A Tandem Queueing Model for Delay Analysis in Disconnected Ad Hoc Networks

An OMNeT++ simulation

Giulia Cantini

18 ottobre 2019

Indice

1 Introduzione

I tradizionali protocolli di routing *store-and-forward*, che richiedono l'esistenza di un cammino end-to-end che colleghi sorgente e destinazione, non possono essere utilizzati in reti soggette a disconnessioni frequenti. Una soluzione che può risolvere questo problema è sfruttare il movimento dei nodi della rete ed utilizzare il paradigma *store-carry-and-forward*.
In tali reti, chiamate *Delay Tolerant Networks* (DTN), il ritardo di trasmissione dei dati tra i nodi può risultare elevato a causa delle disconnessioni.
Un importante fattore che caratterizza le DTN è l'opportunità di contatto tra coppie di nodi. Due nodi sono in contatto se si trovano entro il range di trasmissione l'uno dell'altro, ovvero ad una distanza tale da consentire lo scambio di pacchetti.

Il modello sviluppato consiste di una rete opportunistica formata da un insieme di tre code che collaborano tra loro per realizzare il meccanismo opportunistico.
Esso si basa sulle seguenti assunzioni:

1. I contatti sono puramente opportunistici, ovvero non si ha nessuna conoscenza di quando questi avverranno (protocollo *no-context*);
2. la distribuzione dei tempi di inter-contatto ha media e varianza finite;
3. il nodo sorgente ha in arrivo un flusso di pacchetti;
4. i nodi sorgente e destinazione sono fissi, mentre i nodi intermedi sono mobili e fungono da *relay*;
5. il modello è basato su due code che sono servite in alternanza da un singolo server. Il server è autonomo e non c'è modo di controllarne il movimento;
6. il nodo mobile salva i pacchetti che provengono dalla sorgente e li inoltra a destinazione;
7. il nodo mobile non è mai nel *range* di sorgente e destinazione allo stesso tempo.

2 Modello

Il modello considerato consiste di 3 sistemi a server singolo first-in-first-out (FIFO) con coda a capacità illimitata, Q_i, i = 1,2,3 in cui i pacchetti arrivano in Q_1 e successivamente richiedono il servizio in Q_2 prima di raggiungere la destinazione in Q_3. Particolare caratteristica del modello è che Q_2 si alterna tra le posizioni L_1 e L_2 in modo che il server di Q_1 è disponibile solamente quando Q_2 si trova in L_1 e il server di Q_2 è disponibile solo quando Q_2 si trova in L_2.
Inoltre esiste un tempo di *switch-over* da L_i a L_j ($i \neq j, j \in \{1, 2\}$) in cui né il server in Q_1 né il server in Q_2 sono disponibili.
Q_3 funge da *sink* e non viene considerata nell'analisi.
Q_2 si muove in maniera autonoma, rimanendo in una determinata locazione L per un intervallo di tempo generato da una distribuzione esponenziale (*tempo di permanenza*). Durante il periodo di disponibilità di un server in Q_1 o Q_2, questo può alternare momenti di servizio e momenti in cui è *idle*, in base alla presenza di clienti/pacchetti da servire.

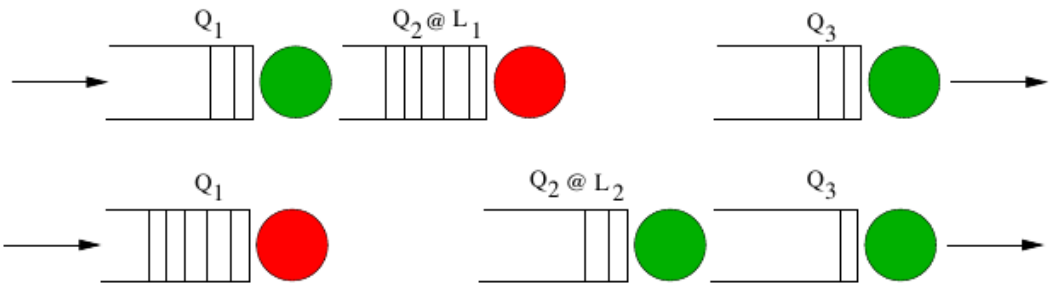


Figura 1: Modello a tre code con una coda mobile. Top: Quando Q_2 è in L_1, il suo server è down. Bottom: Quando Q_2 è in L_2, il suo server è up.

