# Analisi delle prestazioni della Rete Wi-Fi del Campus X di Roma Tor Vergata

## Introduzione

Il presente documento si riferisce allo studio effettuato per lo sviluppo del progetto del corso Performance Modelling of Computer Systems and Networks (9 CFU), tenuto dalla prof.ssa Vittoria De Nitto Personè al 1° Anno del Corso di Laurea Magistrale in Ingegneria Informatica. Il progetto è stato sviluppato da un team composto da 3 persone: Gabriele La Delfa, Gabriele Tummolo e Domenico Verde. L’idea alla base della scelta di questo progetto era quella di generare un qualcosa di utile per il mondo reale, proponendo ed analizzando anche diverse soluzioni migliorative per il problema analizzato.

## Traccia del Progetto

****

Blocco 1

Edificio 4D

Access Points (APs)

Il Campus X di Roma Tor Vergata comprende 5 Blocchi di Edifici ciascuno dei quali comprende a sua volta 3 o 4 palazzi. Ogni palazzo è costituito da 3 o 4 piani (a cui vi si aggiunge il piano terra). L’accesso ad internet per i residenti nella struttura è garantito attraverso una rete wireless che comprende 4 Access Point (AP) per piano, posizionati agli angoli dei piani di ogni edificio. Ciascun Access Point è collegato mediante un cavo Ethernet (RJ45) ad uno Switch a 48 porte posto al piano terra di ogni edificio. Lo switch provvede a fondere i flussi provenienti dai diversi piani del palazzo e ad inoltrare correttamente il traffico in entrata ed in uscita nella giusta direzione.

## Obiettivi dello Studio

L’obiettivo dello studio è:

* Studiare come varia il tempo medio di risposta E(Ts) di un utente al variare dell’intensità del traffico

## Un Modello Concettuale

Una volta definito l’obiettivo dello studio, è necessario procedere costruendo dapprima un modello concettuale.

Da una attenta analisi della traccia, il modello concettuale a reti di code proposto è il seguente:



Chiaramente, ciascun edificio è composto da 5 piani sostanzialmente identici fra loro, pertanto è possibile semplificare il problema e studiare un solo piano dell’edificio ed estendere, per simmetria, i risultati anche agli altri piani. Come evidenziato nella traccia, ciascun utente invia pacchetti all’ AP più vicino, il quale provvede ad inoltrare il pacchetto verso lo switch, che è unico in ogni edificio. Ciascun AP e lo switch, quindi, possono avere 2 stati: Libero (Idle – se non sta trasmettendo alcun pacchetto) ed Occupato (Busy – se un pacchetto è in fase di trasmissione). Le variabili di stato del seguente modello sono:

* Numero di Jobs in ogni nodo nel sistema

Infine, supponiamo per semplicità che ciascun server abbia infiniti posti in coda e che la disciplina in ogni coda sia di tipo FIFO (First-In-First-Out).

## Un Modello di Specifica

In questa fase, invece, provvederemo a convertire il modello concettuale creato in precedenza in modello di specifica, aggiungendo informazioni di natura statistica. Innanzitutto, cercheremo di capire che distribuzioni e quali valori assumere per quanto riguarda i tempi di servizio e di inter-arrivo. Abbiamo assunto che la distribuzione del traffico è esponenziale. Pensiamo che tale ipotesi rappresenti un giusto compromesso tra semplicità ed accuratezza del modello. Studieremo, inoltre, il comportamento del sistema al variare del traffico totale in ingresso al sistema .

Inoltre, abbiamo ritenuto ragionevole assumere che, data la particolare simmetria del problema, in ciascun piano ci sia la stessa quantità di traffico:

Per quanto riguarda i tempi di servizio, in base a quanto riportato nel link 3 osserviamo come sia ragionevole assumere come dimensione media di un pacchetto (per ragioni di test):

= 512 bytes (Medium Packet Size on Internet)

Inoltre, tempo come visto diverse volte durante il corso e confermato da diversi studi, come quello riportato nel link 4, utilizzare la distribuzione Bounded Pareto può essere un ottimo modo per modellare la distribuzione del tempo di servizio dei serventi. In effetti, è noto anche da altri corsi universitari che un pacchetto Ethernet non ha dimensione indefinita, bensì ha una dimensione minima bytes ed una dimensione massima bytes. La distribuzione Bounded Pareto è una distribuzione heavy tailed e presenta 3 parametri: un parametro di shape (utile a modellare la variabilità della distribuzione) a e due parametri che indicano rispettivamente i valori minimo e massimo assumibili. Pertanto, utilizzeremo come distribuzione dei tempi di servizio una distribuzione bounded Pareto e studieremo il comportamento del sistema sia nel caso di alta variabilità del tempo di servizio (), sia nel caso di bassa variabilità ().

Per quanto riguarda i valori delle capacità dei componenti della rete, sono stati ricavati direttamente dai datasheet ufficiali (riportati nella cartella Datasheets):

* APs UniFi UAP-AC-PRO [Max data-rate 1300 Mbps @ 2.4 GHz e 5 GHz using 3x3 MIMO]:
* Switch Cisco MS250-48-HW [Switching Capacity 176 Gbps]

Inoltre, al fine di ottenere una corretta verifica e validazione del modello creato, si è deciso di creare anche un’ulteriore versione di esso avente sia tempi di servizio che i tempi di inter-arrivo con distribuzione esponenziale. In tal caso l’intero sistema diventa così una rete di Jackson e pertanto è analizzabile anche teoricamente come una rete di code separabile. In quest’ultimo caso i valori calcolati risultano:

* APs UniFi UAP-AC-PRO [Max data-rate 1300 Mbps @ 2.4 GHz e 5 GHz using 3x3 MIMO]:
* Switch Cisco MS250-48-HW [Switching Capacity 176 Gbps]

Infine, sono stati aggregati i flussi degli altri piani ottenendo la seguente figura:

In effetti, se ogni edificio possiede 5 piani, in ciascun piano (per simmetria) verrà ripartito 1/5 del traffico totale in ingresso al sistema . L’analisi verrà svolta per un solo piano; tuttavia, è necessario tenere conto anche degli ulteriori 4/5 del traffico (ovvero il traffico dei restanti piani dell’edificio) poiché contribuiscono ad aumentare l’utilizzazione dello switch che è unico per ogni palazzo.

## Il Modello Computazionale

### 1. Introduzione

Il modello computazionale è stato creato utilizzando come linguaggio di programmazione il linguaggio C, per avere maggiore efficienza e per continuità con gli esempi visti durante il corso. Inoltre, per conformità ai requisiti del progetto si è deciso di implementare un simulatore basato su next-event simulation. Il primo step nella progettazione del modello computazionale è stato determinare la Matrice di Routing, di fondamentale importanza per implementare la topologia della rete e di seguito riportata:

Ciascun elemento rappresenta la probabilità di un job di andare dal servente i al servente j:

* Se , s’intende un flusso che proviene dagli AP. Notiamo dalle rispettive righe che tutto il traffico uscente viene diretto verso lo switch.
* Se , s’intende il flusso che proviene dallo switch. Notiamo dall’ultima riga (riga 5) che tutto il traffico proveniente viene direzionato verso l’esterno del sistema.

### 2.Strutture Dati e Funzioni

Nel progetto del simulatore, sono state utilizzate le seguenti strutture dati:

* *event\_list[SERVERS + 1] (event)*: tale struttura, di fondamentale importanza nella simulazione, consiste sostanzialmente in una lista in cui vengono memorizzati i prossimi eventi. Nel caso in considerazione, gli eventi possono essere di due tipi: arrivi (dall’esterno) e partenze (dagli AP e dallo Switch). Il primo elemento della lista contiene il tempo in cui avviene il prossimo arrivo, mentre i restanti 5 elementi contengono il tempo in cui avviene la prossima partenza da ciascun server nell’ordine AP1, AP2, AP3, AP4, Switch. Inoltre, ciascun elemento possiede uno stato che permette di abilitare/disabilitare il processo degli arrivi e delle partenze. Tale stato è risultato utile nella simulazione poiché permette di specificare un tempo di simulazione “close-the-door”, oltre il quale il processo degli arrivi viene disabilitato.
* *t (clock)*: anche questa struttura riveste fondamentale importanza nella simulazione, poiché consente di tenere traccia di ogni istante di tempo nel simulatore ed è utile anche nel calcolo delle statistiche.
* *Sum (statistics)*: essa contiene la somma aggregata di tempi di servizio, numero di arrivi e numero di partenze avvenuti in ogni istante di tempo nel simulatore. È utile per calcolare le statistiche locali di ogni server.

