (es. Burke) e le varie formule non funzionano più.

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

- Comunque riteniamo opportuno verificare che il simulatore funzioni correttamente, quindi la soluzione adottata è simulare con tempi di interarrivo raddoppiati per alleviare il carico e permettere a tutti i centri della rete di avere utilizzazione < 1: $\lambda_{ver} = \frac{\lambda_{med}}{2} = 0.063602 \text{ pax/s}$

Metodologia

Verifica

- Confronto tra tempi servizio esponenziali
- Tempi interarrivo raddoppiati
- Simulazione ad orizzonte infinito
 - Intervallo di confidenza al 95%
 - Batch means
 - B=1080
 - K=96
 - Autocorrelazione < |0.2|
- Confronto centro-per-centro
- Teoremi della teoria delle code
- Formule M/M/k
- Verifica del rispetto dei controlli di consistenza:
 $E(T_{S,k}) = E(T_{Q,k}) + E(S_{i,k})$
 $E(N_{S,k}) = E(N_{Q,k}) + m_k \cdot \rho_k$
 $0 < \rho < 1$

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

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

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

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

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

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

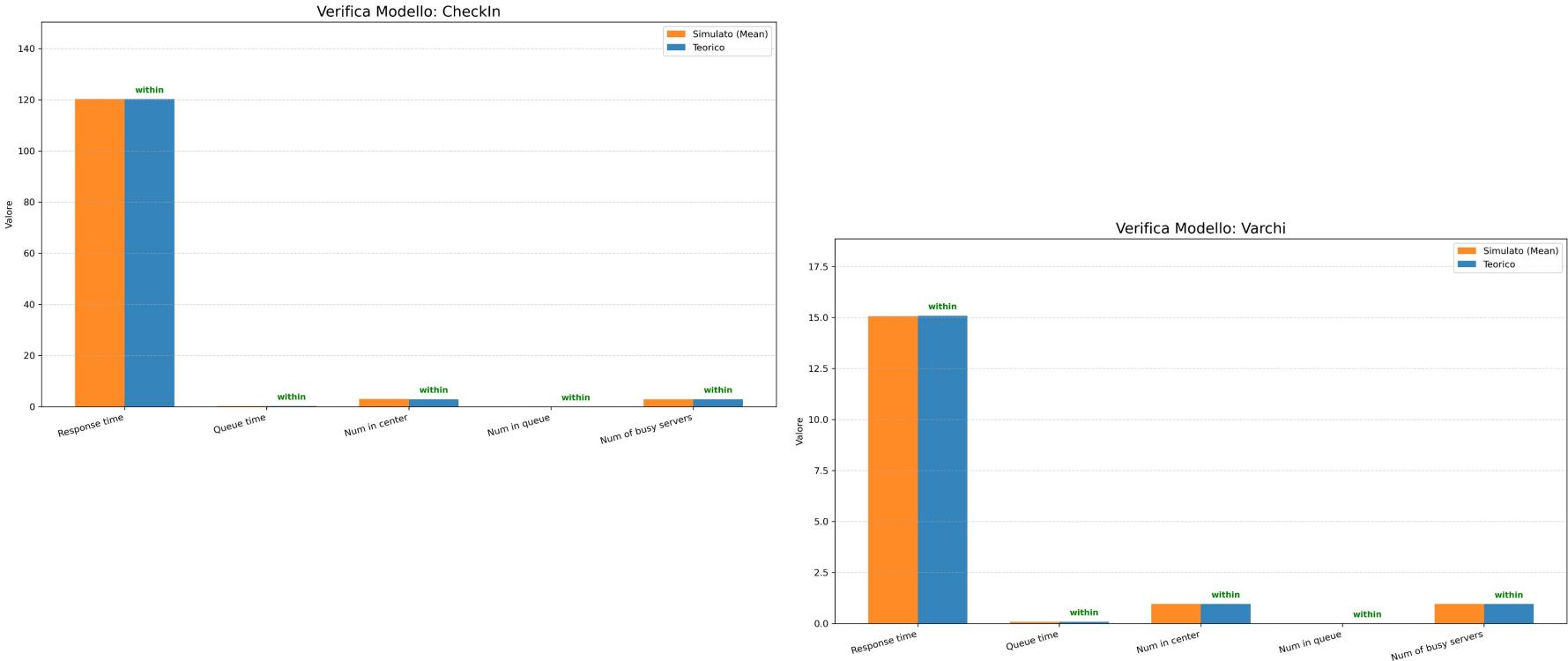
$$E(N_{Q,1}) = \lambda_{ver} \cdot p_{desk} \cdot E(T_{Q,1}) = 0,0070$$

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

$$m_1 \rho_1 = 2,9551$$

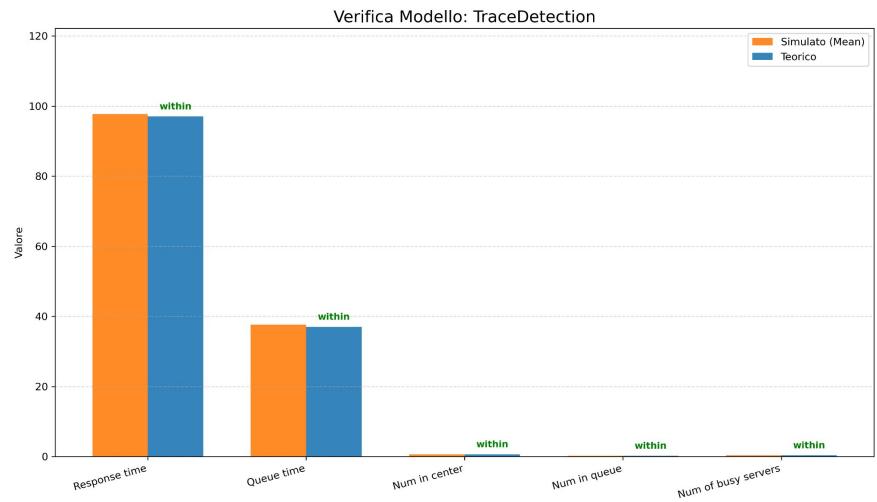
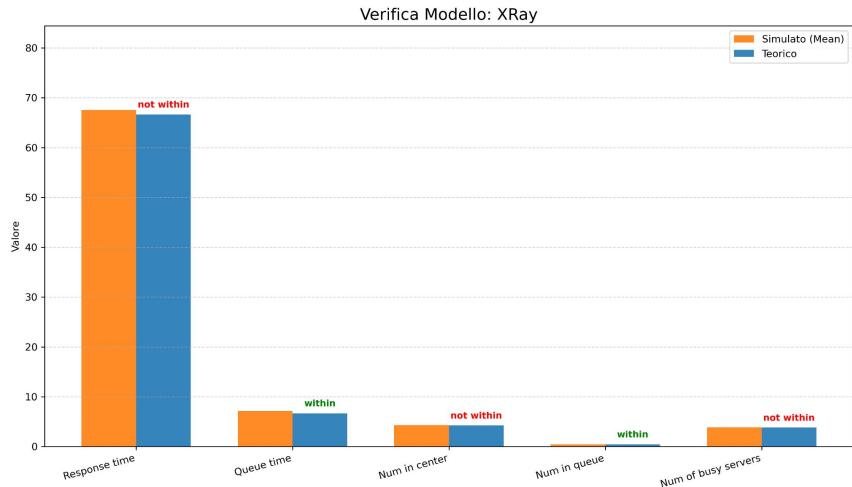
Risultati della verifica

