Instabilità di S-ALOHA

Facci Matteo - 875377

May 2021

Contents

1 Introduzione

In questo progetto si vuole effettuare un'analisi del protocollo S-ALOHA tramite il simulatore Omnet++ e studiarne il suo comportamento. Inoltre effettueremo uno studio del Pure ALOHA, per valutarne e confrontarne le prestazioni in differenti condizioni. Lo scopo sarà quello di verificare quanto i risultati reali ottenuti tramite il simulatore si avvicineranno alle analisi fatte tramite la teoria e capire le caratteristiche e i comportamenti principali dei due protocolli.

2 Strumenti utilizzati

Per la raccolta di dati utilizziamo il simulatore Omnet++[3], un framework di simulazione in C++ che permette tramite la definizione di vari formati di file, di simulare i comportamenti di varie topologie di network. Per l'analisi di questi dati, invece, utilizzeremo Python, con le librerie pandas[6], matplotlib[2] e seaborn[5] che sono librerie specifiche per l'analisi statistica in Python.

3 Definizione dei protocolli S-ALOHA e Pure ALOHA

I protocolli Pure ALOHA e S-ALOHA (abbreviazione di Slotted ALOHA) sono dei protocolli ad accesso casuale che vengono utilizzati a livello MAC per permettere l'accesso a più stazioni al mezzo trasmissivo condiviso. Vengono classificati come 'accesso a contesa': molti nodi in trasmissione condividono un unico canale trasmissivo, quale possono verificarsi collisioni. Ciò che caratterizza i due protocolli sta nel loro modo di trattare il tempo, come vedremo nelle sezioni dedicate al loro funzionamento [4]. Una nota in particolare: faremo inviare frame sempre della stessa lunghezza, poiché ciò consente di avere performance superiori rispetto che inviare frame a lunghezza variabile, come dimostrato nel paper [1].

4 Analisi S-ALOHA

Nel protocollo S-ALOHA il tempo viene diviso in intervalli definiti "slot". In questo modo le stazioni sono forzate a trasmettere negli intervalli di tempo definiti e quindi avremo due possibili situazioni:

- le due stazioni decidono di trasmettere durante un determinato slot e quindi collideranno su quello slot;
- una stazione trasmette in un determinato slot e non ha nessun'altra stazione che collide.

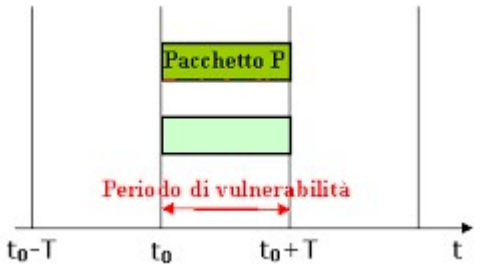

Figure 1: Periodo di vulnerabilità S-ALOHA riferito a 4.1

4.1 Periodo di vulnerabilità

Il periodo di vulnerabilità è quel periodo nella quale le due stazioni vogliono trasmettere i loro frame assieme e vengono spediti nello stesso momento, provocando così una collisione. Nello S-ALOHA può essere spedito un frame solamente in uno slot time, all'inizio. Quindi o un frame viene spedito all'inizio o aspetta il prossimo slot time. Identificando lo slot time come T, lo S-ALOHA ha un periodo di vulnerabilità T.

4.2 Throughput teorico di S-ALOHA

Definiamo innanzitutto il throughput. Il throughput è definito come numero totale di trasmissioni con successo per frame time. L'analisi delle performance di S-ALOHA può essere fatta in due situazioni. La prima è quella nel caso il sistema sia analizzato in una situazione in cui non ci siano nodi backlogged (analisi statica) e la seconda invece abbiamo situazioni di ritrasmissione (analisi dinamica).

4.3 Analisi statica

Vogliamo ottenere la probabilità che il nostro frame sia l'unico inviato nello slot corrente e che non ci sia nessun'altra stazione che invii frame nel mezzo condiviso. Possiamo simulare questo evento con una distribuzione di Poisson.

Definiamo queste variabili per poter effettuare l'analisi statica:

- k, numero di eventi che voglio si verifichi (ovvero voglio che nel periodo di vulnerabilità T vi sia solamente un frame in trasmissione).
- G, numero medio di frame generati nel intervallo T;
- λ, tasso di arrivo in percentuale della nostra rete;
- N(t), frame inviati fino all'istante t;
- N(t+T), frame inviati fino all'istante t+T.

$$P[(N(t+T) - N(t)) = k] = e^{-G}\frac{G^k}{k!} \tag{1}$$

E' la probabilità che si verifichino k numero di trasmissioni sapendo che in media vi sono G trasmissioni. Per ottenere una trasmissione corretta sappiamo che dobbiamo avere solamente 1 invio per slot e di conseguenza il numero di eventi che vogliamo avere nell'intervallo (k) dev'essere posto a 1.

$$P[(N(t+T) - N(t)) = 1] = e^{-G}G \tag{2}$$

Dobbiamo trovare il valore di G che massimizza la probabilità sopra. Lo facciamo effettuando la derivata e ponendola uguale a 0. Sviluppando otteniamo quindi:

$$e^{-G} - Ge^{-G} = e^{-G}(1 - G) = 0 \tag{3}$$

Da cui abbiamo che G=1. Sostituendo il valore di G alla probabilità di prima otteniamo:

$$P[(N(t+T) - N(t)) = 1] = \frac{1}{e} = 0.36787.. \tag{4}$$

Il throughput massimo quando abbiamo il numero massimo di frame inviati (1 per slot) quindi è $\frac{1}{e}$.

Dato che λ_i è il tasso di arrivo da ogni nodo, quindi avremo

$$\sum_{i=1}^{m} \lambda_i = \lambda$$

Siano m il numero delle stazioni presenti nella nostra rete supponiamo sia m$< \infty$ con m un numero molto grande. Se abbiamo $\lambda <1$ e G<1 allora la rete si comporta bene. Se abbiamo invece $\lambda >1$ e G>1 allora entriamo in un caso di congestion collapse, dove il throughput inizia a diminuire dato l'elevato tasso di arrivo dei nodi, che provoca un numero di collisioni maggiori.

4.4 Analisi dinamica

Effettuiamo ora un'analisi basandoci sul fatto che i nodi della rete possono essere backlogged e possono quindi effettivamente essere in ritrasmissione.