Inoltre, sono state create le seguenti funzioni:

GetArrival(), la quale genera il tempo in cui avviene il prossimo arrivo, seguendo una distribuzione esponenziale;

GetServiceSwitch() e GetServiceAP(), le quali generano rispettivamente i tempi di servizio per lo Switch e per gli AP, seguendo in entrambi i casi una distribuzione BoundedPareto. Tali funzioni sono entrambe necessarie poiché le capacità ed i tassi di servizio dei dispositivi sono differenti.

ProcessArrival(), la funzione che gestisce gli arrivi, opera nel modo seguente: se all’arrivo di un job il server è libero, allora tale job viene processato, ovvero, viene generato il tempo di servizio e successivamente aggiornate le statistiche e la next-event list; altrimenti, il job viene posizionato in coda.

ProcessDeparture(), la funzione che gestisce, invece, le partenze. Se si verifica una partenza da uno degli AP, allora sappiamo (in base alla topologia della rete ed anche alla transition matrix) che il job sarà inoltrato verso lo switch; altrimenti, se la partenza si verifica nello switch allora è chiaro che tale job esce dal sistema. Se dopo la partenza vi sono altri job in coda, la funzione provvede a processare il prossimo job, aggiornando le statistiche.

empty\_queues(), tale funzione è utile soprattutto nel determinare se nel sistema esistono ancora jobs da processare: infatti ritorna True se non vi sono jobs e False altrimenti.

NextEvent(), tale funzione, presente anche in altri modelli visti durante il corso, è utile per determinare l’indice del prossimo evento attivo.

### 3.Init, Main Loop e Condizione d’Arresto

Per quanto riguarda l’inizializzazione del programma, sono stati azzerati sia il clock e le statistiche. Dopodiché, è stato generato il primo arrivo nel sistema ed è stato disattivato il processo delle partenze (in effetti non vi possono essere partenze se il sistema è vuoto).

Il main loop del programma è strutturato come riportato nel libro di testo: ad ogni istante di clock, viene determinato il prossimo evento tramite la funzione NextEvent(). Se l’evento è una partenza allora essa viene gestita dalla funzione ProcessDeparture(), indicando opportunamente il servente in cui si verifica tale partenza. Se l’evento è un arrivo allora prima occorre capire in quale servente si verifica: qui ci vengono in aiuto la funzione Random() implementata nella libreria rngs.c e dichiarata nel header rngs.h (entrambi inclusi nel progetto). In che modo? Tramite la funzione è possibile generare un numero casuale (compreso tra 0 e 1):

- se il numero generato è compreso tra 1/20 e 2/20 allora diremo che l’arrivo si verifica all’AP1;

- se il numero generato è compreso tra 2/20 e 3/20 allora diremo che l’arrivo si verifica all’AP2;

- se il numero generato è compreso tra 3/20 e 4/20 allora diremo che l’arrivo si verifica all’AP3;

- se il numero generato è compreso tra 3/20 e 4/20 allora diremo che l’arrivo si verifica all’AP4;

- se il numero generato è compreso tra 4/20 (ovvero 1/5) e 1 allora l’arrivo si verifica allo Switch;

Da notare che in questo modo la probabilità che vi sia un arrivo in ogni AP è 1/20 e che la probabilità che vi sia un arrivo allo switch è 4/5, come riportato nella transition matrix. Infine, viene generato il prossimo arrivo e disattivato il processo degli arrivi se il tempo di simulazione supera il “close the door” time. Al termine del main loop, vengono calcolate e poi stampate le statistiche globali (riguardanti l’intero sistema) e locali (riguardanti ciascun servente).

La struttura di base del modello computazionale è quella descritta in questo paragrafo. Tuttavia, come spiegato in precedenza, sono stati creati due simulatori: nsssn\_exp.c e nsssn\_bp.c. Entrambi differiscono solamente per la distribuzione dei tempi di servizio utilizzata nelle funzioni GetServiceAP() e GetServiceSwitch(). La distribuzione BoundedPareto, invece, è stata implementata nei file rvgs.c e dichiarata nel header rvgs.h.

### 4. Statistiche

Come riportato in precedenza, ciascun simulatore stampa due diverse tipologie di statistiche: quelle riguardanti il sistema nel suo complesso (dette globali) e quelle riguardanti ciascun servente (dette locali). Tra le globali abbiamo tempo medio di inter-arrivo, tempo medio di attesa in coda e tempo medio di risposta, e poi ancora numero medio di jobs in coda e numero medio di jobs nel sistema. Per quanto riguardo le locali, per ogni servente vengono stampati a schermo in ordine l’utilizzazione, il tempo medio di servizio, lo share (ovvero la percentuale di arrivi in tale servente rispetto agli arrivi totali), tempo medio di attesa in coda e tempo medio di risposta. Tuttavia, la metrica che prenderemo in considerazione per le successive analisi è quella che abbiamo definito “tempo medio di risposta per l’utente”, ovvero una somma tra la media dei tempi di risposta negli AP e il tempo di risposta nello Switch, riportata nel codice nella variabile avg\_wait. La scelta è ricaduta su tale metrica poiché rispecchia quello che è il vero comportamento di un utente nel sistema reale: un utente che si trova in un piano dell’edificio invia un pacchetto ad uno degli AP, il quale provvede a trasmetterlo allo switch.

## V&V – Verifica e Validazione

Essendo la fase di verifica un’attività prevalentemente di software engineering, essa è stata svolta sin dall’inizio dello sviluppo del modello computazionale utilizzando un IDE appropriato: è stata svolta un’attenta attività di debugging al fine di individuare potenziali errori nelle funzioni ed inoltre abbiamo preferito progettare ed implementare casi di test per le funzioni da noi progettate: TestProcessArrival(), TestProcessDeparture() e TestEmptyQueue().

Per quanto riguarda l’attività validazione, invece, si è considerato il modello nsssn\_exp.c. Essendo tale modello una rete di Jackson, è risultato possibile confrontare le statistiche generate dal simulatore con i valori teorici, calcolati tramite l’ausilio di apposite funzioni del programma stesso (E\_TQ(), E\_TS()). Dopo aver verificato, tramite numerose simulazioni, che i valori delle statistiche risultino molto simili ai valori teorici, il modello è stato considerato validato.

È possibile attivare/disattivare i tests effettuati nella fase di verifica e validazione cambiando il valore della variabile RUN\_TESTS\_AND\_CHECKS nel file nsssn\_exp.c.

## Analisi Transiente

Per effettuare tale analisi è stato utilizzato il meccanismo delle repliche, in particolare sono stati eseguiti 100 e 1000 run.

Il flusso di arrivo preso in esame è uguale a 10, mentre per quanto riguarda la variabilità del sistema sono stati considerati due valori di (0,5 e 1,5) per analizzare il comportamento del sistema sia con un’alta variabilità del tempo di servizio, che con una bassa variabilità del tempo di servizio.

È stato simulato il comportamento del sistema rispettivamente per:

* 1024 Jobs
* 2048 Jobs
* 4096 Jobs
* 8192 Jobs
* 16384 Jobs
* 32768 Jobs
* 65536 Jobs
* 131072 Jobs

Il programma per effettuare tale analisi prende nome di transiente.c, dove sono state riprese le funzioni ( GetArrival(), GetServiceSwitch(), GetServiceAP(), ProcessArrival(), ProcessDeparture(), NextEvent() ) e le strutture ( *event\_list[SERVERS + 1]event, t (clock), sum* ) utilizzate per il simulatore. È stata aggiunta un ulteriore funzione Initialize() per inizializzare le statistiche per ogni replica. Nel main del programma vengono definite tre variabili:

* t\_arresto, che consente di definire quale sarà il tempo di esecuzione per cui ciascuna replica.
* seed, che permette di definire il seme iniziale.
* response, dove verrà memorizzato il risultato della funzione transient() che verrà analizzata a breve.

Successivamente viene utilizzata FILE \*file = fopen(“file.txt”, “w+”) per creare un nuovo file (in questo caso denominato “file.txt”) in modalità lettura/scrittura e in questo file verranno memorizzati il tempo medio di risposta per gli utenti per ciascuna replica (uno per riga).

La funzione che permette di ricavare tali valori degli utenti prende nome di transient() ed è strutturata in egual modo al main del simulatore, ma presenta delle differenze:

* a tale funzione viene passato come argomento il tempo di esecuzione delle repliche (tramite la variabile t\_arresto)
* viene utilizzata la funzione Initialize() per azzerare tutte le statistiche della funzione stessa.
* nella condizione di arresto è stato omesso il controllo dei jobs rimanenti in coda (funzione empty\_queues del simulatore)
* viene ritornato il valore della variabile avg\_wait.