Verifica



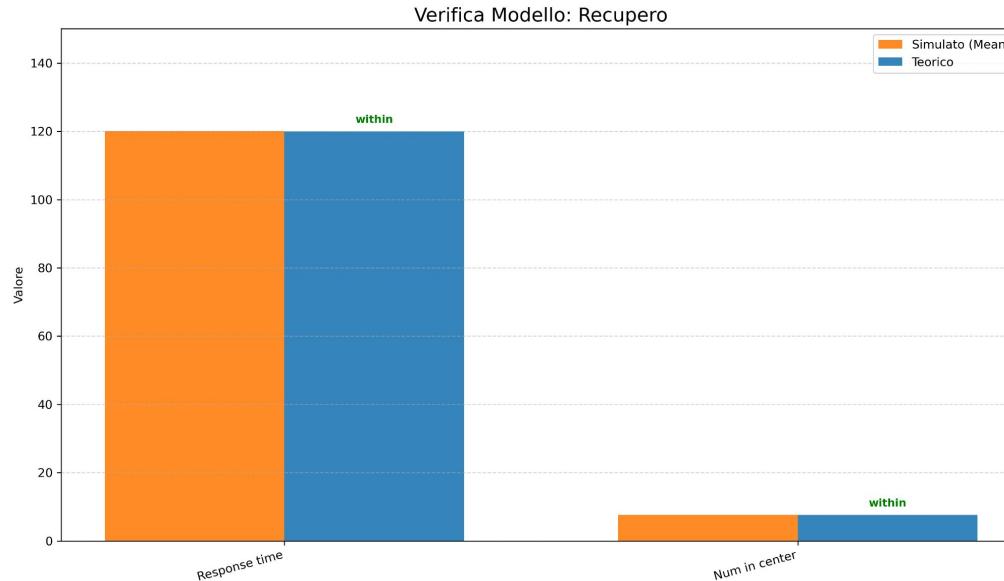
Risultati della verifica (2)

Verifica



Risultati della verifica (3)

Verifica



Validazione

1. Verifica delle probabilità di routing, volumi di traffico
2. Valutazione macroscopica

Verifica delle probabilità di routing, volumi di traffico

Validazione

- **Verifica delle probabilità di instradamento:** controlliamo 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, da cui abbiamo stimato che
 - 38.72% al check-in
 - 10% selezionato per controlli

Totale 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%

- **Verifica dei volumi di traffico:** controlliamo la consistenza del generatore di arrivi rispetto ai parametri di input usando il parametro $\lambda_{med} = 0,127205 \text{ pax / s}$ e $T = 18 \text{ h} = 64800 \text{ s}$

$N_{atteso} = \lambda_{med} \cdot T = 8242 \text{ passeggeri}$, il risultato della simulazione è 8168 pax processati (orizz. finito)

Valutazione macroscopica

Validazione

- **Verifica dei parametri di servizio:** controlliamo se il motore di simulazione genera dei tempi di servizio che siano in accordo con le distribuzioni teoriche presentate nel modello di specifica. Evitiamo di entrare nei dettagli ma il risultato è in generale **positivo**.
- **Valutazione macroscopica (legge di Little):** consideriamo l'intero aeroporto come black box in condizione di stabilità (quindi è necessario $\lambda_{med} / 2$) applichiamo la legge di Little su di esso: confrontiamo $E[Ns]_{teorico}$ e è 15, 6329 *passeggeri* con quello simulato che è di 15, 6731 *passeggeri*

Design degli esperimenti

1. Analisi del transitorio
2. Simulazione ad orizzonte finito

Analisi del transitorio

Design degli esperimenti

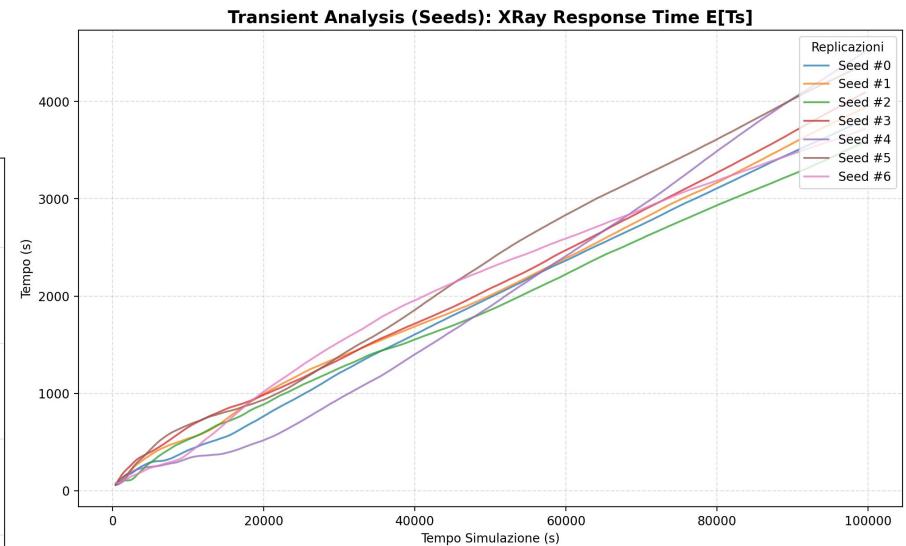
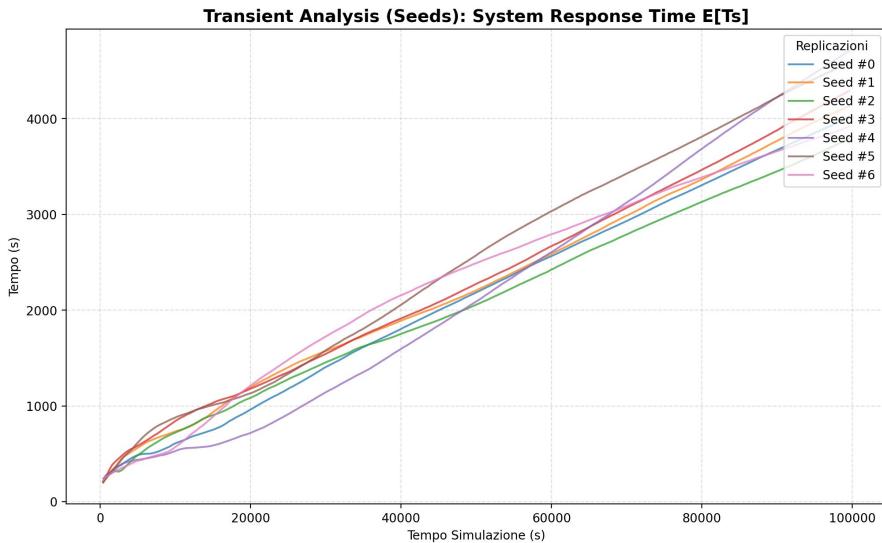
- Con l'analisi del transitorio rispondiamo alla domanda: il sistema raggiunge mai lo stato stazionario? Se sì, quando?
- In questa analisi ci concentriamo sul **tempo medio di risposta del sistema**, **tempo medio di risposta del centro di controllo a raggi X** (già identificato come possibile bottleneck) e il **numero medio di job in coda in quest'ultimo centro**
- Eseguiamo una run molto lunga e sfruttiamo la tecnica delle replicazioni indipendenti

Risultati dell'analisi del transitorio

Design degli esperimenti → Analisi del transitorio

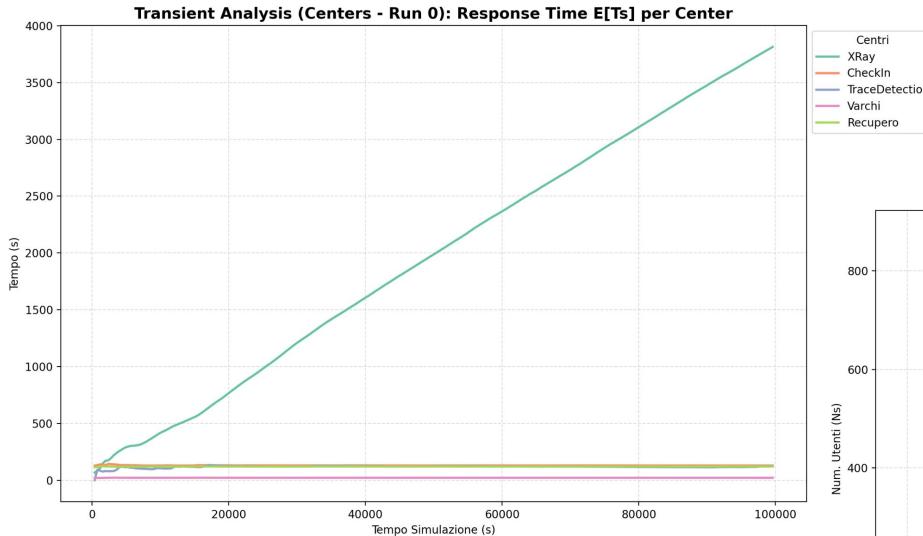
- Concludiamo che con un $\lambda_{med} = 0,127205 \text{ pax / s}$ il sistema **non converge** e la causa è, appunto, come previsto sia da noi, che dalla verifica, il centro dei controlli a raggi X:

Focus sul tempo di risposta del sistema e del centro a raggi X al variare dei seed (limitati a 6)

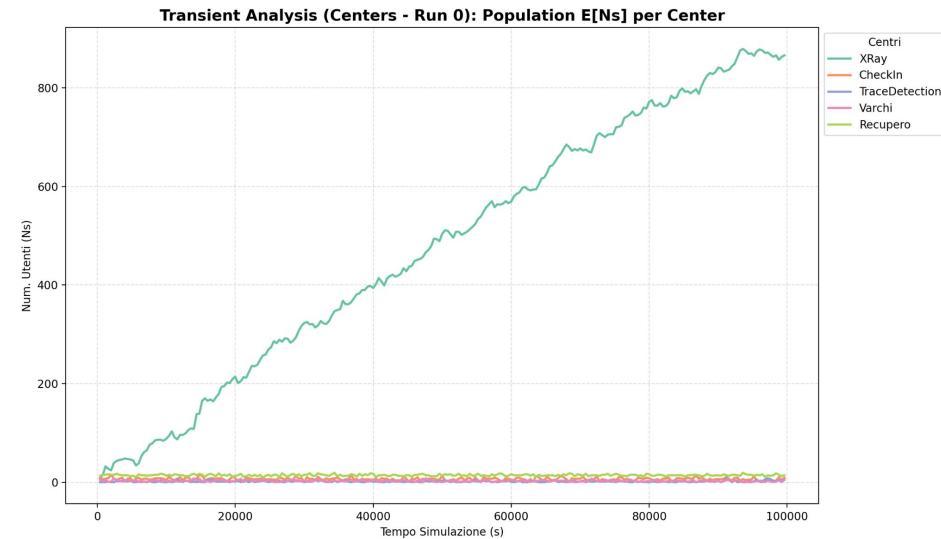


Risultati dell'analisi del transitorio (2)

Design degli esperimenti → Analisi del transitorio



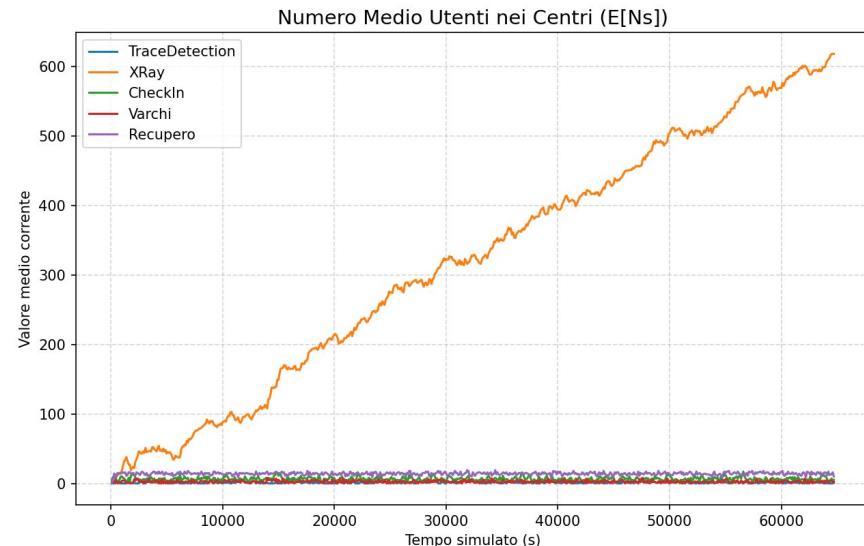
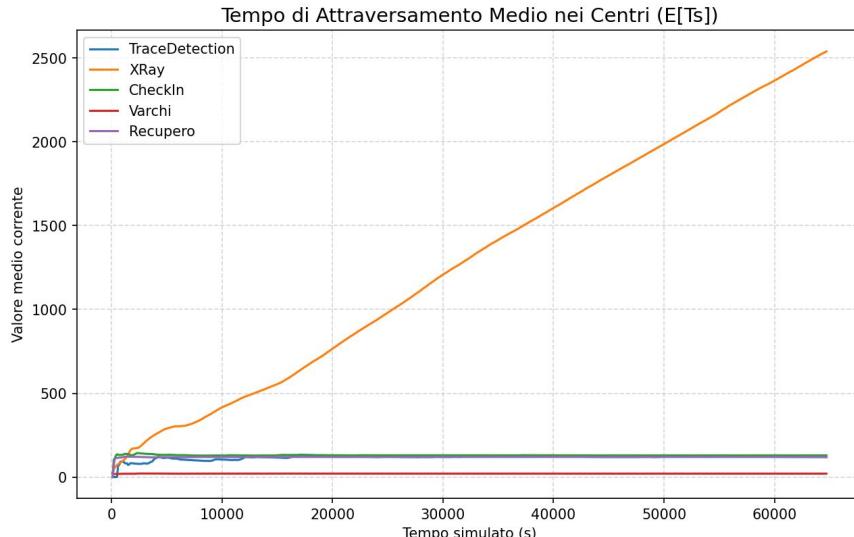
Osserviamo come l'influenza degli altri centri sia pressoché nulla sul sistema complessivo



Simulazione ad orizzonte finito

Design degli esperimenti

- Si sceglie, come orizzonte, quello finito perché meglio rappresenta il caso di studio (una giornata lavorativa simulata, 18h = 64800s)
- Replicazioni indipendenti (64)
- Intervalli di confidenza al 95%
- Persiste il problema del centro a raggi X



Risultati della simulazione ad orizzonte finito

Design degli esperimenti → Simulazione ad orizzonte finito

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

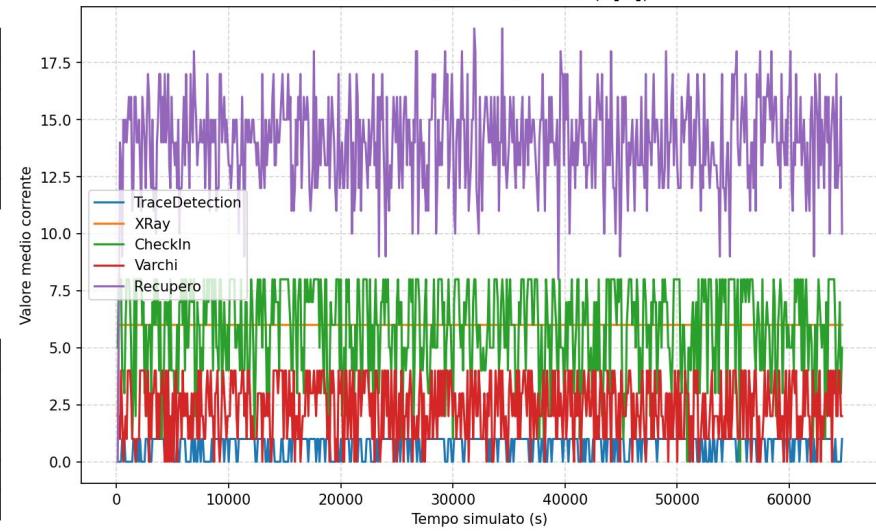
Tempo di Attesa in Coda (Tq) - Confidenza 95%

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

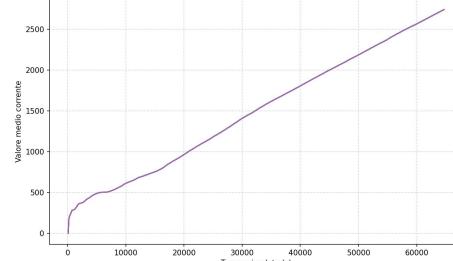
Tempo Totale Attraversamento (SystemResponseTime) - Confidenza 95%