Possiamo modellare S-ALOHA con una catena di Markov discreta (DTMC). Definiamo queste variabili per poter effettuare l'analisi dinamica:

- m, numero di nodi totale.
- n, numero di nodi backlogged nella nostra rete, in un dato istante;
- q_a, probabilità che un nodo non backlogged invii un frame q_a=$1 - e^{\frac{\lambda}{m}}$;
- q_r, probabilità che un nodo backlogged ritrasmetta il frame che ha in coda;

- $Q_a(i,n)$, probabilità che i nodi non backlogged trasmettano contemporaneamente;
- $Q_r(i,n)$, probabilità che i nodi backlogged trasmettano contemporaneamente;

$$Q_a = \binom{m-n}{i}(1-q_a)^{m-n-i}q_a^i \tag{5}$$

$$Q_r = \binom{n}{i}(1-q_r)^{n-i}q_r^i \tag{6}$$

Nel nostro sistema possiamo avere m+1 stati, da n=0 stazioni backlogged a n=m stazioni backlogged. Possibili transizioni:

- da n a n-1: se un nodo riesce ad uscire dallo stato backlogged effettuando una ritrasmissione correttamente.
- da n a n+i: quando i, $i > 1$, nodi non backlogged tentano di trasmettere, ma falliscono collidendo e otteniamo i nodi backlogged in più.
- rimango in n (nessun nodo diventa backlogged): se un nodo non backlogged riesce a trasmettere o se nessun nodo prova a trasmettere e nessun nodo backlogged trasmette con successo (condizione che fa rimanere i nodi backlogged nello stesso numero).

Dopo aver elencato le possibili transizioni definiamo la matrice di transizione come la seguente:

$$P_{n,n+i} = \begin{cases} Q_a(i,n), & 2 \leq i \leq m-n \\ Q_a(i,n)[1-Q_r(0,n)], & i = 1 \\ Q_a(1,n)Q_r(0,n) + Q_a(0,n)[1-Q_r(1,n)], & i = 0 \\ Q_a(0,n)Q_r(1,n), & i = -1 \end{cases} \tag{7}$$

Per studiare il regime stazionario di S-ALOHA dobbiamo definire il Drift. D_n è definito come drift dello stato n e indica il numero medio di terminali che passeranno nello stato backlogged dallo stato n.

$$D_n = X_{n+1} - X_n \tag{8}$$

Possiamo scrivere D_n anche come

$$D_n = (m-n)q_a - P_s \tag{9}$$

dove:

- (m-n)$q_a$indica la possibilità di trasmettere un frame da parte delle stazioni non backlogged.

- P_s indica la probabilità di successo di un frame allo stato n / tasso di uscita.

Definiamo P_s come segue:

$$P_s = Q_a(1,n)Q_r(0,n) + Q_a(0,n)Q_r(1,n) \tag{10}$$

Definiamo invece G(n) come il numero di tentativi di trasmissione totale nella rete nello stato n:

$$G(n) = (m-n)q_a + nq_r \tag{11}$$

Dimostriamo che $G(n)e^{-G(n)} = P_s$

L'equazione 11 mostra come, in questo caso, il numero di tentativi di trasmissione totali dipenda dal parametro di ritrasmissioni n. Dn diventa quindi:

$$D_n = (m-n)q_a - G(n)e^{-G(n)} \tag{12}$$

In cui vale:

- $(m-n)q_a$ indica il tasso dei frame in arrivo nella rete
- $G(n)e^{-G(n)}$ indica il tasso dei frame in uscita nella rete

Il drift è ottenuto dalla differenza delle due. Nella situazione "fortunata" nessuna stazione è backlogged ($n = 0$), quindi avremo il picco di prestazioni della nostra rete. Nella situazioni in cui, invece, tutti i nostri nodi sono backlogged, la rete è bloccata poiché nessun nodo può inviare frame. Nella situazione di analisi statica, avendo $m \to \infty$, il tasso di arrivo non dipendeva da n (poiché $m - n = m$), mentre nel caso dell'analisi dinamica n influenza sicuramente il throughput della nostra rete, andando a rallentare in maniera estrema la nostra rete.

5 Analisi Pure-ALOHA

Il modello Pure ALOHA è il modello quale si basa lo S-ALOHA, in cui non vi è una suddivisione del tempo in slot, al contrario di S-ALOHA. Questo significa che un nodo della rete non è forzato ad inviare un messaggio all'interno di un determinato slot ma può inviarlo appena il frame è pronto.

5.1 Periodo di vulnerabilità

Sia t il tempo necessario per l'invio di un frame da un nodo. Supponiamo che il frame venga inviato in un tempo t_0. Il periodo in cui viene inviato un frame quindi è da t_0 a $t_0 + t$. Se volessimo trasmettere il frame correttamente nessun frame dovrebbe essere inviato in questo periodo. In realtà, però, per essere inviato il frame ha bisogno di un altro periodo t per l'invio dato che, se il frame partisse dopo t_0, nessun frame potrebbe essere inviato nemmeno nel intervallo $t_0 + 2t$, altrimenti colliderebbe con il frame appena inviato. Il periodo di vulnerabilità è quindi 2T.

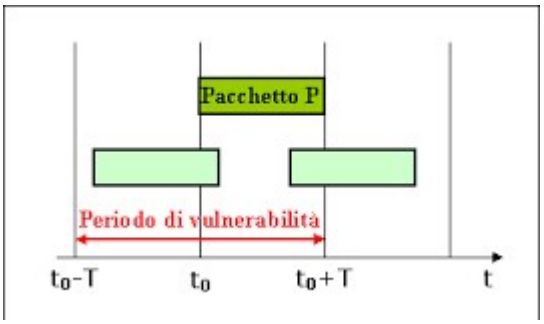

Figure 2: Periodo di vulnerabilità Pure-ALOHA

5.2 Throughput teorico del Pure ALOHA

5.3 Analisi statica

Per analizzare il throughput teorico del Pure ALOHA possiamo riferirci al throughput dello S-ALOHA. Infatti, quest'ultimo è identico ad eccezione del periodo di vulnerabilità. La proprietà che k frame siano generati in un tempo di frame definito, in cui ci si aspetta G frame generati, è data dalla seguente distribuzione di Poissant:

$$P[(N(t+T) - N(t)) = k] = e^{-2G}\frac{G^k}{k!} \tag{13}$$

poiché il periodo di vulnerabilità è di 2T la media di frame generati non è G, ma 2G. Siccome k=1 allora il throughput è il seguente:

$$P[(N(t+T) - N(t)) = 1] = e^{-2G}G \tag{14}$$

Come prima, troviamo il valore di G che massimizza il throughput derivando la probabilità sopra e poniamo il suo valore a 0 per capire quando la probabilità è massimizzata.

$$e^{-2G}(1-2G) = 0 \tag{15}$$

Otteniamo quindi che G=$\frac{1}{2}$. Sostituendo G otteniamo che:

$$P[(N(t+T) - N(t)) = 1] = \frac{1}{2e} = 0.18393972058.. \tag{16}$$

Questo significa che, nel migliore dei casi, l'utilizzazione del canale trasmissivo sarà del 18%.