3 Implementazione

Nella rete modellata, denominata *OppNet*, Q_1 e Q_2 sono due moduli *OppPassiveQueue* serviti da un unico server S, Q_3 è una coda con server built-in, e sono presenti inoltre una *source* che genera i pacchetti e un *sink* che li raccoglie dopo che hanno attraversato l'intero sistema.

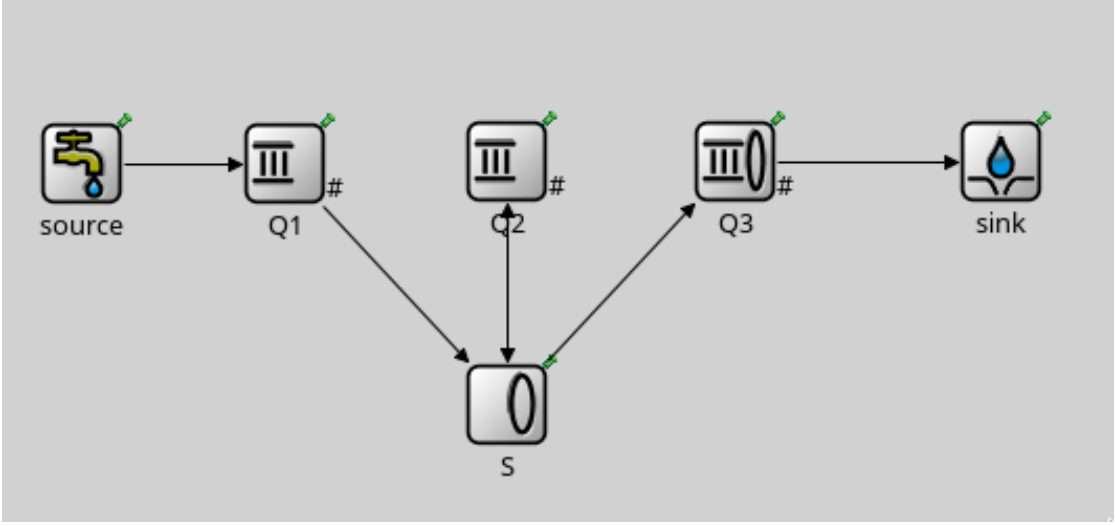


Figura 2: Visualizzazione del file oppnet.ned

Il comportamento di S è determinato dalla classe OppServer.cc che estende la classe *Server* presente nella libreria *queueinglib*.
Il meccanismo opportunistico è implementato introducendo due nuovi tipi di eventi: *startSwitchEvent* e *endSwitchOverTimeEvent*, che delimitano il periodo di *switch over*,