Center	Mean	Width	Min	Max
Success	2972.078868	103.961222	2868.117646	3076.040089

Numero Medio Server Attivi (E[X])



Tempo di Risposta Medio Totale Aeroporto



Introduzione al modello migliorativo

Modello migliorativo basato sul fast track

Introduzione al modello migliorativo

- Abbiamo constatato che il centro di controllo ai raggi X è **il collo di bottiglia**, quindi ci concentriamo sull'alleggerire il carico su questo centro.
- **Non possiamo fare cambiamenti radicali all'aeroporto**
- Sfruttiamo il meccanismo **fast track** che è oggi poco utilizzato e quindi trascurato nella trattazione del modello base.
 - Gli utilizzatori di questo path alternativo sono tipicamente i *frequent flyer*.
 - Comunque, gli utilizzatori del servizio devono necessariamente affrontare i controlli a raggi X
 - Ma va tenuto conto del fatto che *frequent flyer* sono tipicamente **più veloci** nelle operazioni di **preparazione** ai controlli di sicurezza
- Offrono questo tipo di servizio le compagnie aeree e/o gli aeroporti stessi.

Obiettivi del modello migliorativo

Introduzione al modello migliorativo

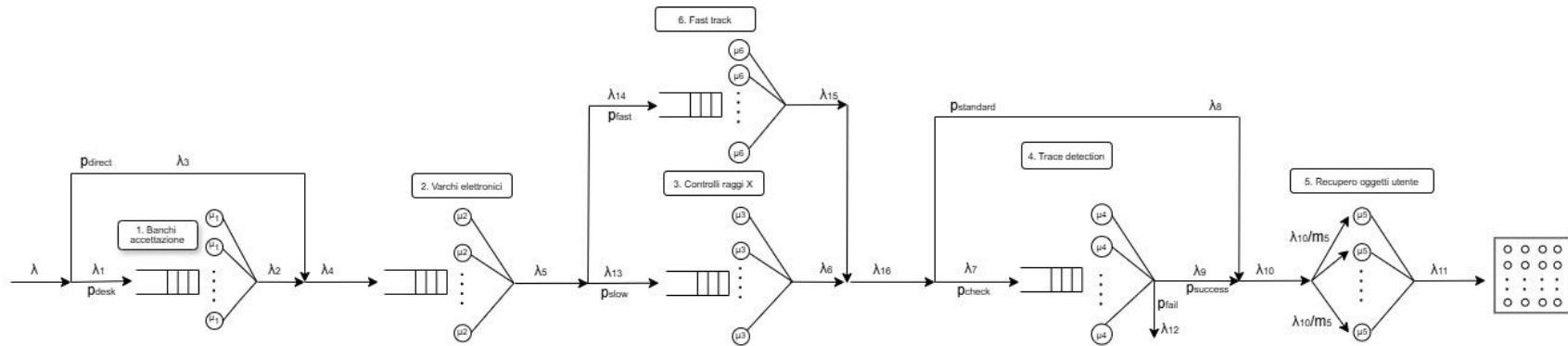
- Vogliamo dimostrare che aumentando il flusso di passeggeri che passano per il fast track (ad esempio con offerte), il carico sul centro dei controlli a raggi X diminuisce, eliminando così il collo di bottiglia
- Di conseguenza, al rispetto dei QoS riportati negli obiettivi.
- Ricordiamo che:
 - Il tempo medio di attesa in coda ai controlli a raggi X non deve superare i **15 min**
 - Il numero di passeggeri in prossimità dei controlli di sicurezza non deve superare il limite di **240 persone**
 - Il tempo medio totale di completamento del percorso minore di **80 min**
- **Nel modello base i primi due vengono violati, mentre il terzo viene rispettato**

Modello concettuale (migliorativo)

Diagramma del sistema

Diagramma del sistema

Modello concettuale (migliorativo)



Le nuove probabilità di routing sono: p_{fast} p_{slow}

Il nuovo stato è identificato da: $S(t) = (N_1, N_2, N_3, N_4, N_5, N_6)$

Nuovi eventi possibili arrival, departure e sampling per il centro 6

Modello di specifica (migliorativo)

1. Probabilità di routing per il fast track
2. Matrice di routing e equazioni di traffico
3. Modellazione dei singoli centri

Probabilità di routing per il fast track

Modello di specifica (migliorativo)

- Cruciale per l'eliminazione del bottleneck
- Non abbiamo dati a disposizione per decidere le probabilità di routing, quindi proponiamo, almeno inizialmente la seguente split:
 - $p_{fast} = 0.33 = \frac{1}{3}$
 - $p_{slow} = 1 - p_{fast} = 0.67 = \frac{2}{3}$
- Ci aspettiamo che il centro dei controlli a raggi X sia notevolmente più “leggero” rispetto al caso precedente, ridotto del 67%: da $\lambda_{med} \rightarrow \lambda_{med} \cdot 0.67$

Matrice di routing e equazioni di traffico

Modello di specifica (migliorativo)

- **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

Le equazioni di traffico:

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

$$\lambda_2 = \lambda_1$$

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

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

$$\lambda_5 = \lambda_4$$

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

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

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

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

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

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

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

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

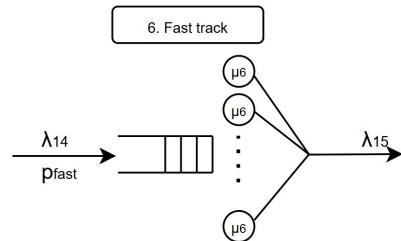
$$\lambda_{10} = \lambda_8 + \lambda_9$$

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

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

Modellazione dei singoli centri

Modello di specifica (migliorativo)

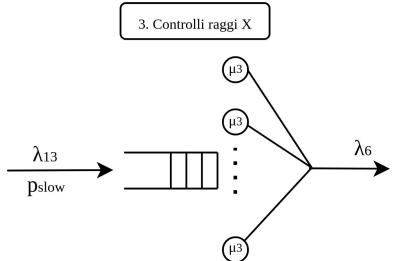


MSSQ, tempo di servizio modellato con una **normale troncata** di parametri:

- Media: 50s
- DevStd: 20s
- LB: 20s
- UB: 60s

3 server

Nuovo centro

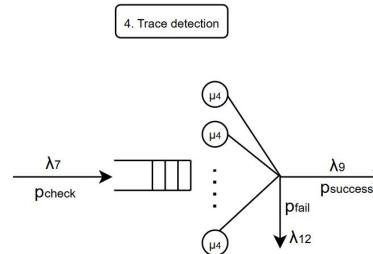


MSSQ, tempo di servizio modellato con una **normale troncata** di parametri:

- Media: 60s
- DevStd: 30s
- LB: 30s
- UB: 70s

6 server

Cambiato flusso degli arrivi



MSSQ, tempo di servizio modellato con una **normale troncata** di parametri:

- Media: 60s
- DevStd: 20s
- LB: 30s
- UB: 90s

2 server

Cambiata modellazione serventi

Modello computazionale (migliorativo)

Cambiamenti rispetto al modello base

Cambiamenti rispetto al modello base

Modello computazionale (migliorativo)