6 Simulazioni e analisi delle performance con Omnet++

In questa sezione ci approcceremo in questo modo: innanzitutto simuleremo i protocolli Pure ALOHA e S-ALOHA su una rete tramite il simulatore Om-

net++. Ne effettueremo un analisi statica per vedere se i risultati ottenuti nella teoria combaciano nella simulazione. Prima però, osserviamo quali sono i parametri che possono influenzare le nostre simulazioni e vediamo come il throughput varia in seguito.

I parametri comuni che vengono valutati sia nella simulazione S-ALOHA sia per il Pure ALOHA sono i seguenti:

- 9.6kps, rate di trasmissione degli host;
- 952b, lunghezza dei frame inviati (in bit).

Per il numero degli host effettuiamo un confronto tra 10, 15, 20 e 50 host, per vedere come cambia il comportamento quando i nodi della rete aumentano.

I parametri che possono influenzare il throughput di S-ALOHA e del Pure ALOHA sono:

- il numero degli host;
- lunghezza dello slot time;
- il tempo che intercorre tra l'invio di ogni trasmissione.

6.1 Definizione della topologia della rete in Omnet++

6.1.1 Modulo del server

Nel modulo del server, sono definiti all'interno della sua classe i seguenti parametri per la registrazione dei dati durante la simulazione. Parametri:

- **double** x, coordinata x del server;
- **double** y, coordinata y del server;
- Signal (parametri che sono utilizzati per l'invio di segnali all'accadere di un determinato evento):
  - **long** receiveBegin, utilizzato per indicare quanti frame arrivano al server. Se il server non riceve più frame allora verrà settato a 0;
  - **long** receive, indica lo stato della trasmissione: 1 se è avvenuta con successo, 0 se è fallita;
  - **long** collision, indica il numero dei frame collisi nel periodo di slot time;
  - **simtime_t** collisionLength, indica la lunghezza dell'ultima collisione nel periodo di slot time;
  - **long** channelState, indica lo stato del canale trasmissivo (0=arrestato, 1=trasmissione, 2=collisione).
- Statistic (parametri che sono utilizzati per raccogliere statistiche, alcuni di essi usano i segnali descritti sopra):

- serverChannelState, utilizza il parametro "channelState" per indicarne il suo stato;
- receiveBegin, mostra il parametro "receiveBegin";
- channelUtilization, viene effettuata una media dell' utilizzo del canale effettuando "timeavg(receive)", calcolando quindi quanto il canale viene utilizzato in media (da 0 a 1);
- collisionMultiplicity, viene mostrato un istogramma in cui vengono raffigurate la molteplicità delle collisioni. Avremo quindi una barra più o meno alta in base a quante collisioni sono avvenute nella nostra simulazione;
- receivedFrames, "sum(receive)", vengono calcolati quanti sono i frame ricevuti correttamente;
- collidedFrames, "sum(collision)", vengono calcolati quanti sono i frame collisi.

6.1.2 Modulo del host

Come per il server, anche gli host hanno il loro modulo con all'interno le variabili dichiarate e parametri specificati per la raccolta dei dati. I dati statistici raccolti qui non sono molti, ma sono definite importanti variabili per la definizione delle proprietà del host.

Parametri:

- **long** state, indica lo stato di trasmissione del host (parametro con decorator signal, vedi 6.1.1);
- radioState, unica statistica del host, ritorna se la stazione sta trasmettendo(1) oppure no(0) (parametro con decorator statistic, vedi 6.1.1);
- **double** iaTime, tempo di interarrivo dei pacchetti;
- **double** txRate, tasso di trasmissione;
- **volatile int** pkLenBits, indica la lunghezza dei pacchetti in bits;
- **double** slotTime, lunghezza dello slot time;
- **double** x, coordinata x del host;
- **double** y, coordinata y del host;

6.2 Rete ALOHA

L'ultimo modulo che viene definito è quello della rete, per definire la topologia della rete nel dettaglio.

Parametri:

- **int** numHosts, indica il numero di host;

- **double** txRate, indica il trasmission rate;
- **double** slotTime, indica la durata in ms dello slot;

Nella classe è definito anche un submodule in cui vengono istanziate le due classi create in precedenza: server e host.

6.2.1 Omnetpp.ini - File di configurazione

In questo file vengono assegnati tutti i parametri per le varie configurazioni della rete. Sono definite in totale 7 configurazioni su cui andremo ad eseguire le simulazioni. Con numHosts=20 abbiamo:

- pure ALOHA, congestionato, iaTime modellata con un'esponenziale di parametro 1s;
- pure ALOHA, carico ottimale , iaTime modellata con un'esponenziale di parametro 4s. In questo caso con l'evento dell'invio che si ripete in media ogni 4 secondi, abbiamo una performance di $\frac{1}{2e}$, che è ottimale per il Pure ALOHA.
- pure ALOHA, basso carico, iaTime modellata con un'esponenziale di parametro 30s;
- Slotted ALOHA, congestionato, iaTime modellata con un'esponenziale di parametro 0.5s con slot time di 100ms;
- Slotted ALOHA, carico ottimale, iaTime modellata con un'esponenziale di parametro 2s, con slot time di 100ms. Se inviamo pacchetti ogni 2 secondi otterremo un throughput ottimale per lo s-ALOHA, che è di $\frac{1}{e}$;
- Slotted ALOHA, basso carico, iaTime modellata con un'esponenziale di parametro 20s, con slot time di 100ms;
- configurazione avanzata del pure Aloha, in cui sono specificati:
  - repeat, il numero di ripetizioni della simulazione;
  - sim-time-limit, durata limite della simulazione;
  - un array con diverse cardinalità di host;
  - iaTime, modellata con una distribuzione esponenziale che utilizzerà un parametro in secondi, che indicheremo come λ.