in cui il server non è disponibile per nessuna delle due code: il primo si verifica per segnalare l'inizio dello *switch* ed il secondo per segnalarne la fine.
La non disponibilità del server è espressa dalla variabile booleana *serverIsAvailable*.
Un'altra *flag isServingQ1* viene invece utilizzata per controllare quale coda tra Q_1 e Q_2 deve essere servita.

```
void OppServer::handleMessage(cMessage *msg) {

    EV << "Message name " << msg->getName() << endl;

    // Self-messages
    if (msg == startSwitchEvent) {
         serverIsAvailable = false;
         // If server is not doing anything
         if (!jobServiced)
             scheduleAt(simTime()+switchOverTime, endSwitchOverTimeEvent);
         // If jobServiced = true then the server will process the job and the next event
         // will be a end service event
    }
    else if (msg == endSwitchOverTimeEvent) {
        // Invert isServingQ1 (0 to 1 and 1 to 0)
        serverIsAvailable = true;
        isServingQ1 = !isServingQ1;

        // Schedule next switch event
        if (isServingQ1)
            scheduleAt(simTime()+Q1visitTime, startSwitchEvent);
       else
            scheduleAt(simTime()+Q2visitTime, startSwitchEvent);

        requestJob();
    }
    else if (msg == endServiceMsg) {

        simtime_t d = simTime() - endServiceMsg->getSendingTime();
        jobServiced->setTotalServiceTime(jobServiced->getTotalServiceTime() + d);

        EV << "Deciding where to send the job, isServingQ1 is " << isServingQ1 << "\n";
        if (isServingQ1) {
            // Send the job out to Q2
            EV << "Sending the job out to Q2" << endl;
            send(jobServiced, "out", 0);
        }
        else {
            // Send the job out to Q3
            EV << "Sending the job out to Q3" << endl;
            send(jobServiced, "out", 1);
        }
```

Figura 3: Gestione dello *switch over* e alternanza del server tra le due code *Q1* e Q_2.

4 Simulazione

4.1 Misure di prestazione

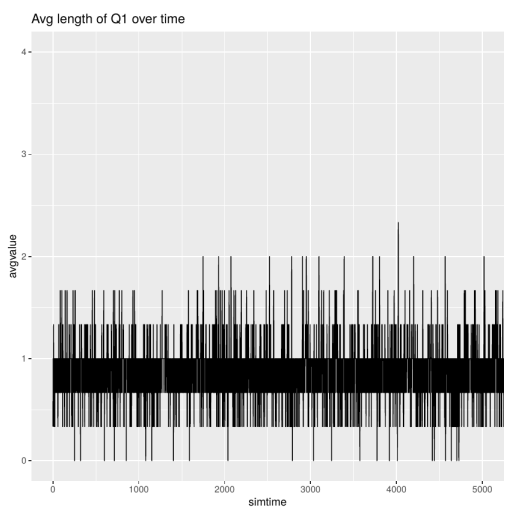
Le variabili di interesse considerate per la simulazione sono state:

- Tempo medio di risposta del sistema, ovvero il tempo medio di soggiorno dell'utente (*pacchetto*) nel sistema, dalla generazione nella *source* all'arrivo nel *sink*.
- Numero medio di utenti in coda nel sistema, di cui si è analizzato singolarmente la media delle lunghezze delle tre code Q_1, Q_2, Q_3.
- Throughput medio del sistema, ovvero la media di utenti serviti sull'intervallo di tempo.

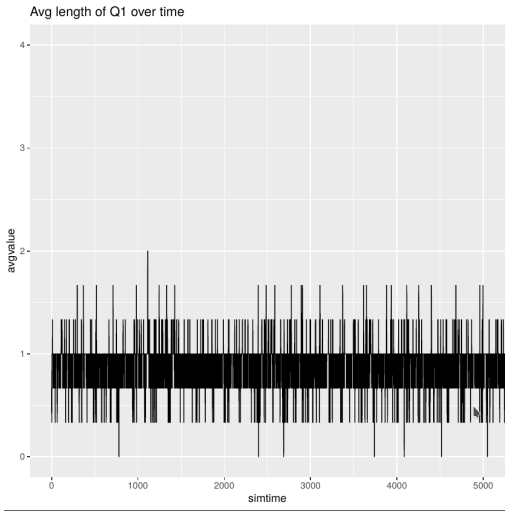
La simulazione è stata parametrizzata in base al processo di arrivo dei pacchetti nel sistema, esponenziale di media 5s, 7.5s, 10s.