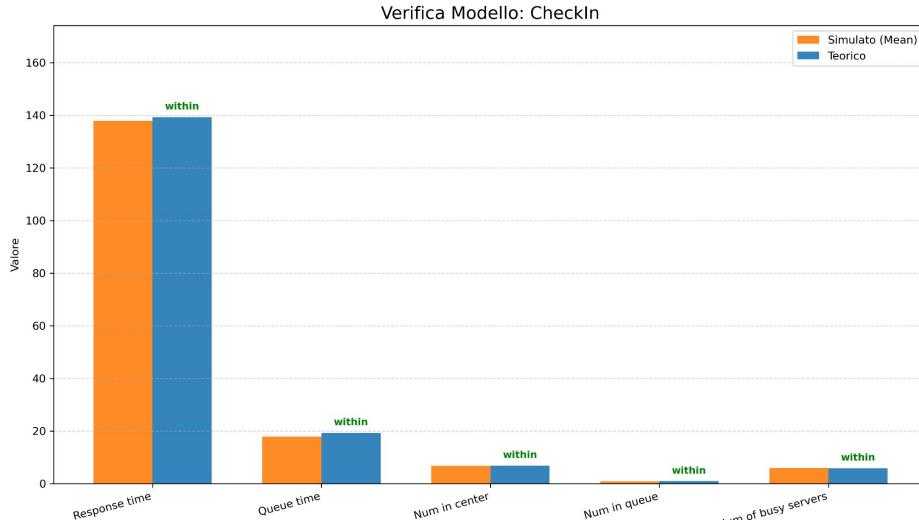
- Nessun cambiamento degno di nota, il sistema è rimasto praticamente invariato
 - Aggiunto un'attributo nella classe `Job` (e relativi getter/setter)
 - Creata nuova classe `VarchiRouting`
- E' stato sufficiente assemblare il nuovo modello in `ImprovedSimulationModel`

Verifica (migliorativo)

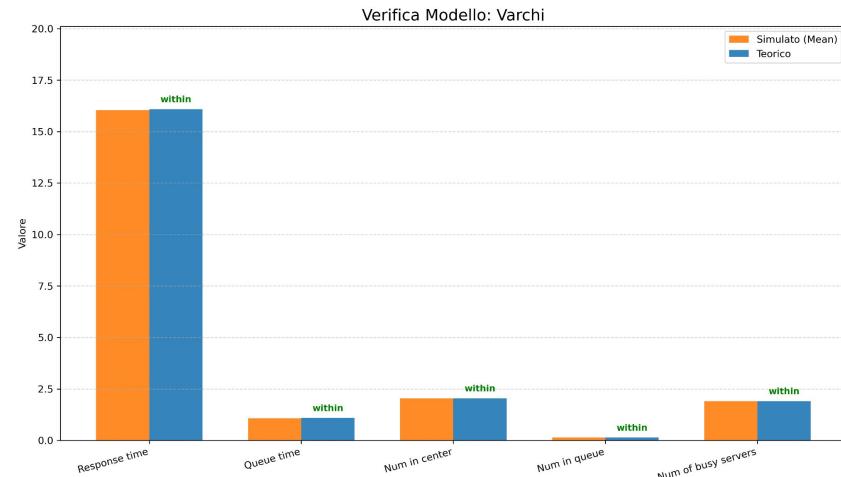
Risultati della verifica

Risultati della verifica

Verifica (migliorativo)



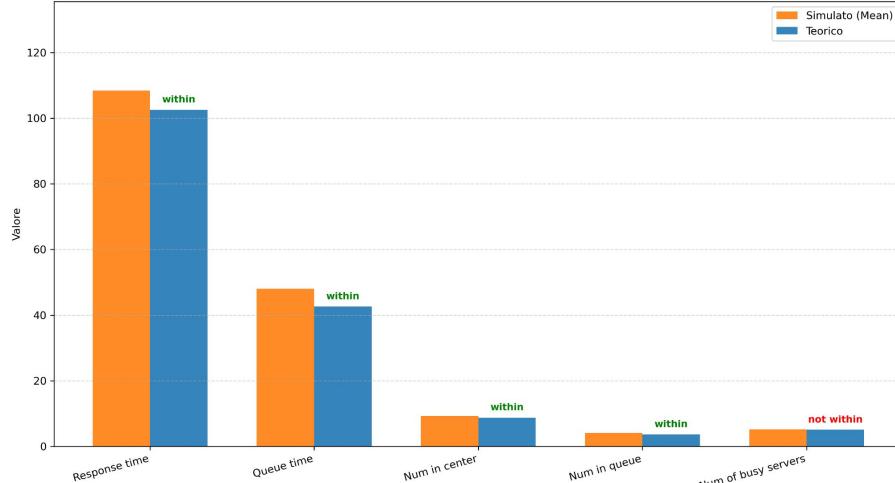
Metodologia di verifica identica
al caso base, solo che usiamo
tempi di interarrivo “regolari”



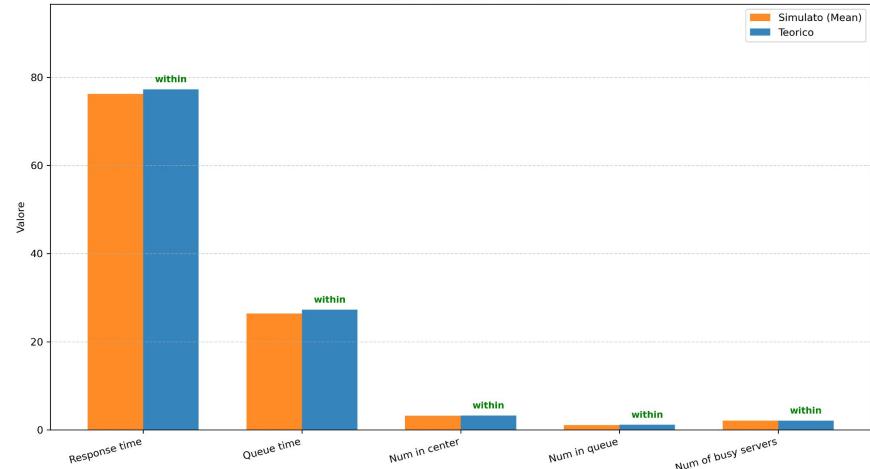
Risultati della verifica (2)

Verifica (migliorativo)

Verifica Modello: XRay

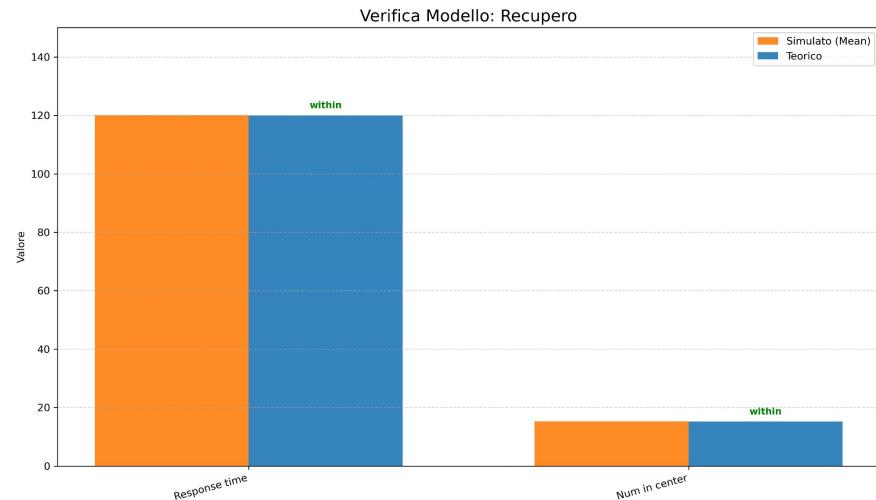
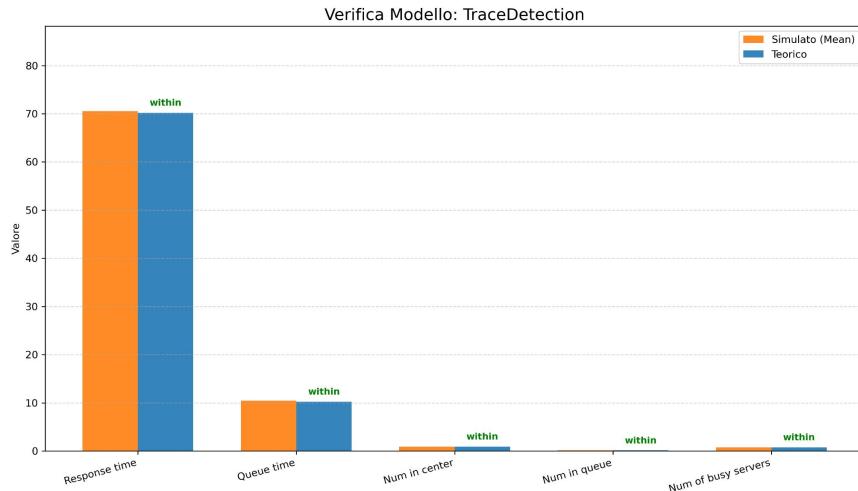