Inoltre la PlantSeeds è stata definita fuori dalla funzione transient() in modo tale che lo stato del generatore non venga modificato ad ogni replica, garantendo così l’indipendenza tra le repliche; in questo modo allo stato iniziale di ogni replica corrisponde lo stato finale della replica precedente.

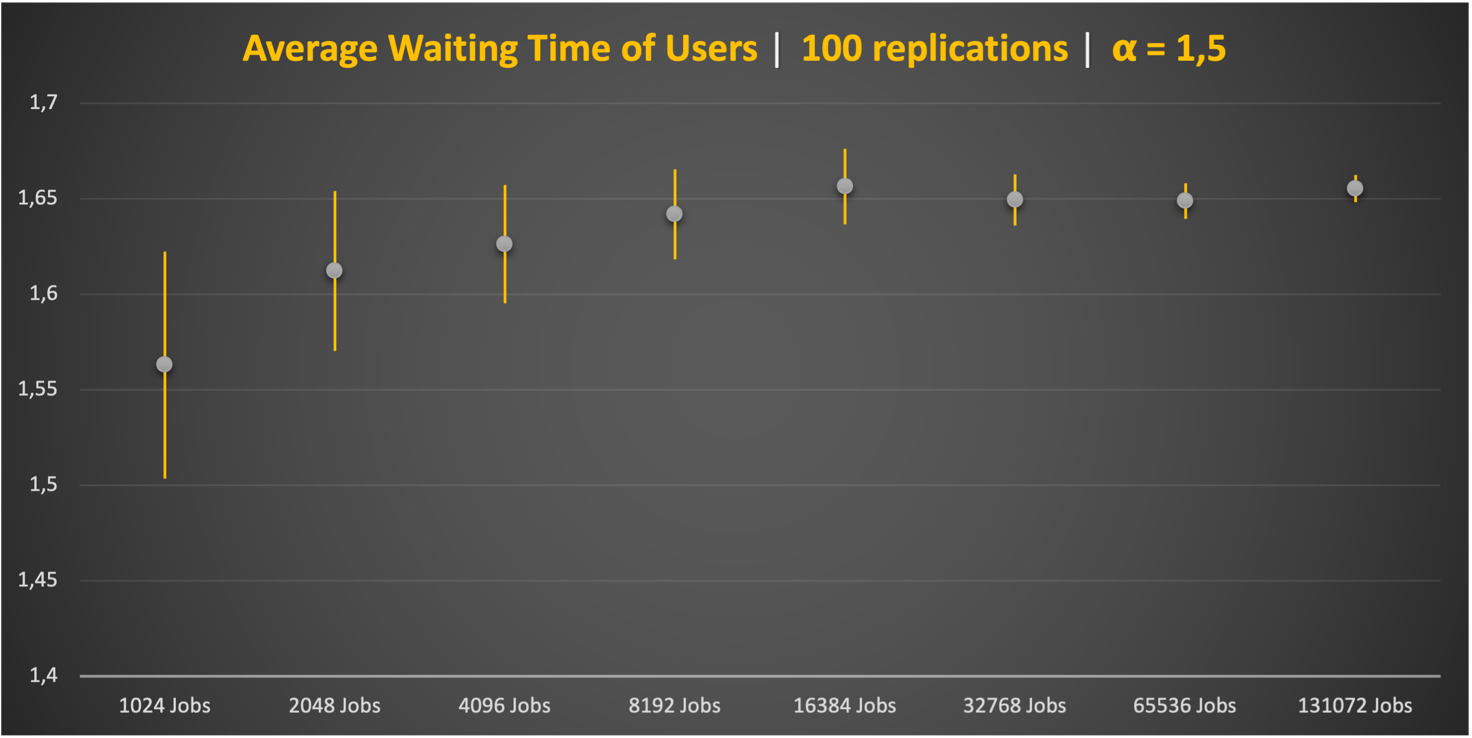
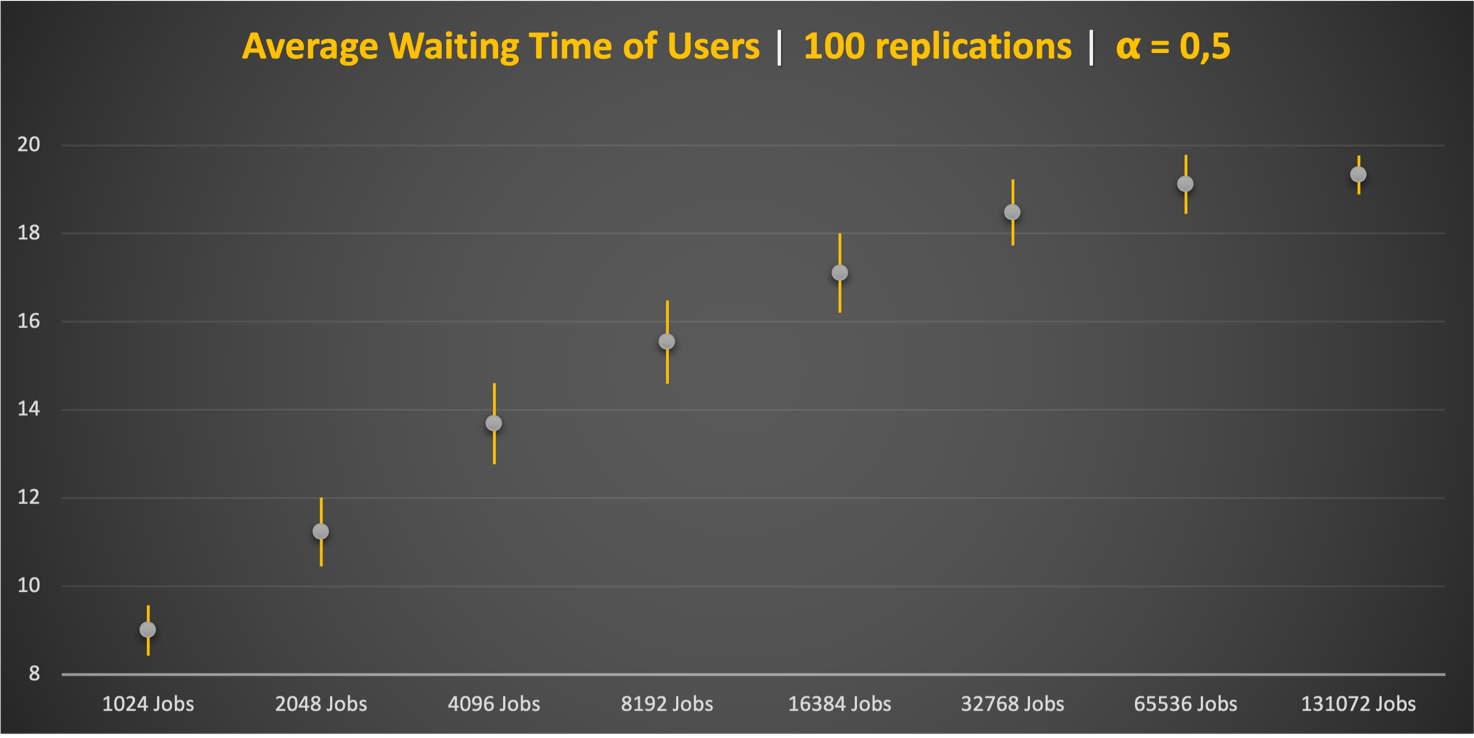
Per far vedere tale meccanismo, sono disponibili nel codice 2 diverse implementazioni della funzione transient() (di cui la seconda è attualmente commentata):

1. Prende come argomento solo la variabile t\_arresto
2. Prende come argomento sia la variabile t\_arresto che la variabile seed e tramite la funzione GetSeed consente di ottenere lo stato del flusso del generatore di numeri casuali corrente. Tale funzione GetSeed() viene implementata sia all’inizio che alla fine della funzione transient(), in modo da poter stampare a video il seed iniziale e finale della replica considerata.

Infine è stato generato il ciclo delle repliche tramite un ciclo for dove il valore ritornato dalla funzione transient() viene assegnato alla variabile response e stampata su file (file.txt).

Tramite il programma estimate.c è stato possibile calcolare l’intervallo di confidenza per ogni file testuale (riportati nella cartella Relazione/Output Statistics/Analisi Transiente), in modo da poter riportare i risultati sia in forma grafica che in forma numerica.

Nei grafici sottostanti vengono riportati nell’asse verticale il tempo medio di attesa degli utenti, mentre nell’asse orizzontale si hanno le etichette facenti riferimento ai jobs presi in considerazione per ogni replica.