4.2 Transiente iniziale

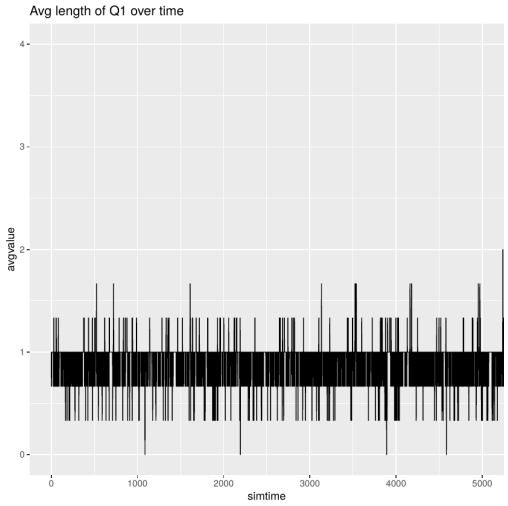
In una simulazione stazionaria si è interessati ad analizzare il comportamento del modello nello *stato stazionario*, raggiunto quando le variabili di interesse iniziano a presentare regolarità statistica. Il periodo antecedente a questo stato è denominato *transiente iniziale* o *warm-up period*.
Per determinare questo intervallo di tempo si è eseguita inizialmente una simulazione preliminare relativamente lunga (*1h*, lunghezza determinata tramite prove ripetute), su tre *run* independenti per parametro, misurando lunghezze delle tre code, tempo di soggiorno e throughput sia per singole *run* che per *averaged*.
Nelle pagine che seguono sono riportati alcuni grafici ottenuti tramite analisi con lo script R *estimate_warmup* e utilizzati per la stima del transiente.
I grafici in questione riportano in modo puntuale lunghezze di Q_1, Q_2, Q_3, tempo di risposta del sistema e throughput del sistema, dove ogni punto è risultato della media delle tre *run*.



(a) p = 5s

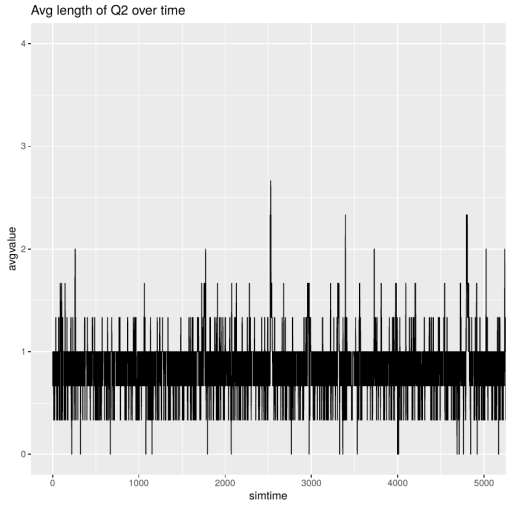


(b) p = 7.5s

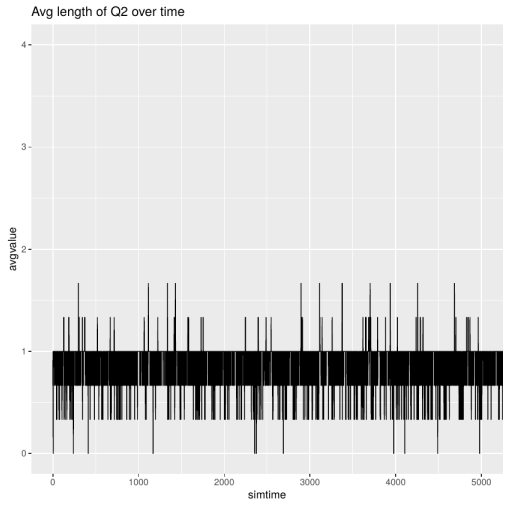


(c) p = 10s

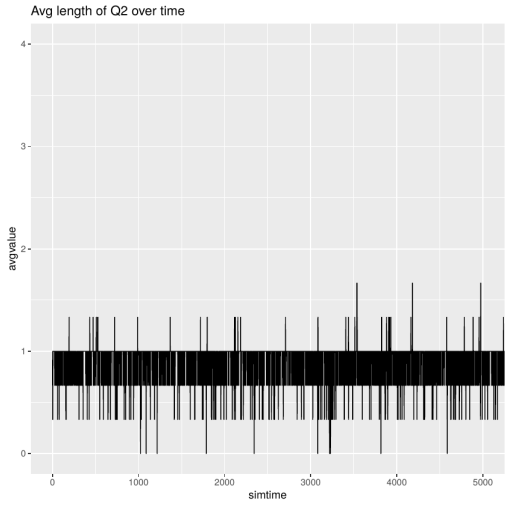
Figura 4: Lunghezza di Q_1 per i tre rispettivi tempi di interarrivo p