Nella figura 7 abbiamo un esempio di ciò che osserviamo prima della simulazione con Omnet++ tramite GUI: in questo caso vi sono 20 host in una rete che utilizzano il Pure ALOHA per l'accesso al mezzo.

```
simple Server
{
    parameters:
        @display("i=device/antennatower_l");
        @signal[receiveBegin](type="long");  // increases with each new frame arriving to the server and drops to 0 if the channel
becomes once idle
        @signal[receive](type="long");  // for successful receptions (non-collisions): 1 at the start of the reception, 0 at the end
of the reception
        @signal[collision](type="long"); // the number of collided frames at the beginning of the collision period
        @signal[collisionLength](type="simtime_t");  // the length of the last collision period at the end of the collision period
        @signal[channelState](type="long");

        double x @unit(m); // the x coordinate of the server
        double y @unit(m); // the y coordinate of the server

        double animationHoldTimeOnCollision @unit(s) = default(0s); // in animation time

        @statistic[serverChannelState](source="channelState";title="Channel state";enum="IDLE=0,TRANSMISSION=1,COLLISION=2";
record=vector);
        @statistic[receiveBegin](source="receiveBegin"; record=vector?; interpolationmode=sample-hold; title="receive begin");
        @statistic[channelUtilization](source="timeavg(receive)"; record=last; interpolationmode=linear; title="channel
utilization");
        @statistic[collisionMultiplicity](source=collision; record=vector?,histogram; title="collision multiplicity");
        @statistic[collisionLength](record=vector?,histogram,mean,sum,max; title="collision length");
        @statistic[receivedFrames](source="sum(receive)"; record=last; title="received frames");
        @statistic[collidedFrames](source="sum(collision)"; record=last; title="collided frames");
    gates:
        input in @directIn;
}
```

Figure 3: Modulo del server

```
simple Host
{
    parameters:
        @signal[state](type="long");
        @statistic[radioState](source="state";title="Radio state";enum="IDLE=0,TRANSMIT=1";record=vector);
        double txRate @unit(bps);          // transmission rate
        volatile int pkLenBits @unit(b);   // packet length in bits
        volatile double iaTime @unit(s);   // packet interarrival time,
        double slotTime @unit(s);          // zero means no slots (pure Aloha)
        double x @unit(m);                 // the x coordinate of the host
        double y @unit(m);                 // the y coordinate of the host
        double idleAnimationSpeed;         // used when there is no packet being transmitted
        double transmissionEdgeAnimationSpeed; // used when the propagation of a first or last bit is visible
        double midTransmissionAnimationSpeed; // used during transmission
        bool controlAnimationSpeed = default(true);
        @display("i=device/pc_s");
}
```

Figure 4: Modulo dell'host

```
network Aloha
{
    parameters:
        int numHosts;  // number of hosts
        double txRate @unit(bps);  // transmission rate
        double slotTime @unit(ms);  // zero means no slots (pure Aloha)
        @display("bgi=background/terrain,s;bgb=1000,1000");
    submodules:
        server: Server;
        host[numHosts]: Host {
            txRate = txRate;
            slotTime = slotTime;
        }
}
```

Figure 5: Network di ALOHA

```
[General]
network = Aloha
#debug-on-errors = true
#record-eventlog = true

Aloha.numHosts = 20
Aloha.slotTime = 0s    # no slots
Aloha.txRate = 9.6kbps
Aloha.host[*].pkLenBits = 952b #=119 bytes, so that (with +1 byte guard) slotTime is a nice round number

**.x = uniform(0m, 1000m)
**.y = uniform(0m, 1000m)
**.animationHoldTimeOnCollision = 0s
**.idleAnimationSpeed = 1
**.transmissionEdgeAnimationSpeed = 1e-6
**.midTransmissionAnimationSpeed = 1e-1

[Config PureAloha1]
description = "pure Aloha, overloaded"
# too frequent transmissions result in high collision rate and low channel utilization
Aloha.host[*].iaTime = exponential(2s)

[Config PureAloha2]
description = "pure Aloha, optimal load"
# near optimal load, channel utilization is near theoretical maximum 1/2e
Aloha.host[*].iaTime = exponential(6s)

[Config PureAloha3]
description = "pure Aloha, low traffic"
# very low traffic results in channel being idle most of the time
Aloha.host[*].iaTime = exponential(30s)

[Config PureAlohaExperiment]
description = "Channel utilization in the function of packet generation frequency"
repeat = 2
sim-time-limit = 3min
#**.vector-recording = false
Aloha.numHosts = ${numHosts=10,15,20,50}
Aloha.host[*].iaTime = exponential(${iaMean=1,5,10,15,20,25..300 step 30}s)

[Config SlottedAloha1]
description = "slotted Aloha, overloaded"
# slotTime = pkLen/txRate = 960/9600 = 0.1s
Aloha.slotTime = 100ms
# too frequent transmissions result in high collision rate and low channel utilization
Aloha.host[*].iaTime = exponential(0.5s)

[Config SlottedAloha2]
description = "slotted Aloha, optimal load"
# slotTime = pkLen/txRate = 960/9600 = 0.1s
Aloha.slotTime = 100ms
# near optimal load, channel utilization is near theoretical maximum 1/e
Aloha.host[*].iaTime = exponential(2s)

[Config SlottedAloha3]
description = "slotted Aloha, low traffic"
# slotTime = pkLen/txRate = 960/9600 = 0.1s
Aloha.slotTime = 100ms
# very low traffic results in channel being idle most of the time
Aloha.host[*].iaTime = exponential(20s)
```

Figure 6: File di configurazione omnetpp.ini

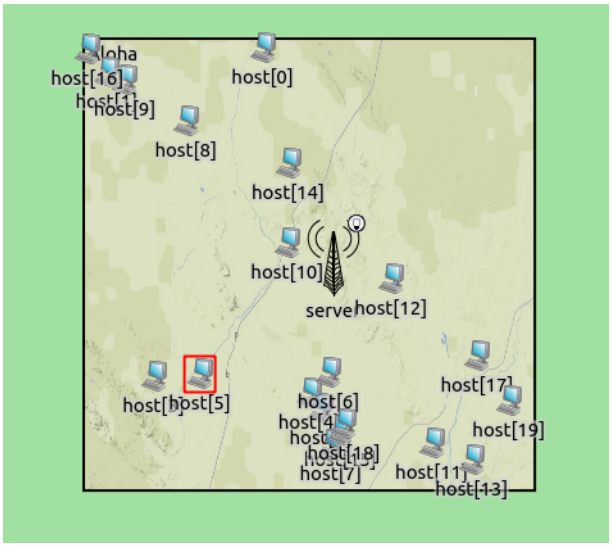

Figure 7: Topologia della rete con 20 host.

6.3 Analisi delle simulazioni sul Pure ALOHA e S-ALOHA

Per il numero degli host, se aumentiamo il numero degli host avremo un aumento direttamente proporzionale delle collisioni dovuto al grande numero di utenti che proverà a trasmettere, quindi il numero di utenti backlogged n aumenterà di conseguenza. Se il numero di utenti $m \to \infty$ allora il throughput$\to 0$. Se diminuiamo invece il tempo che intercorre tra l'invio dei frame, aumenteranno i frame in media in un periodo T sul mezzo condiviso e quindi, il mezzo condiviso non riuscirà a ricevere un frame correttamente: data il grande numero di frame in arrivo, essi collideranno tra di loro.

Nell'immagine 8 sono rappresentate le prestazioni del Pure ALOHA testandolo in base ad un numero diverso di host (posti sulle ordinate) e un intervallo diverso della generazione dei frame (λ, ascisse).

Nell'immagini 10 e 11 possiamo notare il drastico calo di prestazioni delle reti Pure e S-ALOHA quando il canale non viene sfruttato abbastanza e la generazione dei frame avviene con una bassa frequenza nella rete.

Come possiamo osservare, otteniamo un throughput$\approx$0.18 per il Pure ALOHA $\approx$0.36 per lo S-ALOHA, come ci aspettavamo dai calcoli nel paragrafo 16 e 4. Notiamo però che il momento in cui lo otteniamo varia in base al numero dei nodi presenti nella rete: questo perché ogni simulazione raggiungerà il suo picco di throughput ad un diverso carico della rete. Quello che però osserviamo è che per ogni numero di hosts$\lambda \to \infty$ otterremo throughput$\to 0$, come mostrato nella figura 11.

Infatti, nell'immagine 11 analizziamo lo specifico caso di una rete con 50 hosts. Osserviamo che il throughput massimo si raggiunge se gli hosts generano

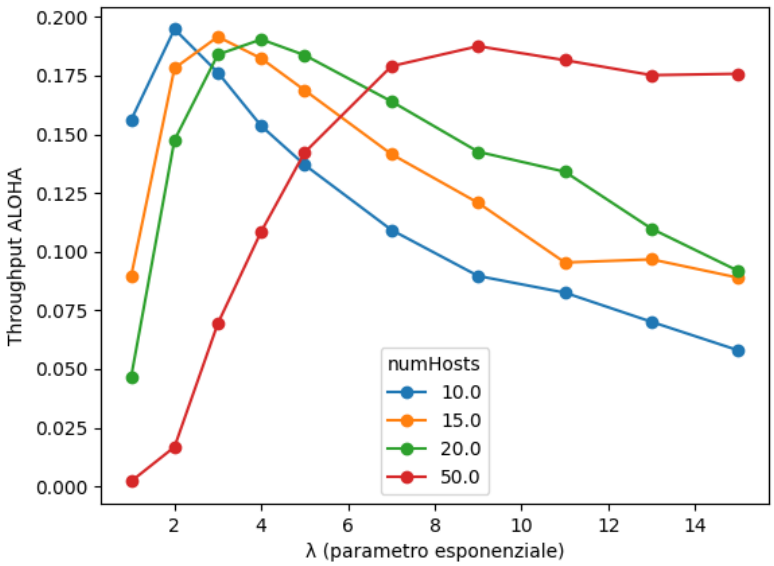

Figure 8: Analisi throughput del Pure ALOHA in caso vi siano 10, 15, 20 e 50 hosts.

frame con una esponenziale di 20 secondi circa. Notiamo anche che quando ogni nodo genera frame molto velocemente, mantiene delle prestazioni abbastanza buone $\approx$0.135. Infine, quando i messaggi generati aumentano, il throughput crolla e le prestazioni diminuiscono di drasticamente. Questa è la conferma che, per $\lambda \to \infty$ il throughput$\to 0$.

6.4 Influenza dello slot time

Nel Pure-ALOHA non abbiamo il tempo scandito in slot, quindi questo aspetto non può influenzare il throughput della nostra simulazione.

Al contrario del Pure-ALOHA lo S-ALOHA è per definizione scandito in slot, vediamo come questo aspetto incide sul throughput. Innanzitutto, per ricavare la lunghezza ideale dello slot nello S-ALOHA possiamo effettuare lunghezza dei frame/rateo di trasmissione, ottenendo così una durata del canale in questo caso di 0.1 secondi. La lunghezza del canale espone il canale stesso a più frame che possono essere contemporaneamente inviati e quindi aumenta anche la possibilità che vi siano collisioni. La variazione di throughput al cambio della durata del canale è una conseguenza diretta alle diverse lunghezze dei frame e il loro rateo di trasmissione, dato che esse determinano la lunghezza del canale. In particolare, avremo un throughput basso quando la lunghezza dei frame sarà molto lunga e il rateo di trasmissione sarà basso, perché il server vedrà arrivare

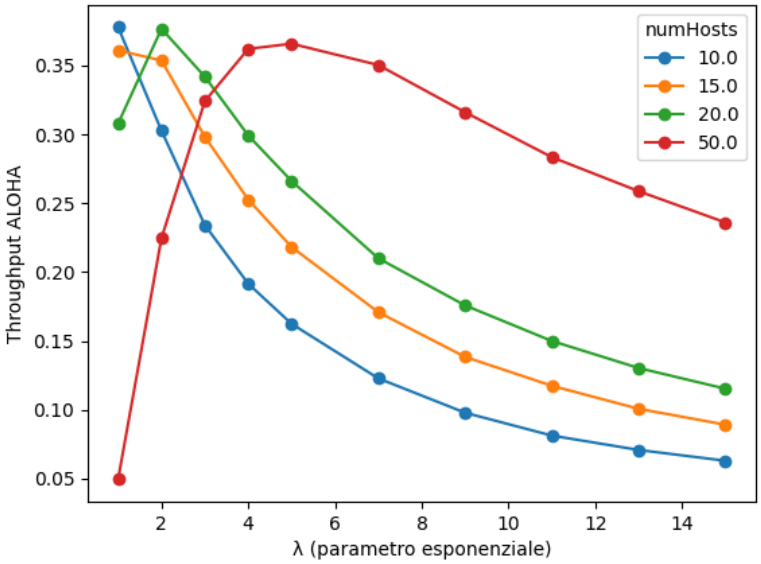

Figure 9: Analisi throughput dello S-ALOHA in caso vi siano 10, 15, 20 e 50 hosts.

molti pacchetti di grandi dimensioni in un periodo T breve, e quindi questo porterà ad una collisione.

6.5 Metodo grafico di Welch

Il metodo di Welch ci è necessario per capire in che momento la nostra simulazione stabilizza il suo valore di throughput, dato che noi sappiamo che per t$\rightarrow \infty$ sicuramente stabilizzerà il suo valore ma non possiamo effettivamente simularla fino a t$\rightarrow \infty$. Per questo usiamo il metodo di Welch e tramite una window w costruiamo la sequenza di m-w variabili casuali, dove m era il numero delle nostre iniziali osservazioni (nel nostro caso 50).

- m=50, numero delle osservazioni: prendiamo una simulazioni con durata 1,3,5 fino a 100(s) con step 2, ottenendo così 50 simulazioni di durata differenti;
- seed=-1, utilizziamo lo stesso seme per porre una correlazione all'intervallo di tempo in cui verranno generati i pacchetti dalla distribuzione esponenziale (altrimenti non potremmo fare questa osservazione);
- slot-time=100ms (solo sullo S-ALOHA);
- numHost=15;

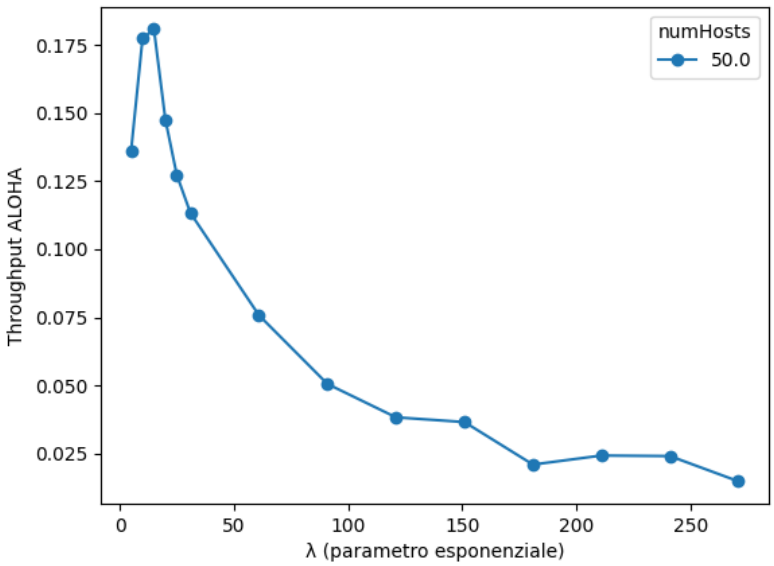

Figure 10: Analisi del Pure ALOHA nel particolar caso in cui abbiamo 50 host nella rete

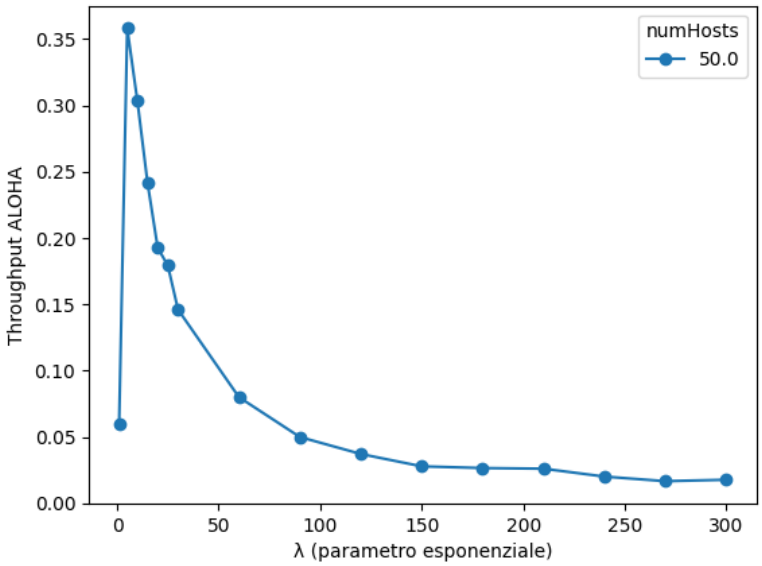

Figure 11: Analisi dello S-ALOHA nel particolar caso in cui abbiamo 50 host nella rete

- iaMean=3.

Per verificare la stabilità del nostro valore utilizziamo windows di tre dimensioni: 10, 20 e 30. Da 12 e 14 possiamo osservare che la nostra simulazione si stabilizza a partire da 57 secondi circa. Notiamo che nella figura 12 la media del throughput ottenuta è maggiore del throughput massimo del Pure ALOHA stesso ma questo è solamente perché la media è "truccata" dai primi valori ottenuti nel grafico stesso, che essendo alti fanno innalzare di molto la media. Se continuassimo le simulazioni per una durata maggiore di 100s, vedremmo che il throughput si stabilizzerebbe sul throughput medio reale (16). In questo caso il Pure ALOHA necessiterebbe di ulteriori simulazioni con durata maggiore di 100s. Nel caso dello S-ALOHA questo non si verifica, poiché notiamo che da 57s in poi il throughput resta stabile vicino al valore medio e non si discosterà di molto da quel valore, come indicato dai comportamenti delle medie mobili sulle varie windows (15).

6.6 Intervalli di confidenza

Supponiamo che dopo aver effettuato 10 run indipendenti otteniamo 10 campioni Xi, istanze di variabili casuali i.i.d. e vogliamo stimarne la media. La media in realtà è una variabile casuale poiché se ripetessimo l'esperimento altre n volte e calcolassimo di nuovo la media, essa sarebbe diversa. Ecco che entra in gioco l'intervallo di confidenza. Un intervallo di confidenza δ ci permette di capire con che probabilità il valore della media reale rientra nell'intervallo da noi calcolato. Se il CI è stretto allora posso aspettarmi una stima calcolata precisa, altrimenti la misurazione da noi effettuata non risulterà precisa.

Nella figura 16 abbiamo un intervallo di confidenza del 95 percento, rappresentato come una zona che circonda ognuno dei grafici. Questo significa che per ogni grafico abbiamo il 95% di possibilità che la media reale cada su quel intervallo. Ad esempio, se dovessimo effettuare la media per sample size$\rightarrow \infty$ avremmo il 95% che la media sia entro il confidence level. In 16 e 17 i confidence level sono molto stretti quindi possiamo essere abbastanza sicuri che la nostra stima sia accurata. Osservando il grafico 16 possiamo determinare come nel caso medio il nostro throughput si avvicini a quello teorico calcolato nel equazione 16. Un'ultima considerazione da fare è quella risultante il simulation-time-limit: se impostiamo un valore elevato, la simulazione può potenzialmente simulare per più tempo e quindi il risultato della nostra variabile casuale indipendente si avvicinerà molto alla realtà. Se impostiamo un tempo di simulazione limite molto alto il valore delle variabili casuali i.i.d convergerà su un unico valore, quindi il nostro intervallo di confidenza sarà molto stretto. Nella figura 18 possiamo vedere come con 90 min di simulation-time-limit il valore delle varie variabili casuali i.i.d generate sia pressoché lo stesso.

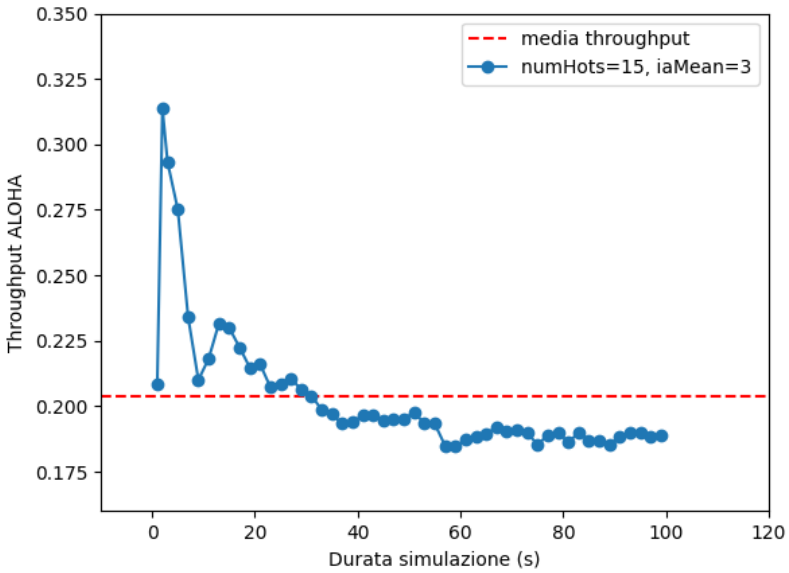

Figure 12: Analisi del Pure ALOHA con differenti tempi di simulazione

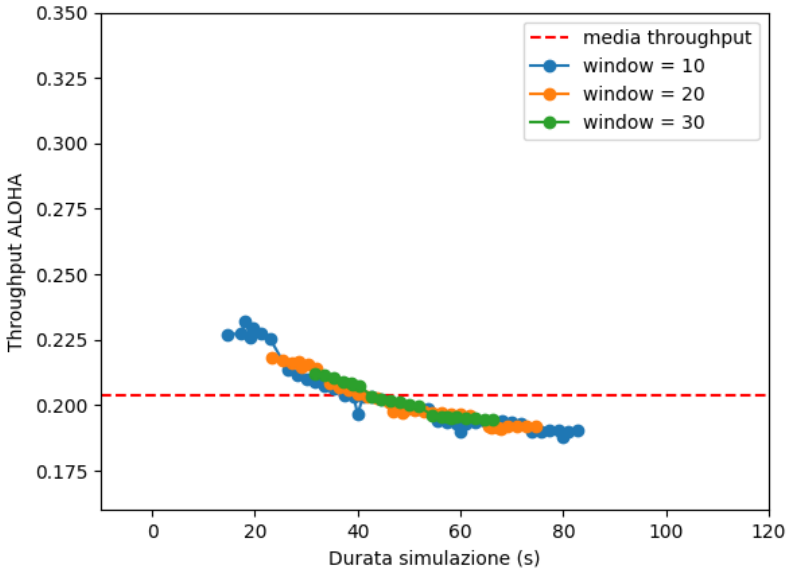

Figure 13: Media mobile calcolata su finestre di varie dimensioni riferite alla figura 12

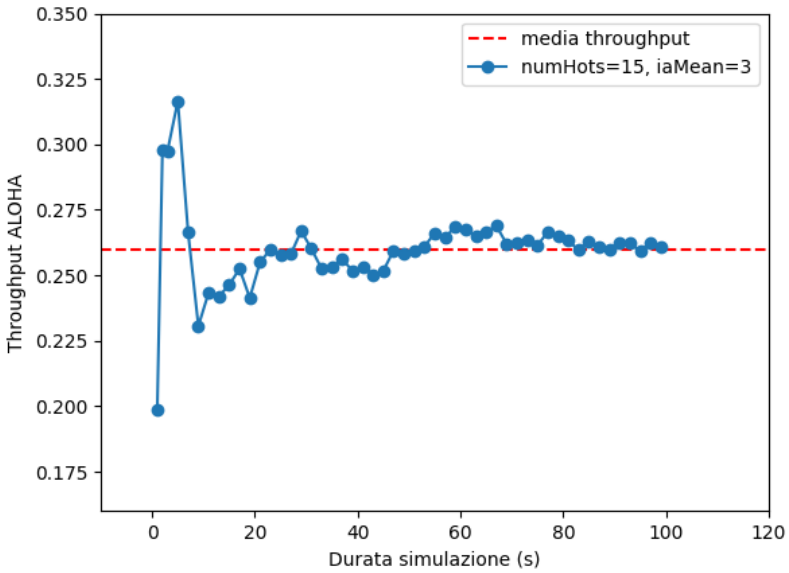

Figure 14: Analisi dello S-ALOHA con differenti tempi di simulazione

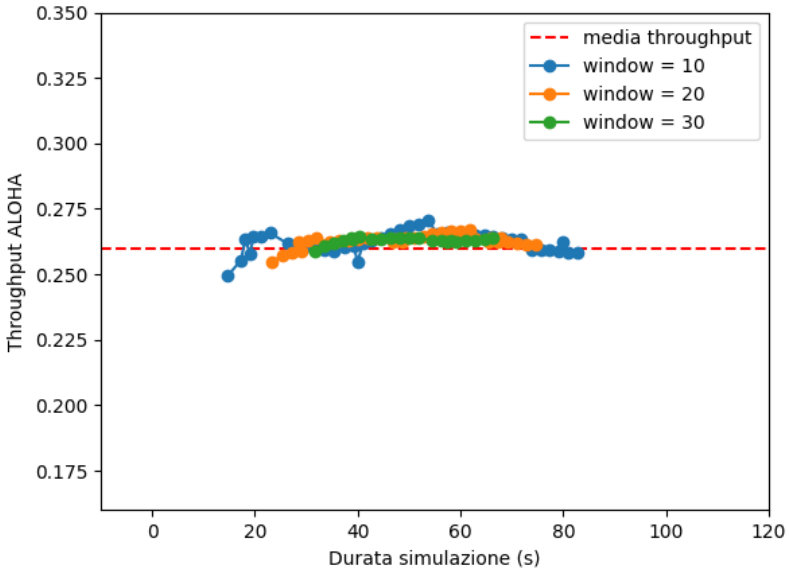

Figure 15: Media mobile calcolata su finestre di varie dimensioni riferite alla figura 14

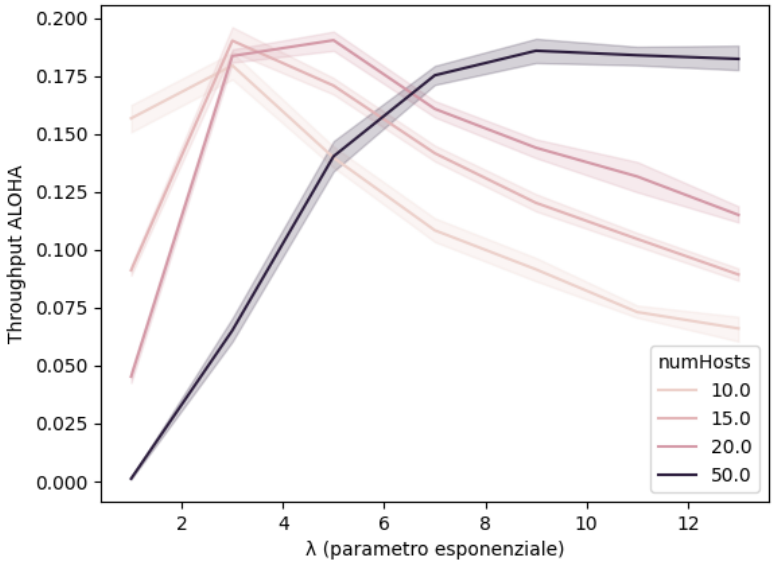

Figure 16: Pure ALOHA con intervallo di confidenza dello .95 su un sample size di 10

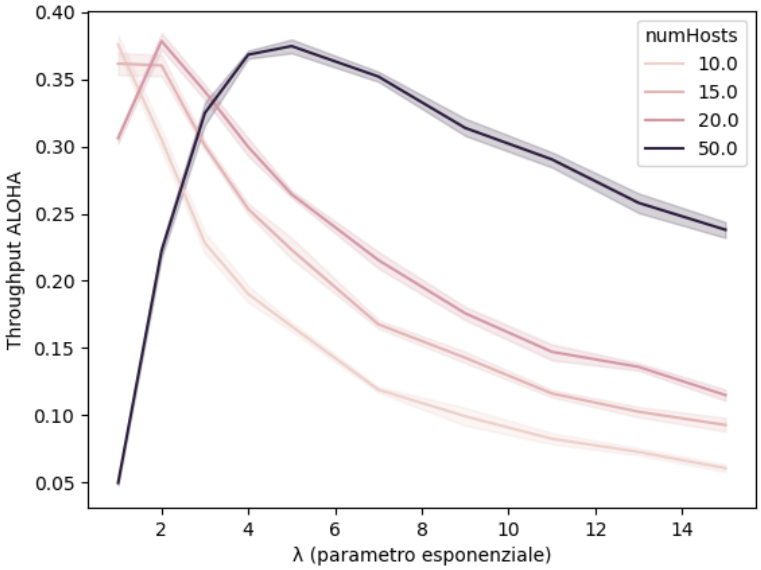

Figure 17: S-ALOHA con intervallo di confidenza dello .95 su un sample size di 10, con un sim-time-limt=3min

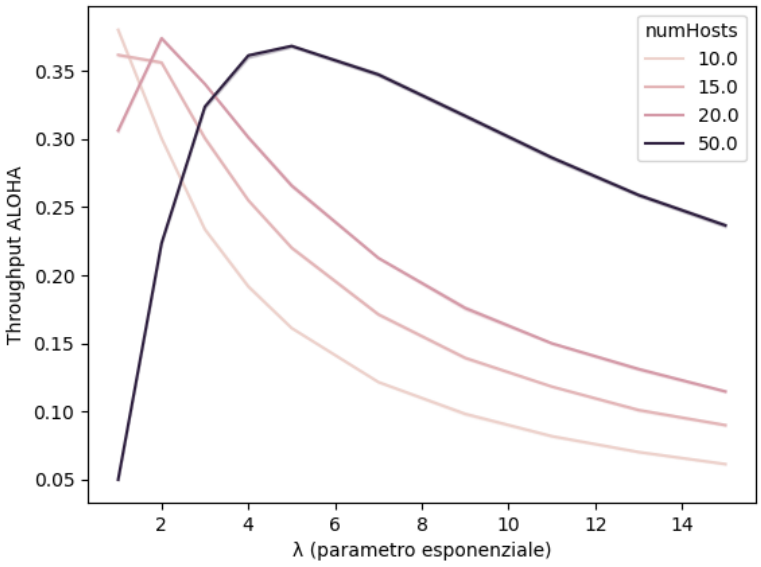

Figure 18: S-ALOHA con intervallo di confidenza dello .95 su un sample size di 10, con un sim-time-limt=90min

7 Conclusioni

Per concludere, abbiamo potuto constatare che i valori calcolati si avvicinano molto ai valori ottenuti dalle nostre simulazioni con il simulatore Omnet++. Inoltre abbiamo visto che il comportamento di Pure ALOHA e S-ALOHA sono simili, in base alle constatazioni fatte sui vari aspetti.

References

[1] Andrea Baiocchi and Fabio Ricciato. Analysis of pure and slotted aloha with multi-packet reception and variable packet size. *IEEE Communications Letters*, 22(7):1482–1485, 2018.

[2] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[3] Koojana Kuladinithi, Raphael Elsner, Leo Kr\"uger, Sebastian Lindner, Christoph Petersen, Daniel Pl\"oger, Zeynep Vatandas, and Andreas Timm-Giel. Teaching modelling and analysis of communication networks using omnet++ simulator. In Anna F\"orster, Asanga Udugama, Antonio Virdis, and Giovanni Nardini, editors, *Proceedings of the 5th International OMNeT++ Community Summit*, volume 56 of *EPiC Series in Computing*, pages 111–123. EasyChair, 2018.

[4] Andrew S. Tanenbaum and David Wetherall. *Computer Networks*. Prentice Hall, Boston, 5 edition, 2011.

[5] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[6] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.