Immagine che contiene testo, monitor, screenshot

Descrizione generata automaticamenteImmagine che contiene testo, screenshot, elettronico, nero

Descrizione generata automaticamente

Si può notare come nel sistema che presenta un’alta variabilità (), il tempo di attesa degli utenti varia in modo considerevole rispetto al sistema con bassa variabilità (). Il sistema assume un comportamento stazionario dopo il raggiungimento di 32768 Jobs per e di 4096 jobs per . Si può notare come l’intervallo di confidenza diminuisca passando da 100 a 1000 e ciò è dovuto alla *square root rule*: tale regola afferma che se si incrementa di 4 volte il numero di replicazioni, l’ampiezza dell’intervallo di confidenza dovrebbe ridursi approssimativamente di ½.

Si evince dai risultati che sia per 100 che per 1000 repliche, nel caso , il valore che si ottiene a 131072 jobs è circa 19,3 microsecondi; nel caso il valore che si ottiene a 131072 jobs è circa 1,65 microsecondi.

Il meccanismo delle repliche, fissato il numero di Job, può essere un ottimo modo per ottenere statistiche transienti. Tale meccanismo può essere utilizzato anche per ottenere statistiche sullo stato stazionario; tuttavia, come è possibile notare dai grafici è presente il bias iniziale che può essere risolto grazie al metodo dei batch means. Per tanto utilizzeremo quest’ultimo per compiere l’analisi dello stato stazionario.

## Analisi Stazionaria

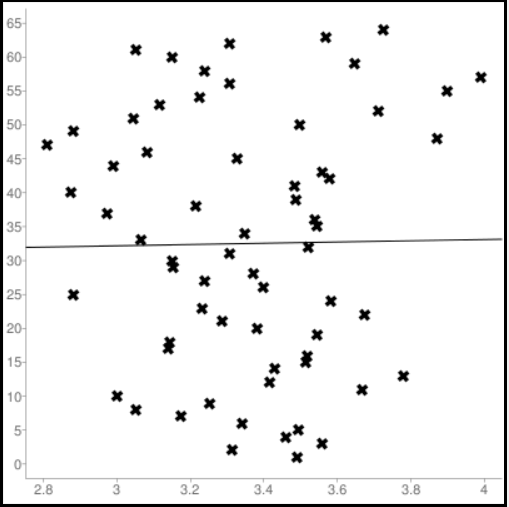
Per effettuare l’analisi stazionaria è stato utilizzato il metodo dei Batch Means. Sono stati analizzati i risultati ottenuti al variare di Lambda, in particolare sono stati analizzati gli output con = 5, 10 e 15 variando anche = 0.5 ed 1.5, che identificano rispettivamente un flusso molto variabile e un flusso a bassa variabilità. Per procedere con l’analisi è stato eseguito un unico run con un tempo di esecuzione molto elevato (assumibile come esecuzione Infinite Horizon), che permette di raccogliere tutte le informazioni necessarie. Il metodo dei Batch Means permette di calcolare un intervallo di confidenza, per le statistiche desiderate, dividendo l’intervallo di tempo dell’esecuzione in K intervalli più piccoli di dimensione B. L’aspetto più importante è scegliere K e B, per scegliere K e B sono state seguite le seguenti linee guida (in base a quanto appreso nel corso):

* B > 1
* K ≥ 32
* N=B\*K con N il numero di Job da processare durante l’esecuzione
* Banks, Carson, Nelson, and Nicol (2001, page 438) scegliere B in modo tale che l’autocorrelazione tra i Batch Means è minore di 0.2

Seguendo queste linee guida, sono stati scelti K e B: K= 64 ed B = 6250, ottenendo così N = 400000.

Utilizzando K e B si è verificato che tutti i tre punti sopra indicati vengono rispettati, in particolare il punto 3, ovvero verificare che i risultati dei singoli Batch hanno un valore di autocorrelazione inferiore a 0.2. Il file che permette di effettuare l’analisi stazionaria si chiama stazionaria.c.

Si è ottenuto il seguente grafico di dispersione:



Con un indice di autocorrelazione pari a **0.013144832139343**, inferiore a 0.2.

Per la scelta del seed iniziale è stato utilizzato un valore random in particolare 46464; tale valore non è importante perché simulando il sistema per un tempo particolarmente lungo, questo risulta indipendente dal seed iniziale. Inoltre, il bias iniziale non è rilevante in quanto questo influenza solo il primo Batch. Sono stati eseguiti 10 run, facendo variare gli stream degli AP e dello SWITCH, per ogni configurazione di Lambda e Alpha ottenendo così i grafici sottostanti.

Con un flusso di arrivo lambda = 5, in entrambi i casi analizzati appena elencati ( = 0,5 ed α = 1,5), il tempo di risposta degli utenti rimangono molto bassi, con un valore compreso tra 1,14 e 3,45 microsecondi circa; questo può essere dovuto al fatto che in corrispondenza di tale flusso di traffico, tutti i serventi (AP e Switch) presentano un valore di utilizzazione non molto elevato.

Dall’analisi transiente si è notato che il valore del tempo di risposta dell’utente in corrispondenza a 32768 jobs (punto in cui il sistema diventa stazionario) è circa uguale a 18,5 microsecondi per ed 1,65 microsecondi per . Quindi, dall’analisi stazionaria si evince che tali valori rientrano negli intervalli di confidenza rispettivamente nel 70% dei casi e nel 90% dei casi.

Si può notare dall’analisi dei bottlenecks, rappresentata successivamente, come in questo caso ovvero in corrispondenza di ed il sistema diventa instabile. Per questa ragione il tempo medio di risposta per l’utente cresce a dismisura.

## È possibile notare, in ogni grafico, come la scelta di alfa influenza il tempo di risposta per l’utente: in effetti a parità del flusso di traffico in ingresso nel sistema, con α = 0,5 in genere tale tempo è più alto rispetto al caso α = 1,5. Tali differenze si notano ampiamente negli ultimi due grafici, in quanto il variare di α influenza la stabilità del sistema.

Anche in questo caso è stato utilizzato il programma estimate.c per il calcolo degli intervalli di confidenza.

## Identificazione dei Bottlenecks

L’identificazione dei bottlenecks (o colli di bottiglia) avviene osservando se all’aumentare del traffico in ingresso al sistema λ esistono alcuni dei devices nella rete la cui utilizzazione tende ad 1 (ovvero al 100%). In tal caso, il valore del traffico per cui questo fenomeno avviene è detto “di saturazione” e si indica con , ed il device si dice che “è un collo di bottiglia” nel sistema. Ovviamente, se tutti i devices della rete raggiungono un’utilizzazione pari a 1 contemporaneamente, allora la rete è perfettamente bilanciata e non esistono colli di bottiglia nel sistema. Nel nostro caso, aumentando l’intensità del traffico in ingresso fino al valore (misurato in pacchetti al microsecondo), osserviamo che tutti gli Access Points raggiungono un’utilizzazione che è circa del 100%:

Quindi, si può concludere dicendo che gli APs costituiscono dei colli di bottiglia e pertanto, terremo conto di questo risultato nella progettazione dell’algoritmo migliorativo per il sistema. Inoltre, è bene osservare che la variabilità nella distribuzione dei tempi di servizio influenza fortemente il valore del traffico di saturazione. Tale variabilità è modellata attraverso il parametro α della distribuzione BoundedPareto: valori di indicano un’alta variabilità, mentre valori di indicano una bassa variabilità. Infatti, eseguendo diverse volte il simulatore con un valore di α = 1.5 (bassa variabilità):