(a) p = 5s

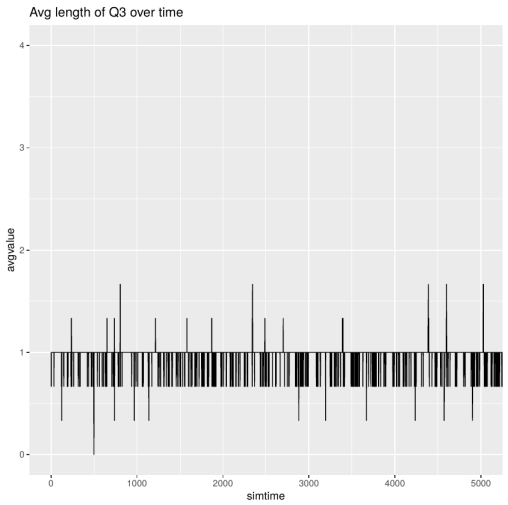


(b) p = 7.5s

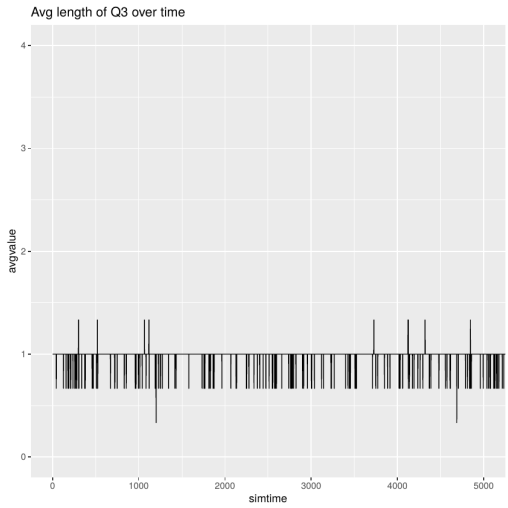


(c) p = 10s

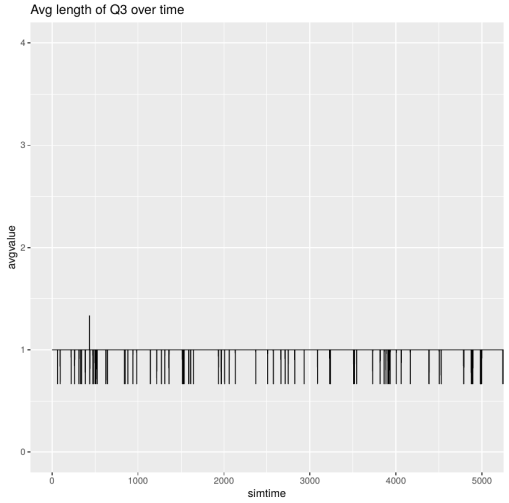
Figura 5: Lunghezza di Q_2 per i tre rispettivi tempi di interarrivo p