Verifica Modello: FastTrack



Risultati della verifica (3)

Verifica (migliorativo)



Validazione (migliorativo)

Risultati della validazione

Risultati della validazione

Validazione (migliorativo)

- **Verifica delle probabilità di instradamento:** andiamo a verificare la correttezza della nuova logica di routing introdotta, ovvero la biforcazione all'uscita dei Varchi Elettronici, dove il flusso deve dividersi tra il Fast Track e i controlli standard: **2705** sono stati instradati al **Fast Track** (0,3311), mentre **5464** sono stati instradati ai **Raggi X** (0,6689)
- **Verifica dei parametri di servizio:** controlliamo se il motore di simulazione genera dei tempi di servizio che siano in accordo con le distribuzioni teoriche presentate nel modello di specifica. Evitiamo di entrare nei dettagli ma il risultato è in generale **positivo**.
- **Valutazione macroscopica (legge di Little):** consideriamo l'intero aeroporto come black box in condizione di stabilità e applichiamo la legge di Little su di esso: confrontiamo $E[N_s]_{teorico}$ che è 32,0022 passeggeri con quello simulato che è di 32,1069 *passeggeri*
- **Verifica dei volumi di traffico:** controlliamo la consistenza del generatore di arrivi rispetto ai parametri di input usando il parametro $\lambda_{med} = 0,127205 \text{ pax / s}$ e $T = 18 \text{ h} = 64800 \text{ s}$, $N_{atteso} = \lambda_{med} \cdot T = 8242 \text{ passeggeri}$, il risultato della simulazione è 8169 pax processati (orizz. finito)

Stessa metodologia del caso base

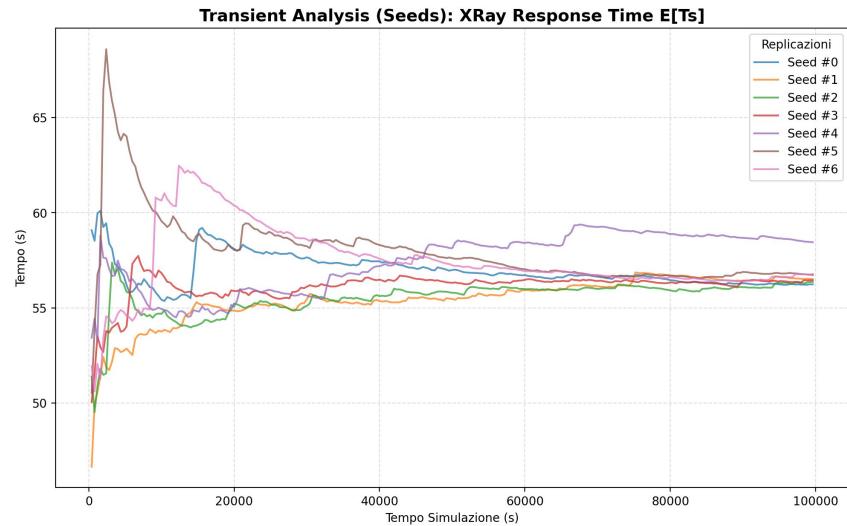
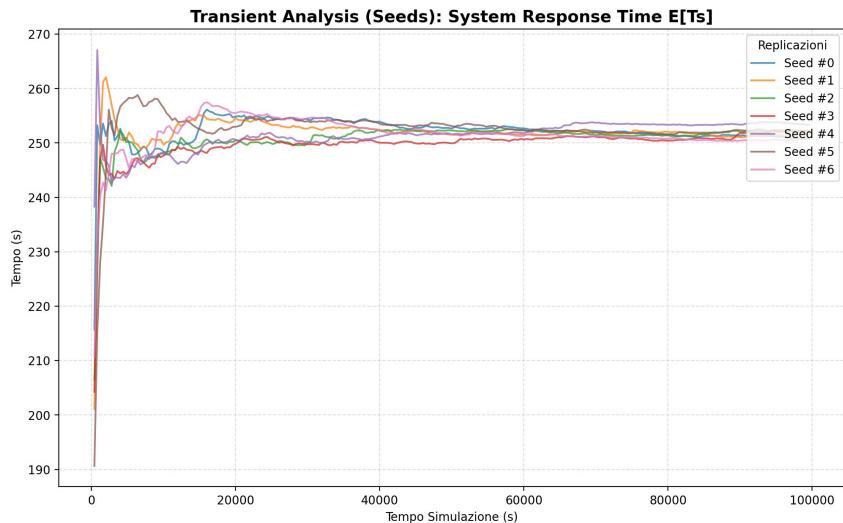
Design degli esperimenti (migliorativo)

1. Risultati dell'analisi del transitorio
2. Risultati della simulazione ad orizzonte finito
3. Risultati della simulazione ad orizzonte infinito

Risultati dell'analisi del transitorio

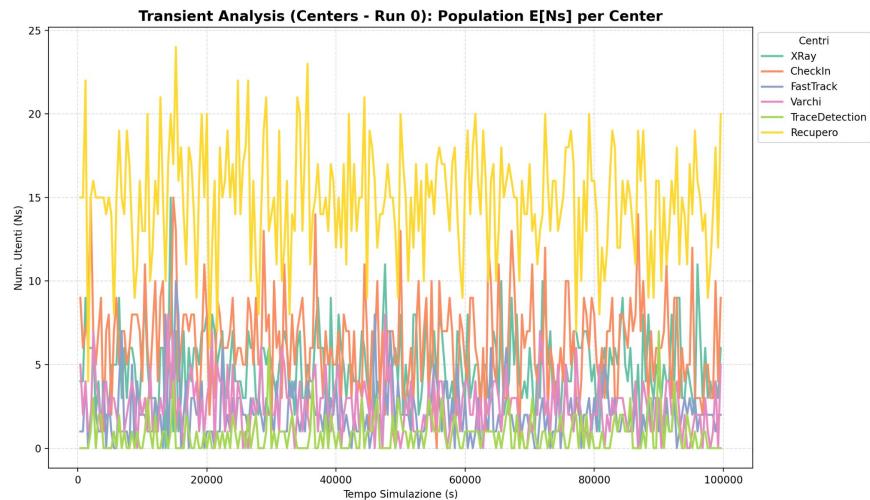
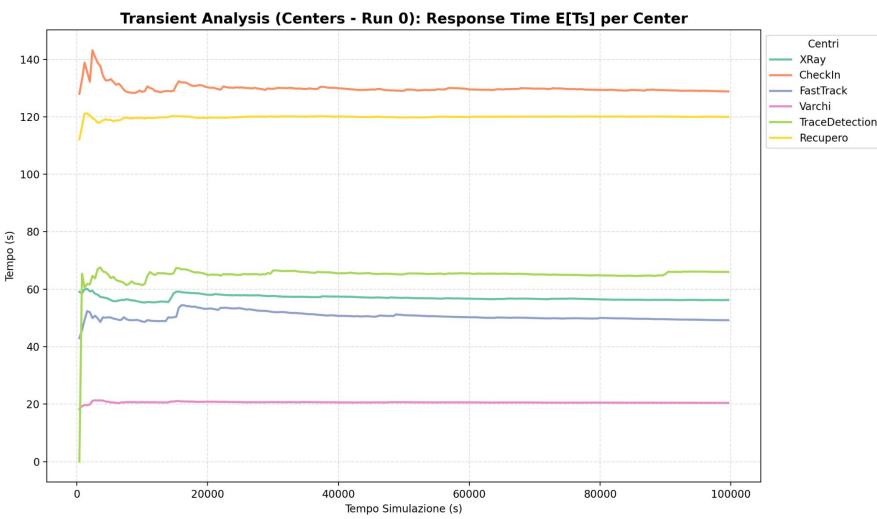
Design degli esperimenti (migliorativo)

- Durata: 100000s
- **Stabilizzazione:** 60000s



Risultati dell'analisi del transitorio (2)

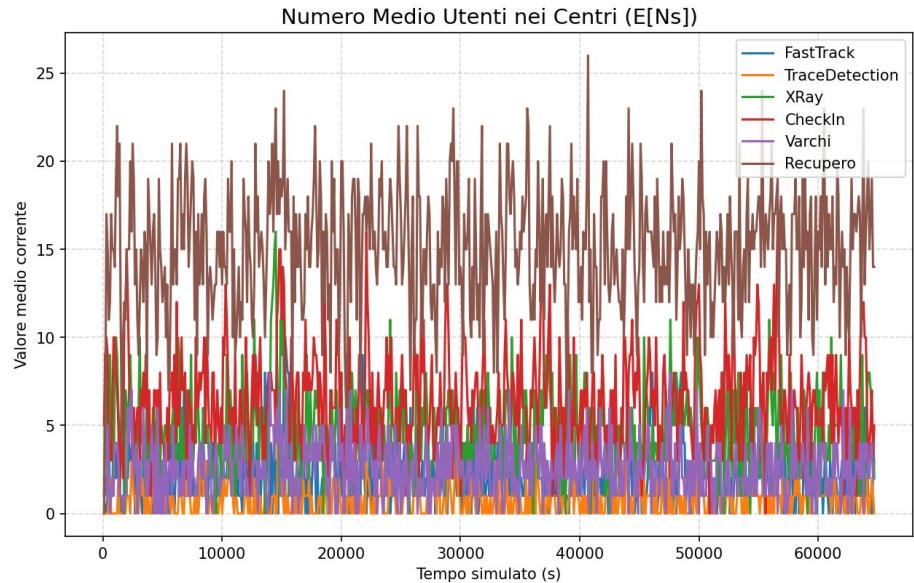
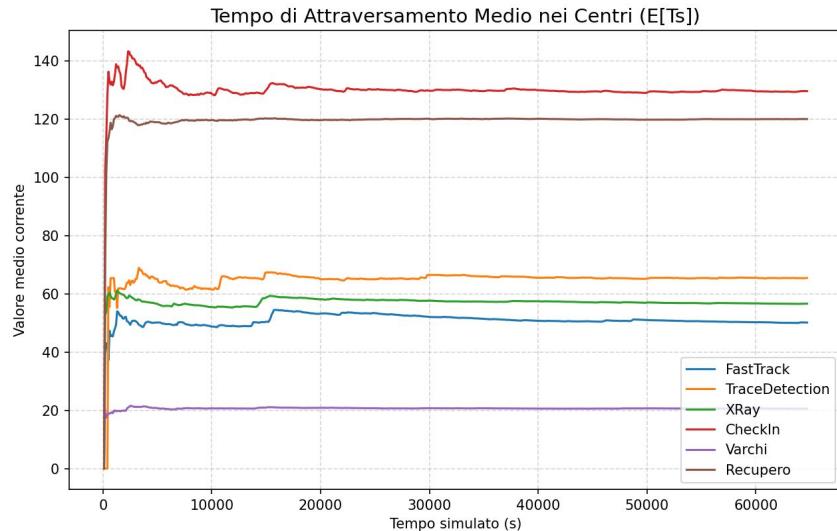
Design degli esperimenti (migliorativo)



Risultati della simulazione ad orizzonte finito

Design degli esperimenti (migliorativo)

- Durata: 18h = 64800s (work day)
- 64 replicazioni indipendenti
- Intervalli di confidenza al 95%



Risultati della simulazione ad orizzonte finito (2)

Design degli esperimenti (migliorativo)

Numero Utenti nel Sistema (Ns) - Confidenza 95%

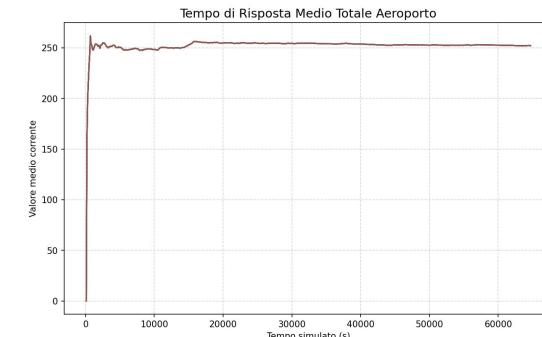
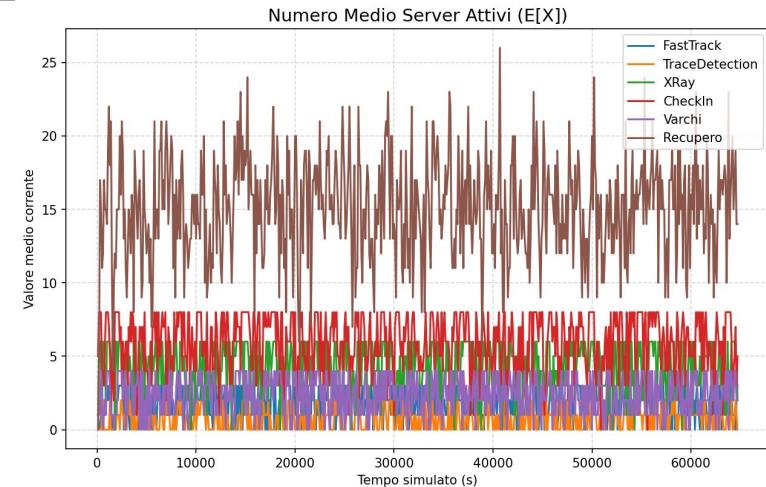
Center	Mean	Width	Min	Max
CheckIn	6.385066	0.04886	6.336206	6.433926
FastTrack	2.048213	0.012484	2.035729	2.060697
Recupero	15.169146	0.042496	15.126651	15.211642
TraceDetection	0.830729	0.008257	0.822472	0.838986
Varchi	2.614209	0.010596	2.603614	2.624805
XRay	4.791958	0.029419	4.762539	4.821377

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

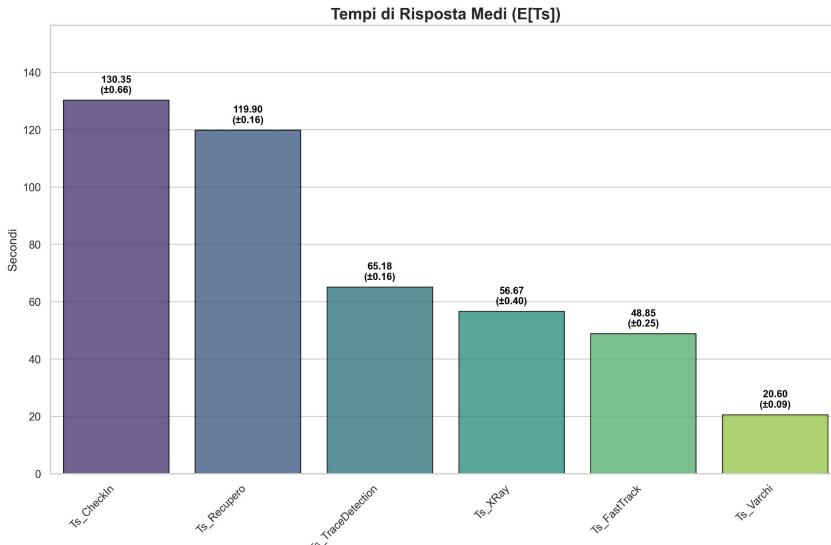
Tempo Totale Attraversamento (SystemResponseTime) - Confidenza 95%

Center	Mean	Width	Min	Max
Success	251.218328	0.396696	250.821632	251.615023

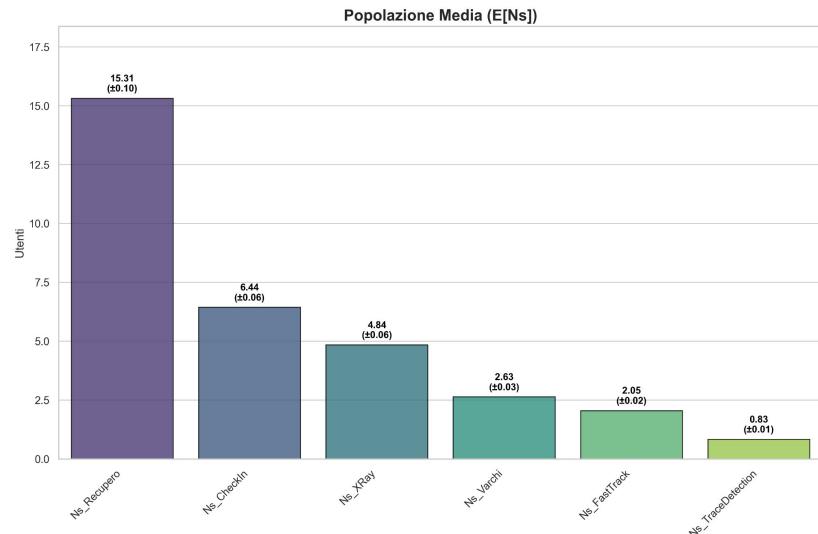


Risultati della simulazione ad orizzonte infinito

Design degli esperimenti (migliorativo)

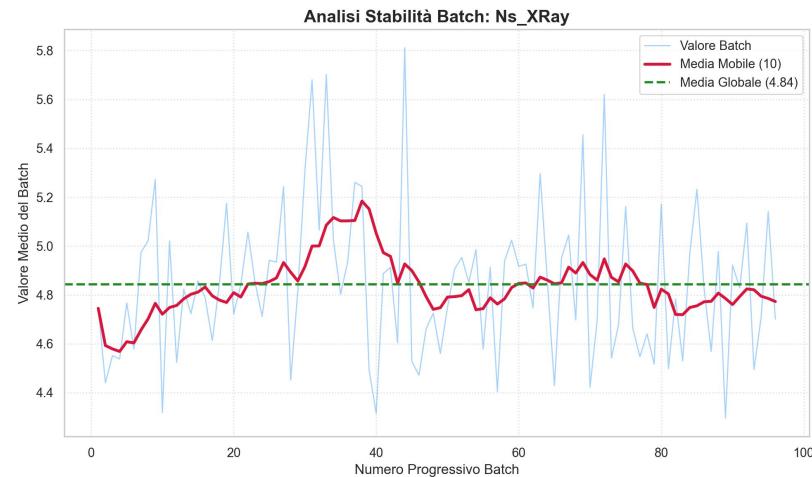
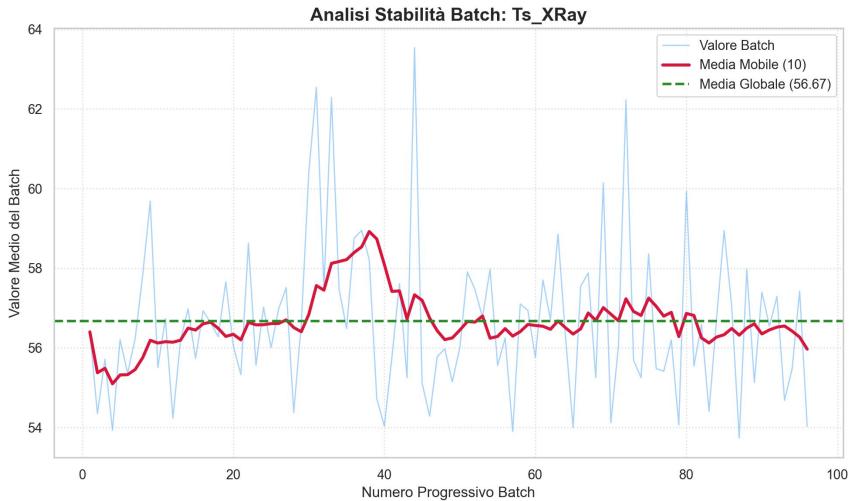


- Batch means ($B=1080$, $K=96$)
- Autocorrelazione $< |0.2|$
- Warm-up 60000s
- Intervalli di confidenza al 95%



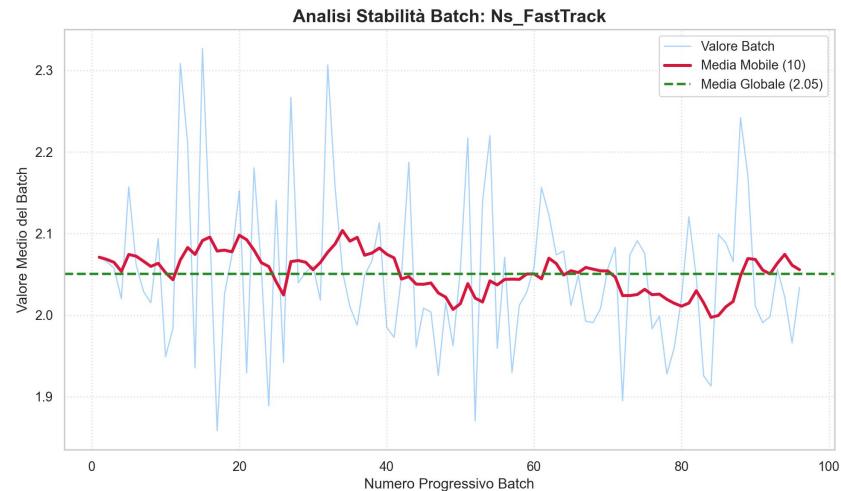
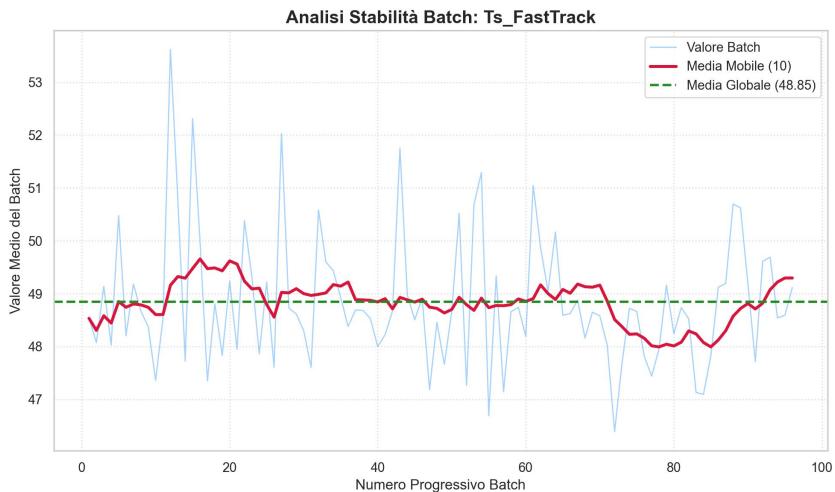
Risultati della simulazione ad orizzonte infinito (2)

Design degli esperimenti (migliorativo)



Risultati della simulazione ad orizzonte infinito (3)

Design degli esperimenti (migliorativo)



Conclusioni

1. Il **modello base** violava il primo e il secondo QoS (quelli in merito all'attesa e alla popolazione nel centro a raggi X), e in generale c'era un *inefficienza* sostanziale nel tempo totale di attraversamento del sistema, causato dal bottleneck (che era il centro a raggi X):
 - a. Il **tempo di attesa ai controlli a raggi X** era di **45,3 minuti**, violando il QoS
 - b. Il **numero medio di persone nella zona controllo** era di **327 passeggeri**, violando il QoS
 - c. Il **tempo medio di attraversamento dell'intero sistema** è **49,5 minuti**, inefficiente
2. L'introduzione del **fast track**, la **ripartizione del traffico**, e l'introduzione di **due server** nel **trace detection** ha permesso di eliminare il bottleneck e quindi il *rispetto dei QoS*:
 - a. Il **tempo di attesa ai controlli a raggi X** è **5 secondi**, rispetta il QoS
 - b. Il **numero medio di persone nella zona controllo** è **7,67 passeggeri**, rispetta il QoS
 - c. Il **tempo medio di attraversamento dell'intero sistema** è **4,2 minuti**, molto efficiente

Grazie per l'attenzione!