Osserviamo facilmente che, in caso di bassa variabilità della distribuzione dei tempi di servizio ed a parità di traffico in ingresso, l’utilizzazione dei devices si riduce fortemente. In altre parole, meno è variabile la distribuzione dei tempi di servizio e più performante è il comportamento della rete (ovvero maggiore è il traffico che la rete riesce a sopportare). Tuttavia, tale variabilità non influenza lo studio fatto per l’identificazione dei bottlenecks, che continuano a rimanere gli APs in ogni caso. I valori mostrati nei grafici precedenti sono riportati rispettivamente nei files bottleneck-high.txt e bottlenecks-low.txt e, per completezza, vengono riportati in forma tabellare di seguito.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Lambda** | **AP1** | **AP2** | **AP3** | **AP4** | **Switch** |
| 06 | 0.539459 | 0.545725 | 0.549087 | 0.535034 | 0.078725 |
| 07 | 0.645568 | 0.640247 | 0.640872 | 0.624962 | 0.091742 |
| 08 | 0.718426 | 0.722649 | 0.740033 | 0.713626 | 0.104981 |
| 09 | 0.794464 | 0.830568 | 0.811054 | 0.800644 | 0.117331 |
| 10 | 0.896684 | 0.894283 | 0.928219 | 0.890877 | 0.129918 |
| 11 | 0.995716 | 0.982219 | 0.974416 | 0.977301 | 0.141422 |
| 12 | 0.999290 | 0.994507 | 0.974642 | 0.974117 | 0.143312 |
| 13 | 0.991379 | 0.999841 | 0.996060 | 0.989148 | 0.143483 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Lambda** | **AP1** | **AP2** | **AP3** | **AP4** | **Switch** |
| 13 | 0.581249 | 0.598913 | 0.582157 | 0.577501 | 0.084874 |
| 14 | 0.643155 | 0.630188 | 0.632899 | 0.636000 | 0.090829 |
| 15 | 0.669480 | 0.668723 | 0.685700 | 0.675952 | 0.096928 |
| 16 | 0.710267 | 0.733011 | 0.727268 | 0.713491 | 0.104383 |
| 17 | 0.769204 | 0.772666 | 0.771969 | 0.746125 | 0.110765 |
| 18 | 0.808799 | 0.814836 | 0.813391 | 0.816793 | 0.116930 |
| 19 | 0.853394 | 0.849017 | 0.853363 | 0.852466 | 0.123373 |
| 20 | 0.889390 | 0.892075 | 0.911906 | 0.888710 | 0.129954 |
| 21 | 0.944546 | 0.943441 | 0.945751 | 0.946383 | 0.136795 |
| 22 | 0.982787 | 0.976771 | 0.978850 | 0.991418 | 0.142026 |
| 23 | 0.987263 | 0.992586 | 0.997780 | 0.999659 | 0.144607 |

## Alcune Possibili Soluzioni

### Sostituzione degli AP

Considerato lo studio effettuato nel paragrafo precedente, ne è risultato che per eliminare i colli di bottiglia è necessario effettuare delle operazioni di Load Balancing, ovvero di bilanciamento del carico dovuto al traffico in ingresso. Una prima operazione, molto semplice da attuare, consiste nel sostituire gli AP con altri di capacità maggiore: in tal modo ne risulterebbe sicuramente un incremento delle prestazioni; tuttavia, l’acquisto di nuovi apparecchi ha un costo (di diverse decine di euro cadauno) e sostituire tutti gli AP di ogni piano di ogni edificio potrebbe arrivare a costare anche migliaia di euro (considerati anche i costi per la manodopera).

### Incrementare il numero degli AP

Un altro modo per effettuare load balancing potrebbe essere quello di incrementare il numero di access points presenti in ogni piano: in questo modo, il traffico viene ripartito in più AP e pertanto si osserverà una diminuzione dell’utilizzazione di ogni dispositivo e quindi un incremento delle prestazioni; tuttavia, anche in questo caso vale il discorso fatto in precedenza, ovvero: l’acquisto di nuovi apparecchi ha un costo e potrebbe risultare anche elevato. Per evitare ciò, si potrebbe pensare di fornire ogni abitazione di una presa Ethernet e comunicare a ciascun utente in fase contrattuale che l’acquisto dell’AP è a carico dell’inquilino. In questo modo, si riducono le spese derivanti dagli acquisti degli AP e si massimizza il numero di AP stessi in ogni piano (Upper Bound: 1 AP per abitazione). Tuttavia, anche questa soluzione è sconsigliata poiché l’intera rete andrebbe riprogettata per dotare ogni abitazione di una presa ethernet (quindi ulteriori costi di progetto, materiali e manodopera).

### Meccanismo QoS per il Controllo dei Pacchetti

La casa madre degli Access Points Ubiquity, tramite il controller implementato nel firmware delle proprie apparecchiature, mette a disposizione una funzionalità chiamata “Smart Queue”, attraverso la quale è possibile personalizzare la configurazione dei dispositivi. Grazie a questa tecnologia è quindi anche possibile impostare dei limiti sulla lunghezza della coda in cui vengono memorizzati i pacchetti ancora non trasmessi. In questo modo, scartando alcuni pacchetti, è possibile migliorare le performance in caso di congestione del sistema. Tale soluzione rappresenta una soluzione a costo-zero, nel senso che non è necessario acquistare nuovi dispositivi o modificare la rete esistente, ma consiste solamente in una piccola modifica della configurazione degli AP, ed è pertanto quella che consigliamo.

## Analisi dell’Algoritmo Migliorativo Proposto

Per convincere della validità della soluzione proposta, creeremo ed analizzeremo un nuovo modello di simulazione che rappresenti fedelmente la rete nel caso in cui, appunto, le code siano limitate (ovvero abbiano dimensione finita). Il sistema così facendo diventa un sistema con perdita.

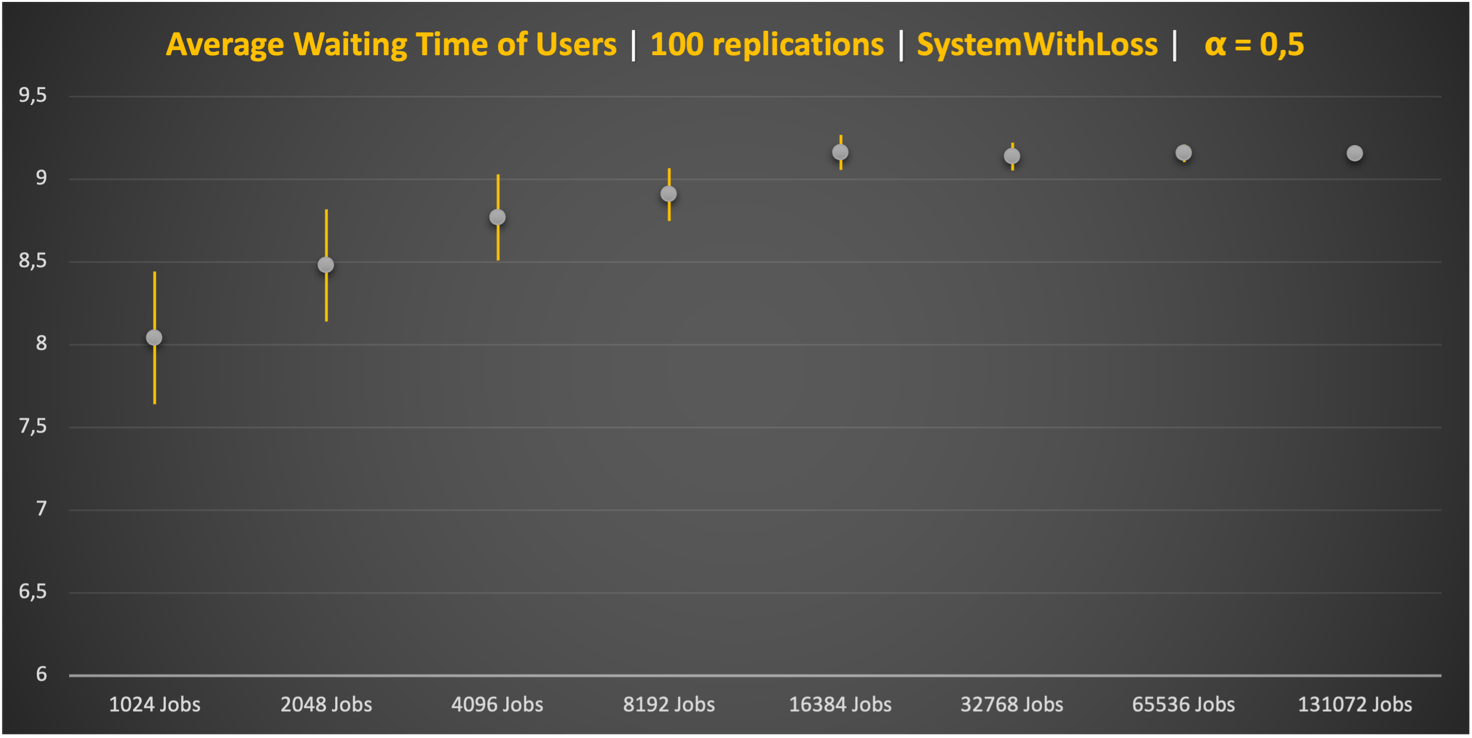
Per quanto riguarda i modelli concettuale e di specifica, in sostanza valgono le stesse considerazioni fatte in precedenza per valori, distribuzioni e variabili di stato. L’unica differenza, appunto, rimane la lunghezza delle code degli AP che non sarà più infinita, ma di dimensione 10 (il valore 10 è stato scelto poiché rappresenta il numero medio di job in coda con utilizzazione dell’80% circa in ogni AP). Per quanto riguarda il modello computazionale, rappresentato dal file nsssn\_bp\_loss.c, anch’esso rimane sostanzialmente identico al precedente, con l’aggiunta di una porzione di codice che permette di scartare i jobs se la coda è piena (righe 254-262). Tale modello, inoltre, stampa sia le statistiche che il numero di jobs scartati.

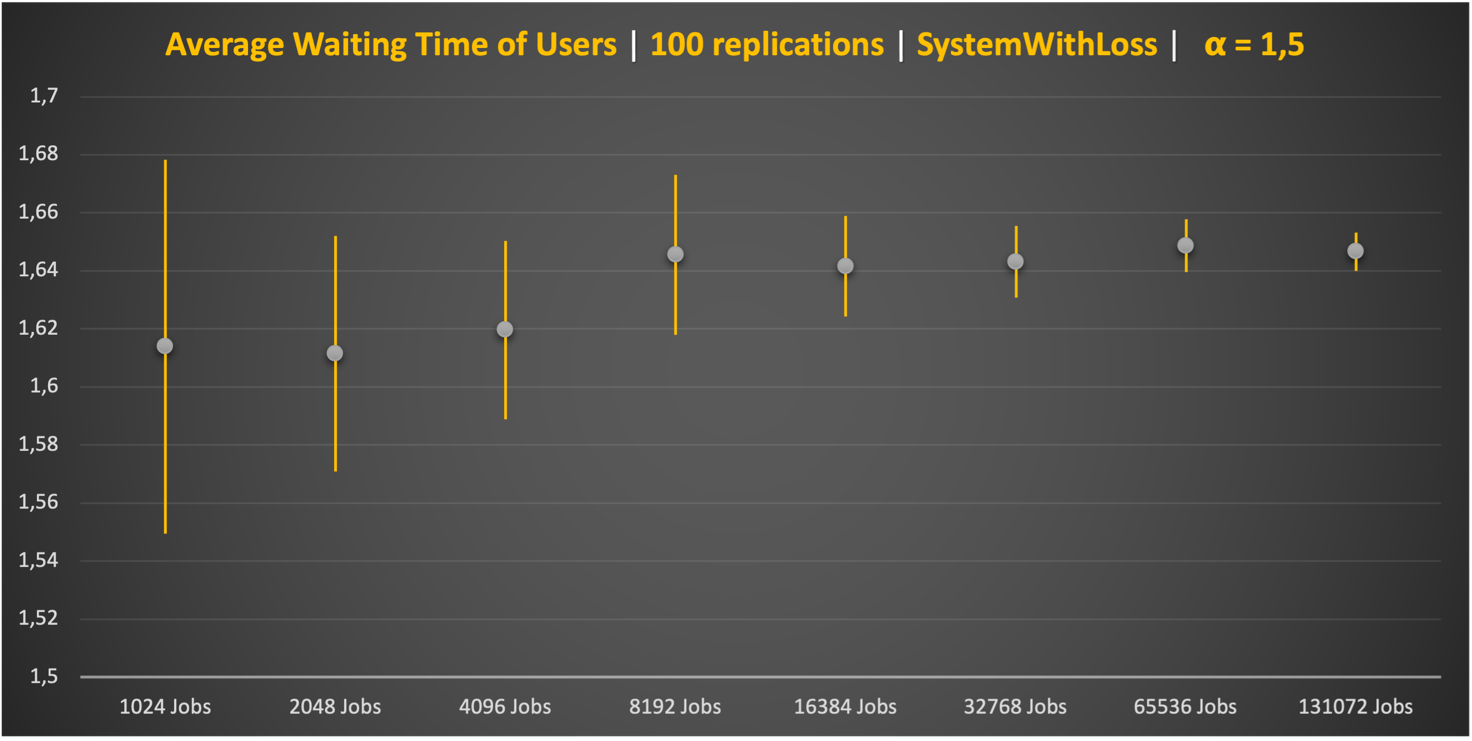
Di seguito, infine riportiamo analisi transiente e stazionaria, confrontando i risultati col modello precedente.

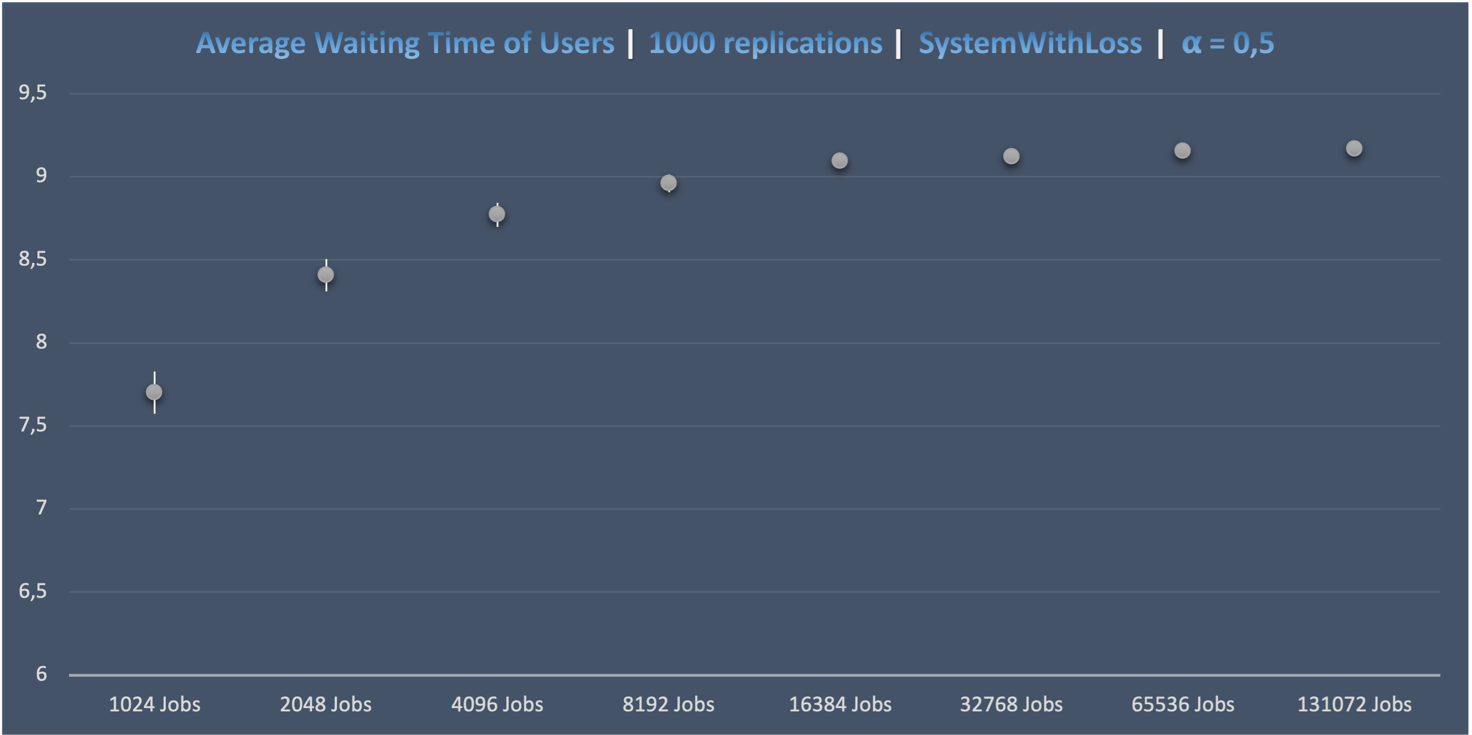
## Analisi Transiente

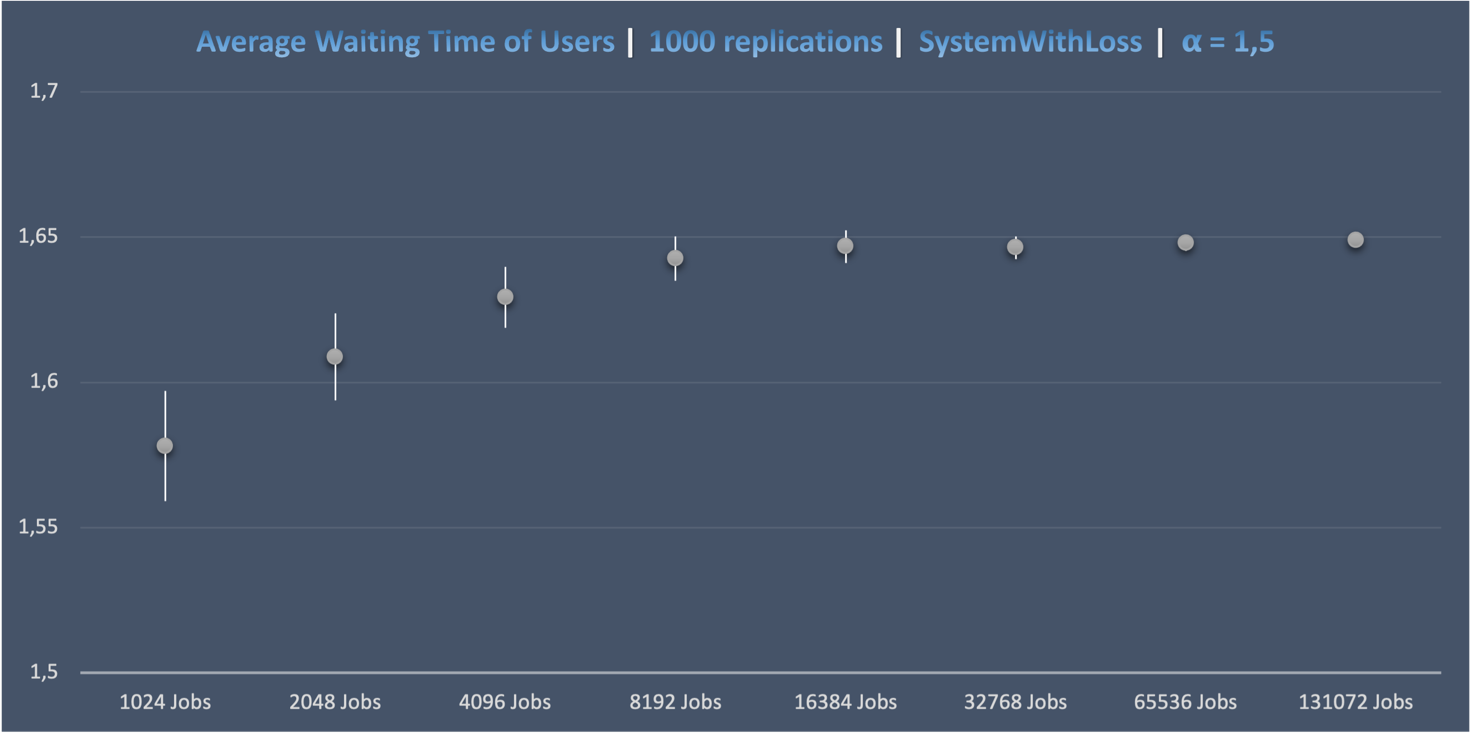
Anche per quanto riguarda l’analisi transiente, nel file transiente\_loss.c, la struttura rimane pressoché identica al file transiente.c, ma anche in questo caso si è provveduto ad aggiungere una porzione di codice che permette di scartare i jobs se la coda dovesse raggiungere il limite massimo (10). Inoltre, viene stampato a video il numero dei job scartati e la loro percentuale rispetto agli arrivi; per poter utilizzare tale informazione è stato reindirizzato l’output della file transiente\_loss.c in modo da poter memorizzare i valori dei jobs scartati e della loro percentuale in appositi file txt sotto il nome di “xRefused.txt” dove con “x” si intende il numero di job presi in considerazione (riportati nella cartella Relazione/Output statistics/Analisi Transiente/SystemWithLoss)

Come nel caso precedente vengono riportati nell’asse verticale il tempo medio di attesa degli utenti, mentre nell’asse orizzontale si hanno le etichette facenti riferimento ai jobs presi in considerazione per ogni replica.









Di seguito verranno mostrate le differenze del modello migliorativo rispetto al modello precedente tramite tabelle numeriche. Da tale analisi si può notare come la percentuale dei Job respinti si trova al di sotto del 1% in tutti i casi considerati, ma il tempo medio di attesa dell’utente è diminuito notevolmente.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **100 REPLICHE - Alfa = 0,5** | | | | | | | | | |
|  | **1024 Jobs** | **2048 Jobs** | **4096** Jobs | **8192 Jobs** | **16384 Jobs** | **32768 Jobs** | **65536 Jobs** | **131072 Jobs** |
| MAX | 9,575483 | 12,016587 | 14,602429 | 16,479599 | 18,000087 | 19,227605 | 19,782127 | 19,76646 |
| MIN | 8,428055 | 10,44602 | 12,773431 | 14,594418 | 16,201811 | 17,727826 | 18,445921 | 18,887191 |
| MEDIA | **9,001769** | **11,231303** | **13,68793** | **15,537008** | **17,100949** | **18,477716** | **19,114024** | **19,326826** |
| (+/-) | 0,573714 | 0,785283 | 0,914499 | 0,94259 | 0,899138 | 0,74989 | 0,668103 | 0,439635 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **100 REPLICHE - SystemWithLoss - Alfa = 0,5** | | | | | | | | |
|  | **1024 Jobs** | **2048 Jobs** | **4096** Jobs | **8192 Jobs** | **16384 Jobs** | **32768 Jobs** | **65536 Jobs** | **131072 Jobs** |
| MAX | 8,441322 | 8,817597 | 9,028865 | 9,067199 | 9,265916 | 9,219711 | 9,211451 | 9,199054 |
| MIN | 7,640158 | 8,140686 | 8,508165 | 8,746865 | 9,054633 | 9,05242 | 9,102226 | 9,109258 |
| MEDIA | **8,04074** | **8,479142** | **8,768515** | **8,907032** | **9,160275** | **9,136065** | **9,156839** | **9,154156** |
| (+/-) | 0,400582 | 0,338456 | 0,26035 | 0,160167 | 0,105642 | 0,083645 | 0,054613 | 0,044898 |
| Jobs Refused | 6,46 | 15,64 | 32,84 | 67,9 | 150,32 | 302,78 | 605,7 | 1213,96 |
| % Jobs Refused | ***0,6102%*** | ***0,7414%*** | ***0,7994%*** | ***0,829%*** | ***0,9172%*** | ***0,9227%*** | ***0,9227%*** | ***0,9261%*** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **100 REPLICHE - Alfa = 1,5** | | | | | | | | |
|  | **1024 Jobs** | **2048 Jobs** | **4096** Jobs | **8192 Jobs** | **16384 Jobs** | **32768 Jobs** | **65536 Jobs** | **131072 Jobs** |
| MAX | 1,622418 | 1,653937 | 1,657203 | 1,665374 | 1,676096 | 1,662584 | 1,658005 | 1,662297 |
| MIN | 1,503584 | 1,570218 | 1,595334 | 1,618196 | 1,636694 | 1,636112 | 1,639338 | 1,648319 |
| MEDIA | **1,563001** | **1,612077** | **1,626269** | **1,641785** | **1,656395** | **1,649348** | **1,648671** | **1,655308** |
| (+/-) | 0,059417 | 0,041859 | 0,030935 | 0,023589 | 0,019701 | 0,013236 | 0,009333 | 0,006989 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **100 REPLICHE - SystemWithLoss - Alfa = 1,5** | | | | | | | | |
|  | **1024 Jobs** | **2048 Jobs** | **4096** Jobs | **8192 Jobs** | **16384 Jobs** | **32768 Jobs** | **65536 Jobs** | **131072 Jobs** |
| MAX | 1,678337 | 1,652098 | 1,650299 | 1,673132 | 1,658919 | 1,655433 | 1,657764 | 1,653225 |
| MIN | 1,549452 | 1,570963 | 1,588955 | 1,617919 | 1,624303 | 1,630881 | 1,639596 | 1,640081 |
| MEDIA | **1,613895** | **1,61153** | **1,619627** | **1,645525** | **1,641611** | **1,643157** | **1,64868** | **1,646653** |
| (+/-) | 0,064443 | 0,040567 | 0,030672 | 0,027606 | 0,017308 | 0,012276 | 0,009084 | 0,006572 |
| Jobs Refused | 0,03 | 0,05 | 0,15 | 0,44 | 0,54 | 1,07 | 2,23 | 5,49 |
| % Jobs Refused | ***0,0028%*** | ***0,0024%*** | ***0,0035%*** | ***0,0050%*** | ***0,0037%*** | ***0,0028%*** | ***0,0022%*** | ***0,0037%*** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1000 REPLICHE - Alfa = 0,5 | | | | | | | | |
|  | **1024 Jobs** | **2048 Jobs** | **4096** Jobs | **8192 Jobs** | **16384 Jobs** | **32768 Jobs** | **65536 Jobs** | **131072 Jobs** |
| MAX | 8,860483 | 11,180697 | 13,466116 | 15,503431 | 17,544536 | 18,628623 | 19,126848 | 19,47069 |
| MIN | 8,490974 | 10,745335 | 12,959923 | 14,956005 | 16,982387 | 18,153079 | 18,738383 | 19,191594 |
| MEDIA | **8,675729** | **10,963016** | **13,213019** | **15,229718** | **17,263462** | **18,390851** | **18,932615** | **19,331142** |
| (+/-) | 0,184755 | 0,217681 | 0,253096 | 0,273713 | 0,281075 | 0,237772 | 0,194232 | 0,139548 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1000 REPLICHE - SystemWithLoss - Alfa (0,5) | | | | | | | | |
|  | **1024 Jobs** | **2048 Jobs** | **4096** Jobs | **8192 Jobs** | **16384 Jobs** | **32768 Jobs** | **65536 Jobs** | **131072 Jobs** |
| MAX | 7,826515 | 8,503707 | 8,843938 | 9,014281 | 9,133306 | 9,145486 | 9,174232 | 9,180832 |
| MIN | 7,574131 | 8,310931 | 8,698291 | 8,907693 | 9,057919 | 9,092759 | 9,137159 | 9,15262 |
| MEDIA | **7,700323** | **8,407319** | **8,771114** | **8,960987** | **9,095613** | **9,119122** | **9,155695** | **9,166726** |
| (+/-) | 0,126192 | 0,096388 | 0,072823 | 0,053294 | 0,037694 | 0,026363 | 0,018536 | 0,014106 |
| Jobs Refused | 6,016 | 15,372 | 33,849 | 71,69 | 148,758 | 299,276 | 603,376 | 1209,277 |
| % Jobs Refused | ***0,56963%*** | ***0,72929%*** | ***0,82461%*** | ***0,87404%*** | ***0,90694%*** | ***0,91237%*** | ***0,91966%*** | ***0,92227%*** |

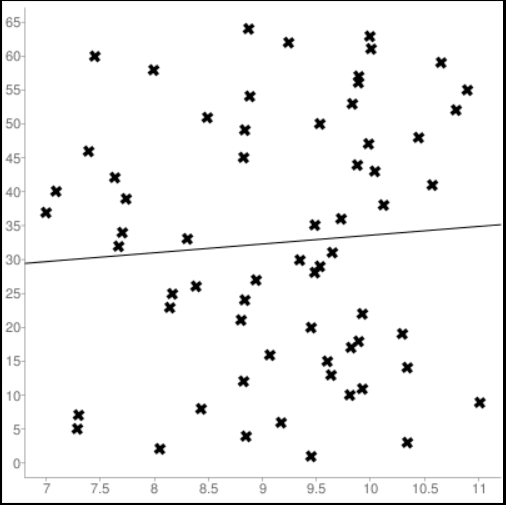
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1000 REPLICHE - Alfa = 1,5 | | | | | | | | |
|  | **1024 Jobs** | **2048 Jobs** | **4096** Jobs | **8192 Jobs** | **16384 Jobs** | **32768 Jobs** | **65536 Jobs** | **131072 Jobs** |
| MAX | 1,603594 | 1,632283 | 1,651125 | 1,652061 | 1,657028 | 1,65244 | 1,651557 | 1,654191 |
| MIN | 1,564279 | 1,602123 | 1,629776 | 1,636789 | 1,64572 | 1,644456 | 1,645816 | 1,650167 |
| MEDIA | **1,583937** | **1,617203** | **1,64045** | **1,644425** | **1,651374** | **1,648448** | **1,648687** | **1,652179** |
| (+/-) | 0,019658 | 0,01508 | 0,010674 | 0,007636 | 0,005654 | 0,003992 | 0,002871 | 0,002012 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1000 REPLICHE - SystemWithLoss - Alfa (1,5) | | | | | | | | |
|  | **1024 Jobs** | **2048 Jobs** | **4096** Jobs | **8192 Jobs** | **16384 Jobs** | **32768 Jobs** | **65536 Jobs** | **131072 Jobs** |
| MAX | 1,597046 | 1,623695 | 1,639852 | 1,650282 | 1,652459 | 1,650335 | 1,650602 | 1,650848 |
| MIN | 1,55916 | 1,593784 | 1,61878 | 1,634989 | 1,64114 | 1,642494 | 1,645293 | 1,646886 |
| MEDIA | **1,578103** | **1,60874** | **1,629316** | **1,642636** | **1,646799** | **1,646414** | **1,647947** | **1,648867** |
| (+/-) | 0,018943 | 0,014956 | 0,010536 | 0,007647 | 0,005659 | 0,00392 | 0,002654 | 0,001981 |
| Jobs Refused | 0,032 | 0,087 | 0,172 | 0,347 | 0,616 | 1,28 | 2,554 | 5,103 |
| % Jobs Refused | ***0,00304%*** | ***0,00419%*** | ***0,00403%*** | ***0,00403%*** | ***0,00403%*** | ***0,004%*** | ***0,00294%*** | ***0,00287%*** |

## Analisi Stazionaria

L’analisi stazionaria è stata effettuata, come nel caso precedente, con K = 64 e B = 6250 con N = 400000, verificando che questi valori rispettino le condizioni viste in precedenza (B > 1, K ≥ 32, N=B\*K, B scelto in modo tale che l’autocorrelazione tra i Batch Means è minore di 0.2). Per effettuare tale analisi è stato utilizzato il file stazionaria\_loss.c .

È stata quindi calcolata l’autocorrelazione dei valori dei batch, ottenendo un valore pari a **0.072194142500102** ed il seguente grafico di dispersione:



Si è proseguito calcolando gli intervalli di confidenza dell’Average Waiting Time for Users che hanno permesso di realizzare i seguenti grafici:

Come si nota da questi grafici c’è un miglioramento in termini di prestazioni rispetto all’algoritmo precedente, in particolare:

* Nel caso di = 5, ovvero quando i componenti presentano una bassa utilizzazione, il comportamento del sistema è pressoché identico al precedente
* Nel caso di = 10 ed = 0,5 (alta variabilità del traffico e quindi alta utilizzazione degli AP) il tempo medio di risposta per l’utente si dimezza rispetto al modello precedente. Se = 1,5 (bassa variabilità del traffico), invece, il comportamento del sistema rimane invariato.
* Nel caso di = 15, infine, è possibile notare come in entrambi i casi ( = 0,5 ed = 1,5) si ottiene un incremento delle prestazioni del sistema rispetto al modello precedente. In particolare, i vantaggi risultano evidenti nel caso = 0,5, dove nel modello precedente si erano ottenuti dei tempi medi di risposta degli utenti molto elevati (in quanto il sistema è instabile), mentre nell’algoritmo migliorativo il sistema presenta dei tempi ragionevoli.

Conclusioni

Il lavoro svolto è consistito nella scelta di un caso di studio (ovvero la rete wi-fi del Campus X) e quindi nello sviluppo successivo del simulatore per il caso di studio scelto. È stata poi effettuata un’analisi transiente e stazionaria del modello creato, non essendo possibile il calcolo dei valori teorici per la rete scelta. Dai risultati delle analisi è emerso che è possibile migliorare le prestazioni della rete; per questo motivo, abbiamo proposto un algoritmo migliorativo che consiste sostanzialmente in una semplice riconfigurazione degli access points (soluzione economicamente vantaggiosa poiché a costo pressoché nullo). È stato creato un simulatore anche dell’algoritmo migliorativo e sono state svolte nuovamente entrambe le analisi (transiente e steady-state) al fine di dimostrare l’effettivo miglioramento delle prestazioni. I risultati delle analisi sono stati comparati con i risultati precedenti, evidenziando in effetti i notevoli vantaggi della soluzione proposta.

Pertanto, il nostro consiglio è quello di implementare il meccanismo di controllo dei pacchetti in ogni AP.

Bibliografia e Link Utili

1. Capacity Calculus (MIMO): <https://documentation.meraki.com/MR/WiFi_Basics_and_Best_Practices/Wireless_Throughput_Calculations_and_Limitations>
2. Switching Capacity: <https://forum.huawei.com/enterprise/en/forwarding-performance-and-switching-capacity/thread/570609-861>
3. Packet Best Size: <https://www.networkworld.com/article/2300175/picking-the--best--packet-size.html>
4. Pareto Distribution: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1045.7&rep=rep1&type=pdf>
5. Smart Queue: <https://community.ui.com/questions/Smart-queue-Limit-setting/3930a03e-b7a3-4ac0-85fe-1bc8f042b9c6>
6. L. M. Leemis, S. K. Park - Discrete Event Simulation: A First Course.
7. Mor Harchol Balter - Performance Modelling and Design of Computer System.