(a) p = 5s

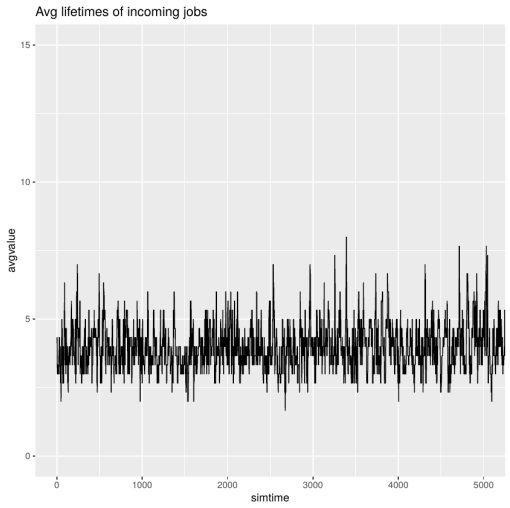


(b) p = 7.5s

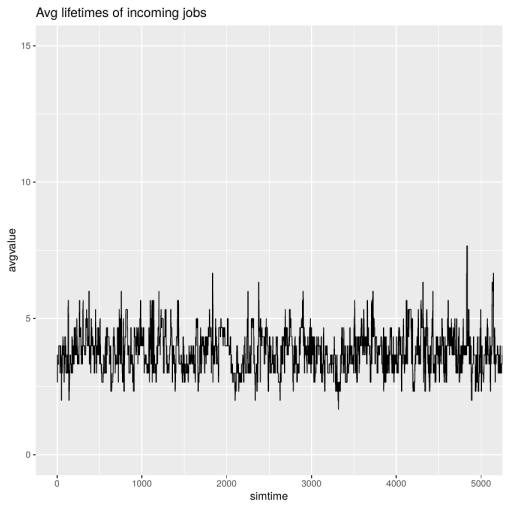


(c) p = 10s

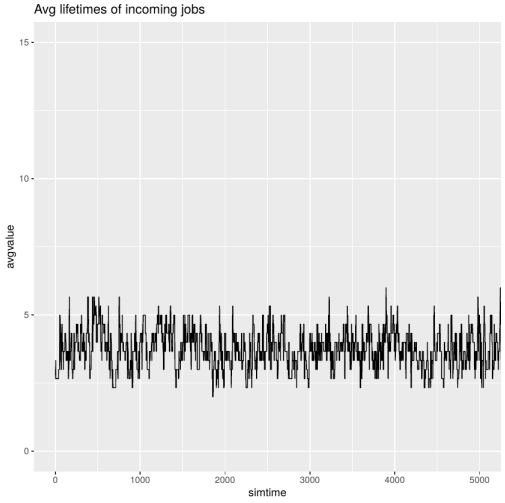
Figura 6: Lunghezza di Q_3 per i tre rispettivi tempi di interarrivo p



(a) p = 5s

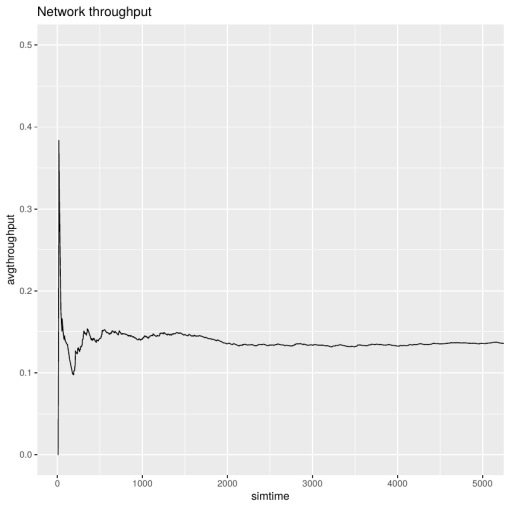


(b) p = 7.5s

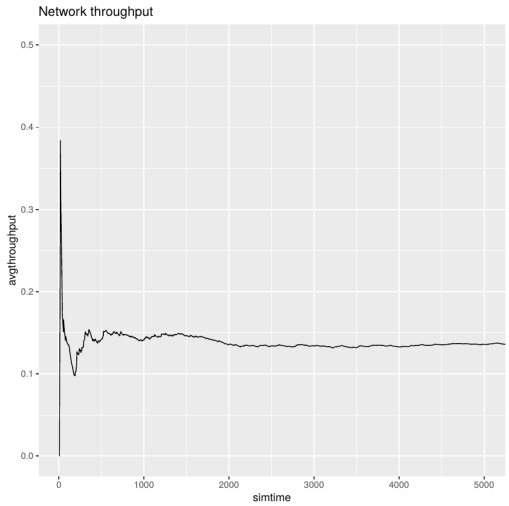


(c) p = 10s

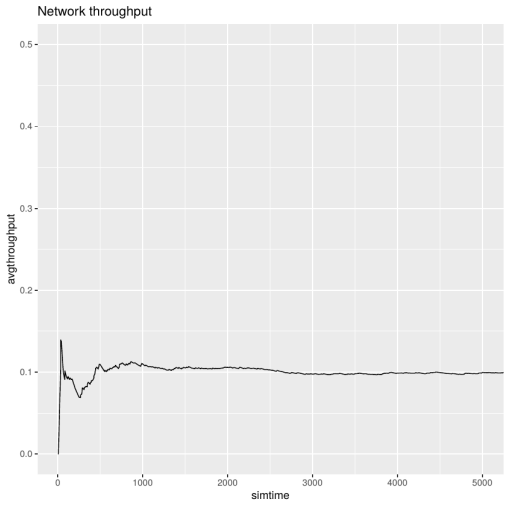
Figura 7: Lifetime di un job per i tre rispettivi tempi di interarrivo *p*



(a) p = 5s



(b) p = 7.5s



(c) p = 10s

Figura 8: Throughput per i tre rispettivi tempi di interarrivo *p*

I grafici che descrivono la lunghezza delle code e il *lifetime* medio di un job non sono informativi, mentre dal plot del throughput medio si può derivare come stima un transiente iniziale di 2000 secondi di *simtime*.
A questo intervallo di tempo si aggiunge un fattore del 20-30% per evitare una sottostima.
Si ottiene quindi il valore di 2600 secondi, che sono stati dati in input al simulatore OMNeT tramite l'opzione di configurazione *warm-up-period* al fine di eseguire la simulazione vera e propria.

4.3 Risultati

La configurazione della nuova simulazione, in seguito all'eliminazione del transiente iniziale, è pensata per essere utilizzata con analisi tramite metodo batch.
Prevede dunque un *running time* di 600000s (circa 150h) molto più lungo del *warm up* e maggiore di quello usato nella simulazione preliminare, e una sola ripetizione per ciascun parametro.
L'analisi batch è eseguita dallo script *analysis.R*, che calcola misure di prestazione (medie) variabili in base al numero di intervalli batch e di osservazioni per batch specificati.
In particolare, l'analisi è stata effettuata per le combinazioni (numero di batch, numero di osservazioni per batch) = (30, 40), (40, 60), (10, 200).
Per ogni stima sono determinate media (batch), varianza e intervalli di confidenza della distribuzione t-student. I risultati sono riportati nelle tabelle presenti nelle pagine successive.

p	q1 length: mean	q1 length: var	q1 length: int l	q1 length: int r
5	0.74333	0.01582	0.70431	0.78235
7.5	0.67500	0.00944	0.64486	0.70514
10	0.62167	0.00839	0.59326	0.65008
p	q2 length: mean	q2 length: var	q2 length: int l	q2 length: int r
5	0.76667	0.02764	0.71510	0.81824
7.5	0.69500	0.01196	0.66107	0.72893
10	0.63833	0.00977	0.60767	0.66899
p	q3 length: mean	q3 length: var	q3 length: int l	q3 length: int r
5	0.77000	0.04217	0.70630	0.83370
7.5	0.69333	0.01771	0.65205	0.73461
10	0.65000	0.01276	0.61496	0.68504
p	lifetime: mean	lifetime: var	lifetime: int l	lifetime: int r
5	3.83960	0.06389	3.76119	3.91801
7.5	3.67062	0.04146	3.60745	3.73379
10	3.62975	0.02790	3.57793	3.68157
p	throughput: mean	throughput: var	throughput: int l	thoughput: int r
5	0.19764	0.00009	0.19470	0.20058
7.5	0.13377	0.00001	0.13279	0.13475
10	0.09729	0.00004	0.09533	0.09925

Tabella 1: numbatch = 30, numobs = 40

p	q1 length: mean	q1 length: var	q1 length: int l	q1 length: int r
5	0.73417	0.00857	0.70951	0.75883
7.5	0.66500	0.00689	0.64289	0.68711
10	0.62500	0.00494	0.60628	0.64372
p	q2 length: mean	q2 length: var	q2 length: int l	q2 length: int r
5	0.76083	0.01435	0.72892	0.79274
7.5	0.67667	0.00810	0.65269	0.70065
10	0.63750	0.00616	0.61659	0.65841
p	q3 length: mean	q3 length: var	q3 length: int l	q3 length: int r
5	0.77083	0.01733	0.73576	0.80590
7.5	0.66500	0.00928	0.63934	0.69066
10	0.63000	0.00609	0.60921	0.65079
p	lifetime: mean	lifetime: var	lifetime: int l	lifetime: int r
5	3.80783	0.03773	3.75608	3.85958
7.5	3.68995	0.01755	3.65466	3.72524
10	3.63671	0.01900	3.59999	3.67343
p	throughput: mean	throughput: var	throughput: int l	thoughput: int r
5	0.19933	0.00004	0.19765	0.20101
7.5	0.13324	0.00005	0.13240	0.13408
10	0.09750	0.00002	0.09631	0.09869

Tabella 2: numbatch = 40, numobs = 60

p	q1 length: mean	q1 length: var	q1 length: int l	q1 length: int r
5	0.73300	0.00247	0.70419	0.76181
7.5	0.66000	0.00156	0.63710	0.68290
10	0.62800	0.00100	0.60967	0.64633
p	q2 length: mean	q2 length: var	q2 length: int l	q2 length: int r
5	0.76000	0.00549	0.71705	0.80295
7.5	0.67300	0.00225	0.64550	0.70050
10	0.64100	0.00074	0.62523	0.65677
p	q3 length: mean	q3 length: var	q3 length: int l	q3 length: int r
5	0.77900	0.01012	0.72069	0.83731
7.5	0.67800	0.00393	0.64166	0.71434
10	0.63100	0.00399	0.59438	0.66762
p	lifetime: mean	lifetime: var	lifetime: int l	lifetime: int r
5	3.80277	0.01357	3.73524	3.87030
7.5	3.69224	0.00295	3.66076	3.72372
10	3.63064	0.00268	3.60063	3.66065
p	throughput: mean	throughput: var	throughput: int l	thoughput: int r
5	0.19866	0.00002	0.19607	0.20125
7.5	0.13312	0.00005	0.13129	0.13495
10	0.09734	0.00001	0.09551	0.09917

Tabella 3: numbatch = 10, numobs = 200

5 Conclusioni

Nel progetto qui presentato si è svolta una simulazione di un sistema a tre code che modellano una rete opportunistica.
Il comportamento del modello è stato implementato in C++ attraverso il simulatore OMNeT++ e l'analisi statistica effettuata in R verteva su tre misure di prestazione, numero medio di utenti in coda, tempo medio di risposta, throughput medio del sistema. Una prima analisi ha consentito di individuare il transiente iniziale, non interessante ai fini statistici. Una volta eliminato il transiente, attraverso l'analisi con metodo batch si sono stimate infine media, varianze e intervalli di confidenza.

Riferimenti bibliografici

[1] A. Al-Hanbali, R. de Haan, R. J. Boucherie, J. van Ommeren, *A Tandem Queueing Model for Delay Analysis in Disconnected Ad Hoc Networks.*

[2] *OMNeT++ Documentation*, https://docs.omnetpp.org/.

[3] *OMNeT++ User Guide*, https://doc.omnetpp.org/omnetpp/UserGuide.pdf.

[4] *OMNeT++ Simulation Manual*, https://doc.omnetpp.org/omnetpp/manual/.

[5] Emmanuel Paradis, *R for Beginners*, https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf.