Simulazione e performance delle reti 2016/2017

Esercitazione Finale

Simulazione Protocolo Tcp/IP

---

v. 1.0 - 6 giugno 2017

Gruppo: **RenoRiders**
Antonio Emanuele Cinà 854866
Massimo Pasqualato 776080
Feliks Hibraj 854342

Indice

1 Introduzione **1**
1.1 Scopo del documento 1

2 Glossario **1**

3 Teoria del protocollo **1**
3.1 Il protocollo TCP 1
3.2 Slow Start 3
3.3 Congestion Avoidance 3
3.4 Fast Retransmit e Fast Recovery 4
3.5 TCP Tahoe 5
3.6 TCP Reno 6

4 Simulatore **6**
4.1 Lo scenario 6
4.2 Come eseguire il simulatore 7

5 Gli ambiti analizzati **8**
5.1 Analisi del Warm-Up 8
5.2 Run pilota e terminazione simulazioni 8
5.3 Punto 1 - Analisi del protocollo con condizioni di canale guasto 9
5.4 Punto 2 - Valutare il Througput in funzione della bontà del canale 14
5.5 Punto 3 - Valutare le dimensioni del Buffer per massimizzare il Througput 16

1 Introduzione

La seguente relazione tratta lo sviluppo e l'analisi dei risultati del progetto assegnato al corso di Simulazione e Performance delle Reti. Scopo del progetto è quello di studiare le performance dei procolli TCP seguenti:

- ◇ AIMD
- ◇ RENO
- ◇ TAHOE

Per semplicità, la consegna del progetto, prevede che il controllo di flusso,, trattato anche a livello Transport, non ponga mai richieste più restrittive rispetto al controllo della congestione, di nostro interesse in quanto i 3 protocolli presentano differenze proprio qui..

1.1 Scopo del documento

Lo scopo del documento è quello di fornire uno scenario del modello e delle componenti del simulatore sviluppato, riportare le logiche di funzionamento dei protocolli simulati, riportare ed analizzare dati e risultati finali.

2 Glossario

Termini e abbreviazioni utilizzate nel presente documento:

FEL	struttura dati contenente gli eventi da eseguire all'interno delle simulazioni. Le entry al suo interno sono composte da (time, evento), dove time è il tempo in cui effettuare l'azione specificata dentro evento (trigger).
Segmenti	unità di trasferimento utilizzata a livello Transport
Payload	segmente contenente i dati da inviare al destinatario.
Ack	segmento contenente solo la conferma di ricezione dell'payload associato. Contiene un campo id che identifica l'identificatore del payload confermato.
SS-thresh	valore di soglia impostato per la Slow Start
CW	congestion window (finestra di congestione)

3 Teoria del protocollo

3.1 Il protocollo TCP

Il TCP (Transfer Control Protocol) è un protocollo di livello di trasporto orientato alla connessione che si appoggia su un livello Network, che generalemente non è orientato alla connessione. Il livello transport dell'architettura TCP/IP non contiene solo questo protocollo ma anche UDP (User Datagram Protocol) che contrariamente è un protocollo non orientato alla connessione e che non prevede alcun controllo di congestione, dunque non adatto ai fine di studio del nostro sistema.
Le caratteristiche principali del protocollo TCP sono:

- ◇ Trasferimento affidabile: tutti i segmenti che vengono recapitati al destinatario prevedono una conferma di ricezione (ACK), la cui mancanza porterebbe allo scadere del timeout ad esso associato ed alla sua ritrasmissione.
- ◇ Trasferimento ordinato, trasferimento e ricezione dei segmenti deve avvenire in modo ordinato. A vantaggio di questo il TCP utilizza la tecnica del *Cumulative Ack*, che permette ad un host di confermare tutti i segmenti in attesa il cui id è <= dell'id dell'ack corrente. Questa tecnica è molto utile quando siamo su un canale con qualche perdita oppure con tecniche come *Fast Retransmit* e *Fast Recovery*.
- ◇ Controllo della congestione: Una delle principali cause di degradazione delle prestazioni delle reti è senza dubbio la congestione, che si verifica quando il traffico generato dagli host è maggiore delle capacità di elaborazione della rete stessa, in questo contesto il canale. Infatti, ogni volta che il canale riceve dati più velocemente di quanto esso riesca a smaltirli, è costretto a memorizzarli all'interno di un buffer (coda) per la loro futura elaborazione. Il verificarsi di queste condizioni può portare ad una perdita di prestazioni del sistema in quanto si aumenta il ritardo medio dei segmenti che alla lunga porterebbero per esempio gli host ad andare in timeout. La situazione ancora peggiore si verifica quando una volta che gli host ricevono il timeout rispediscano ulteriormente sulla coda, che contiene ancora il segmento inviato in precedenza non ancora elaborato a causa di un tempo di attesa maggiore del timeout. In figura si nota come il Throughput sul canale aumenti con il crescere del carico fino ad un certo valore; da questo punto in poi tuttavia l'elevata presenza in rete di inutili dati duplicati provoca un brusco crollo delle prestazioni.

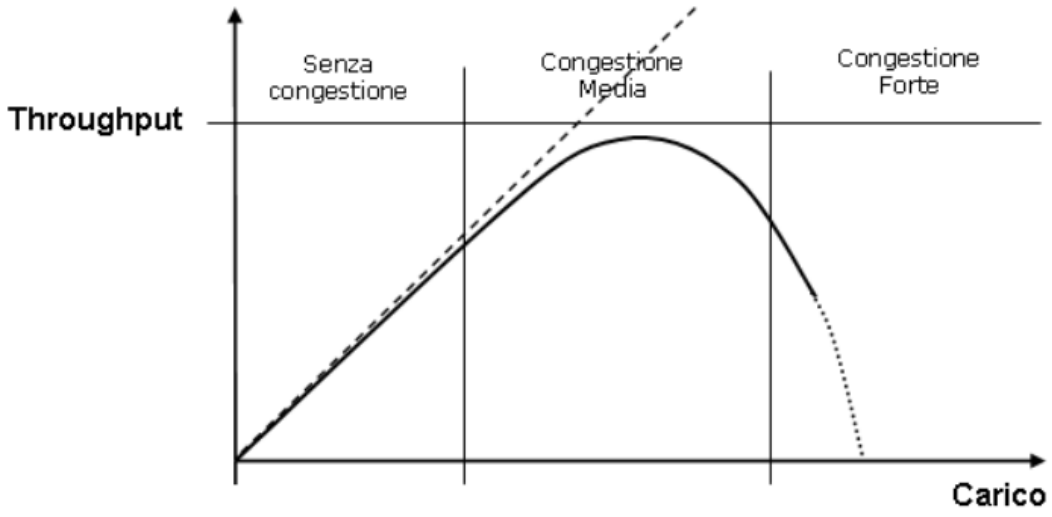

Figura 1: Rapporto tra througput e carico

Quanto al controllo della congestione il TCP effettua dei controlli prevenire il verificarsi della congestione sulla rete tra cui:

- ◇ Calcolo dinamico del timeout, ricalcola i tempi di timeout in base al tempo medio di risposta ottenuto, non lasciando dunque un valore fisso ma lasciandolo variare ed adattare in base alla situazione della rete.

◇ Sliding Window, utilizza un algoritmo a finestra scorrevole, modificandone la grandezza in base allo stato stimato della rete. Se il TCP percepisce che la rete non è in stato di congestione nel percorso tra sé e la destinazione aumenta il suo ritmo di trasmissione, mentre lo riduce quando comincia a notare qualche comportamento anomalo che causa il timeout dei segmenti.

Figura 2: TCP/IP sliding window

3.2 Slow Start

Slow-Start è un algoritmo che viene utilizzato per il calcolo della CW all'avvio della connessione e ad ogni scadenza di un time-out. Esso si basa su due regole principali:

◇ il valore di CW viene impostato ad 1 ad ogni nuova connessione o tutte le volte che si verifica lo scadere di un timeout, che porta alla ritrasmissione del segmento.

◇ la CW viene incrementata esponenzialmente alla conferma (ACK) di tutti i segmenti spediti.

Questo algoritmo si propone di arrivare rapidamente ad ottenere una finestra di trasmissione ampia grazie all'incremento esponenziale della CW, per poter avere un throughput elevato sin dalle prime fasi. Tuttavia esso non può essere utilizzato troppo a lungo altrimenti si raggiungerebbero delle finestre con valori troppo elevati. Perciò si imposta una soglia di CW, detta *Slow-Start Threshold (SS-thresh)*, raggiunta la quale TCP cambia comportamento e passa nella modalità di **Congestion Avoidance**.

3.3 Congestion Avoidance

L'algoritmo di Congestion Avoidance viene utilizzato dopo il raggiungimento di SS-thresh mediante l'utilizzo dello Slow-Start. Infatti per evitare che le CW crescano troppo rapidamente, portando alla congestione del canale, si preferisce incrementare il valore di CW più lentamente (linearmente).

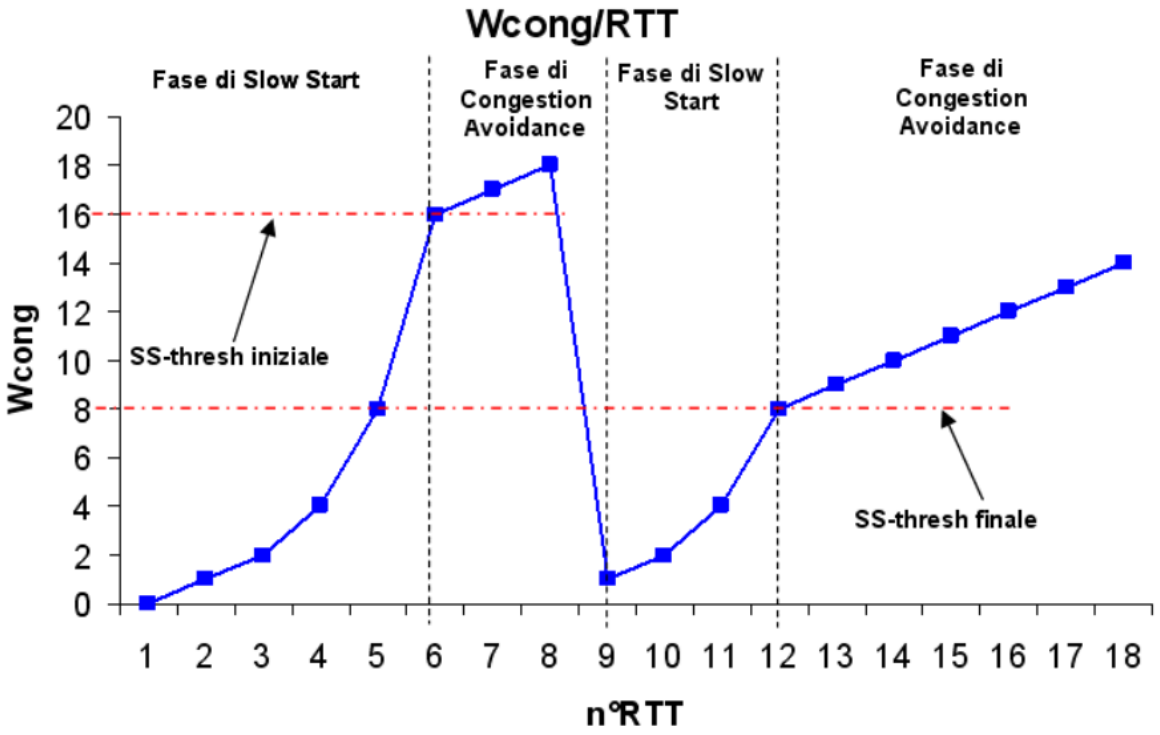

Figura 3: Andamento di CW al variare di Slow Start e Congestion Avoidance

3.4 Fast Retransmit e Fast Recovery

Fino ad ora abbiamo detto che quando un pacchetto viene perso, prima che ne venga effettuata la ritrasmissione deve scadere il timeout corrispondente. Questo comporta però ad un considerevole perdita di tempo che porta ad un peggioramento delle prestazioni. Per ovviare a questo inconveniente è stato allora pensato un algoritmo che, alla ricezione di un certo numero di ACK duplicati, provochi la ritrasmissione immediata del messaggio richiesto senza attendere il time-out. Questo algoritmo viene chiamato Fast Retransmit e si comporta nel seguento modo:

- ◇ Alla ricezione di tre o più ACK duplicati il mittente deduce la rete non è in congestione, sennò non riceverebbe neanche gli ack, ma il segmento non confermato successivo non arriva a causa di una perdità nel canale, probabilmente a causa di interferenze.
- ◇ Il mittente imposta il valore di CW = 1 e ritrasmette immediatamente il pacchetto successivo al valore dei 3 ACK duplicati senza aspettare il timeout, eliminando dunque i tempi di inattività del mittente causati dall'attesa dei timeout. Questa tecnica permette di aumentare al meglio il throughput degli host dando la possibilità di confermare sequenze più lunghe che erano state interrotte dalla perdita di un singolo segmento.

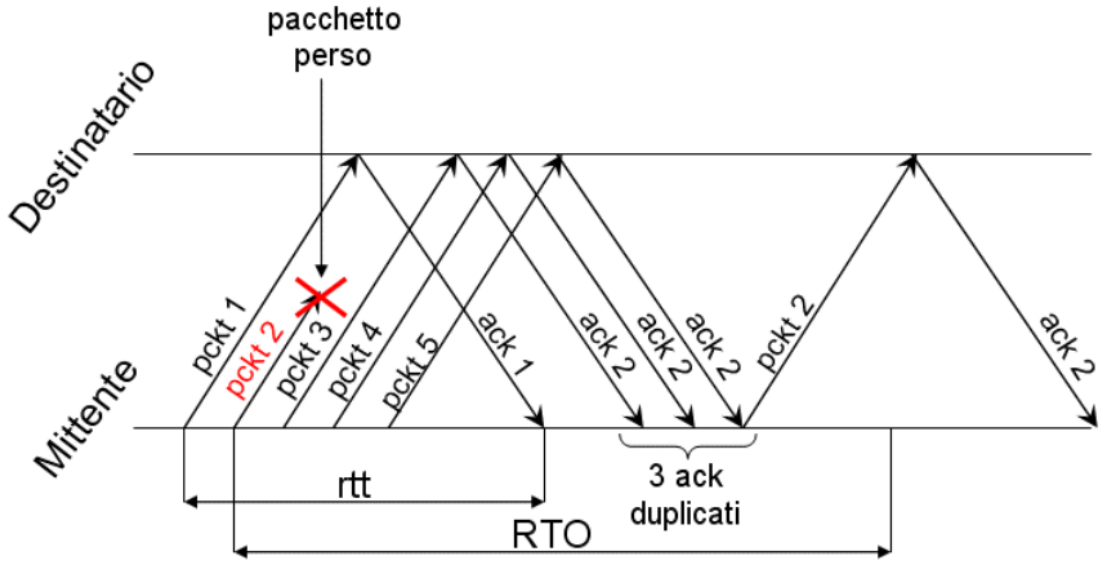

Figura 4: Funzionamento algoritmo di Fast Retransmit

L'algoritmo di Fast Recovery si pone l'obiettivo di eliminare lo svantaggio di dover portare CW = 1 ripartendo dallo stato di congestione avoidance, con il SS-thresh = CW/2, alla ricezione di 3 ack duplicati.
La deduzione forte che viene fatta da entrambi gli algoritmi è che il verificarsi dei 3 ack duplicati non è dovuta alla congestione ma semplicimente all'affidabilità del sistema. La differenza dei due sta dal punto di ripartenza delle corrispettive CW. Fast Recovery viene implementata dal protocollo Tahoe, mentre la Fast Recovery dal protocollo Reno.

3.5 TCP Tahoe

La versione Tahoe di TCP implementata nel nostro simulatore esegue il controllo della congestione implementando i seguenti algoritmi:

- ◇ Slow-Start
- ◇ Congestion Avoidance
- ◇ Fast Retransmit

La finestra di congestione viene incrementata del doppio per ogni ACK corretto ricevuto durante la fase di Slow-Start, ovvero quando CW $<$ SS-thresh, mentre viene aumentata di 1 ad ogni ACK ricevuto nella fase di Congestion Avoidance, ovvero quando è verificata la condizione CW $>$ SS-thresh. TCP si accorge della perdita di un pacchetto dovuta alla congestione della rete quando vede un certo numero di ACK duplicati, che di default è 3, oppure quando scatta un timeout. In entrambi i casi esso reagisce impostando la SS-thresh alla metà del valore della finestra di trasmissione corrente; inoltre viene riportato il valore della CW al suo valore iniziale, che normalmente fa rientrare l'algoritmo in fase di Slow-Start. Nel caso dei 3 ack duplicati applica l'algoritmo di Fast Retransmit.

3.6 TCP Reno

Nella versione Reno gli algoritmi utilizzi sono i seguenti:

- ◇ Slow-Start
- ◇ Congestion Avoidance
- ◇ Fast Recovery

La differenza tra Reno e Tahoe è che la Reno ritorna al CW = 1 ed effettua lo slow start solo in presenza di timeout. Alla ricezione di 3 ck duplicati setta il SS-thresh = CW/2 e ritorna allo stato di congestion avoidance.

4 Simulatore

4.1 Lo scenario

Lo scenario simulato dalla nostra applicazione è quello di studiare le performance di un sistema cosi composto da più host greedy, sempre pronti a trasmettere, che condividono lo stesso path sul canale di rete. I segmenti inviati dagli host vengono inseriti in un buffer, di dimensione parametrizzata T, ed inviati al destinatario oppure droppati se non c'è più spazio nel buffer. Tutti gli host utilizzeranno lo stesso protocollo scelto all'inizio dello start della simulazione, ognuno però avente quantitativo di segmenti da inviare diversi campionati da una distribuzione geometrica di valor medio G, parametro di simulazione. Le simulazioni da noi generate sono simulazioni ad eventi discreti, quindi il tempo non incrementa ad intervalli fissi ma avanza prendendo come tempo attuale il tempo fornito dal prossimo evento in FEL (Future Event List), struttura dati che contiene entry formate (tempo, evento).

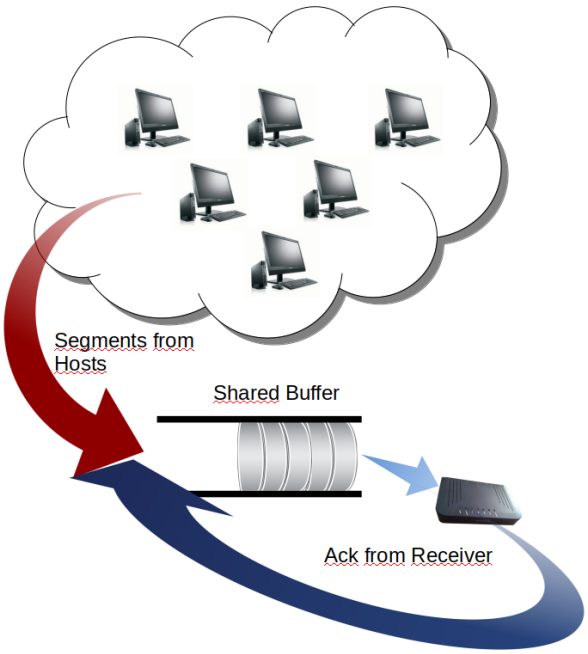

Figura 5: Design del modello

4.2 Come eseguire il simulatore

Per avviare il simulatore è necessario installare JAVA si deve aver installato nel calcolatore il JAVA 1.8 ed eseguire da riga commando "java -jar Simtipee.jar", oppure cliccando sulla sua icona da interfaccia grafica. Il simulatore prevede, inoltre, la possibilità passare attraverso passaggio di parametri i parametri della realtà da simulare, funzionalità ottima che ci ha permessi di scrivere uno script python per l'esecuzione automatica delle simulazioni dando in input un file csv contenente tutti le simulazioni d'interesse. I parametri che utilizza sono:

- ◇ Protocollo, il tipo di protocollo che si intende simulare, le possibilità sono tre, AIMD, RENO, TAHOE. Ognuno è rappresentato da una classe che ne segue il comportamento sulla base delle specifiche reali e semplificate dalla consegna.
- ◇ K, numero di host che contendono il canale
- ◇ T, grandezza del buffer
- ◇ P, probabilità che un segmento sia valido, 1-p invece che sia corrotto. Parametro utile per simulare situazioni di canale con poca affidabilità, disturbati da rumore o altre interferenze.
- ◇ G, parametro della geometrica utilizzata dagli host per campionare il numero di segmenti da inviare. Mentre 1/G rappresenta invece la sua media.
- ◇ S, numero si simulazioni da lanciare in parallelo per simulare la realtà scelta.
- ◇ W, peso del WarmUp da dare alla simulazione

5 Gli ambiti analizzati

Le simulazioni da noi effettuate sono state fatte per rispondere a 3 quesiti proposti dalla consegna:

- ◇ Valutare il throughput con un canale la cui probabilità di successo è pari a 0.01.
- ◇ Valutare il throughput in funzione di p.
- ◇ Valutare il valore massimo di ottimizzazione della grandezza del buffer.

5.1 Analisi del Warm-Up

Il modello ci porta alla realizzazione di simulazioni di tipo non-terminating asintoticamente stazionarie, per cui solamente quando $t \to \infty$ il sistema raggiungerà lo stato di stazionarietà. Uno dei problemi però di cui soffrono questo tipo di simulazioni è l'eliminazione del warm-up, ovvero l'influenza dello stato iniziale della nostra simulazione che influenza i tempi di raggiungimento dello stato stazionario. La sua analisi è stato affrontata utilizzando il metodo grafico di Welch visto a lezione, mediante uno script R realizzato da noi, per trovare il valore l di campioni dopo il quale iniziare a raccogliere le statistiche.

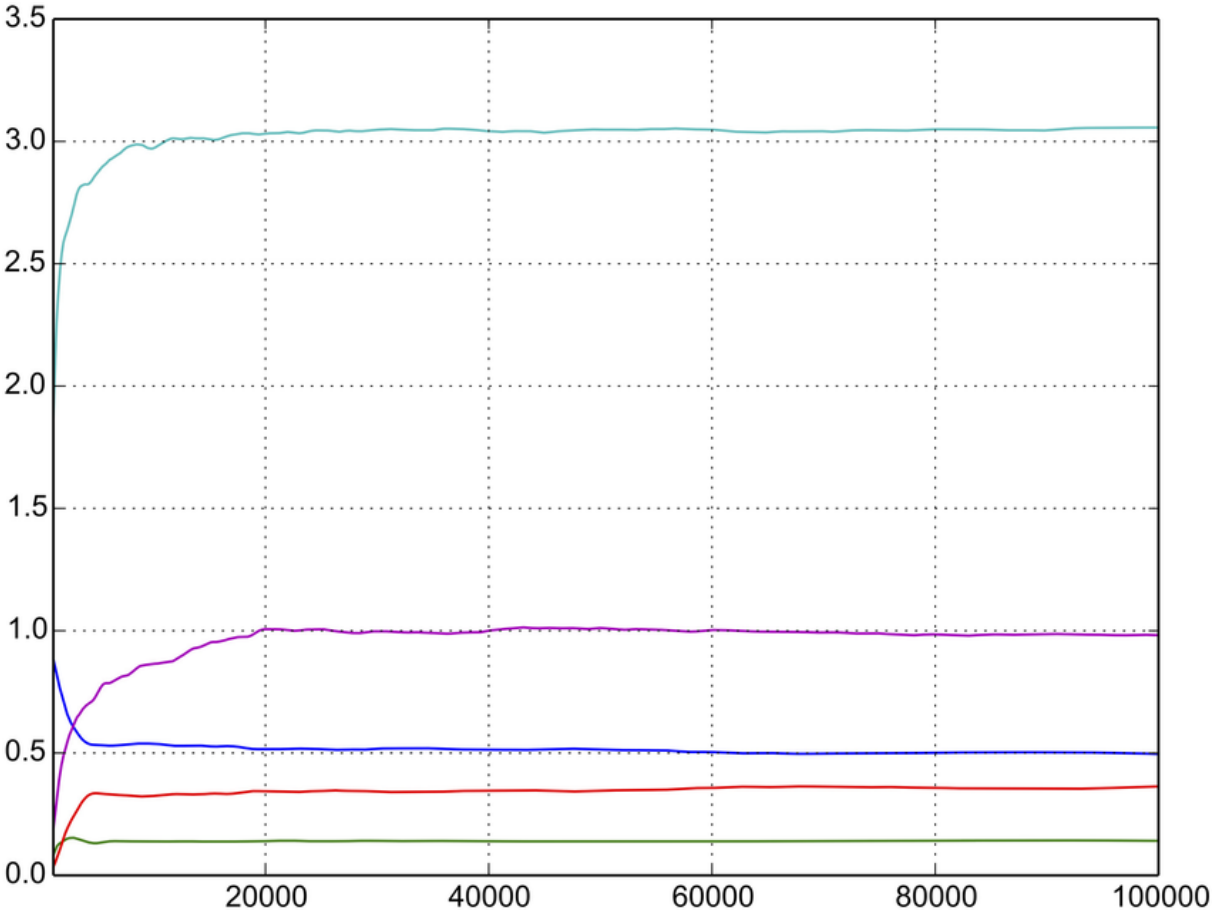

Figura 6: Analisi grafica con metodo di Welch

5.2 Run pilota e terminazione simulazioni

Al fine di migliorare la stima dei risultati delle nostre simulazioni abbiamo deciso di adottare il metodo dei run pilota. Infatti l'applicazione centrale lancerà M simulazioni (parametro in input), ognuna associata ad un thread, a cui successivamente ogni tot di tempo andrà a chiedere le statistiche fino a quel momento. Il metodo run pilota costruisce una matrice NxM, dove N è il numero di campionamenti effettuati, ed ogni istante la i-esima colonna rappresenta la distribuzione della i-simulazione fino a quel momento. La stima finale e fornita in output è calcolata come la media delle medie Xn, aumentandone dunque l'accuratezza della stima ed evitare di concentrarsi su simulazioni troppo estreme. La terminazione delle simulazioni avviene quando l'intervallo di confidenza costruito sulla media Xn ha un errore relativo inferiore al 5% , ad eccezione delle simulazioni del punto 1 per cui è stato aumentato l'errore relativo al 30% per ridurre i tempi di esecuzione (circa 30-40 minuti).

5.3 Punto 1 - Analisi del protocollo con condizioni di canale guasto

Nel seguente punto si vuole studiare le performance del modello in una situazione in cui il canale abbia una probabilità di successo pari a 1/100, ovvero solamente un segmento su 100 risulta essere integro. L'indice d'interesse è il throughput degli host, ovvero quanti segmenti riescono ad inviare in un intervallo di tempo. Sappiamo che per il protocollo tcp un segmento è stato inviato con successo solo quando riceve il suo corrispettivo acknowledge, questo porta ad un ulteriore perdita di performance in quanto sapendo che il canale ha una probabilità di integrità pari al

1/100, noi chiediamo che anche il suo corrispettivo ack arrivi integro e da ciò ne segue che:

$$P(segmento) = P(payload) * P(acknowledge) = 0.01 * 0.01 = 0.0001$$

Da ciò deriva che sarà ancora più difficile riuscire ad ottenere un success di invio del segmento in quanto siamo in presenza di un canale con poca affidabilità. Premesso questo abbiamo simulato il comportamento del sistema con più valori per studiarne il comportamento. Abbiamo potuto notare, sia in questi studi che in altri, con parametro p basso si cerca di mantenere un valore di G più alto cosi da permettere ai timeout di non crescere troppo. Infatti il problema principale è che i protocolli TCP AIMD, RENO e TAHOE concordano tutti e 3 sul fatto che al verificarsi di un timeout ci si sta avvicinando ad una situazione di congestione della rete. In questo caso però gli host riceveranno molti timeout, che non saranno dovuti alla congestione del canale ma bensi alla sua scarsa affidabilità, portando a lunghi tempi di attesa quando il segmento non è più comunque valido. I parametri di studio sono stati riportati in una tabella excel che li riassume:

Host K	Buffer T	P-value	G-value	X Tahoe	RST Tahoe	X Reno	RST reno	X AIMD	RST AIMD
5	5	0.01	0.1	6.788353e-06	10973596	1.824585e-05	10805404	1.105714e-05	11038108
5	5	0.01	0.5	2.713182e-05	2489043	1.243411e-05	2735924	9.386675e-06	2784173
5	25	0.01	0.1	6.122994e-06	10502890	8.054324e-06	12332300	6.685633e-06	11015703
5	25	0.01	0.5	2.13681e-05	2339310	1.020074e-05	3090878	2.532965e-05	2501611
5	50	0.01	0.1	4.9149e-06	11531413	5.029116e-06	11236081	1.848867e-05	11951751
5	50	0.01	0.5	1.888201e-05	2744010	7.973944e-06	2899595	2.624236e-05	2771226
50	5	0.01	0.1	1.797778e-05	7803143	2.030606e-05	8081351	6.659558e-05	693805.2
50	5	0.01	0.5	5.952639e-05	1581998	7.382465e-05	2251582	6.377074e-05	341316.5
50	25	0.01	0.1	5.183485e-06	9169196	9.183378e-06	5573625	3.404006e-05	2269467
50	25	0.01	0.5	3.127181e-05	3263321	3.686229e-05	1663209	5.562409e-05	1378489
50	50	0.01	0.1	7.276757e-06	8503160	9.489418e-06	6071255	1.089673e-05	3395116
50	50	0.01	0.5	1.337832e-05	4065431	2.826686e-05	1546691	2.956411e-05	1156200

Figura 7: Valori di simulazione Punto1

Di seguito riportiamo i grafici che rappresentano il throughput degli host con i diversi parametri di simulazione:

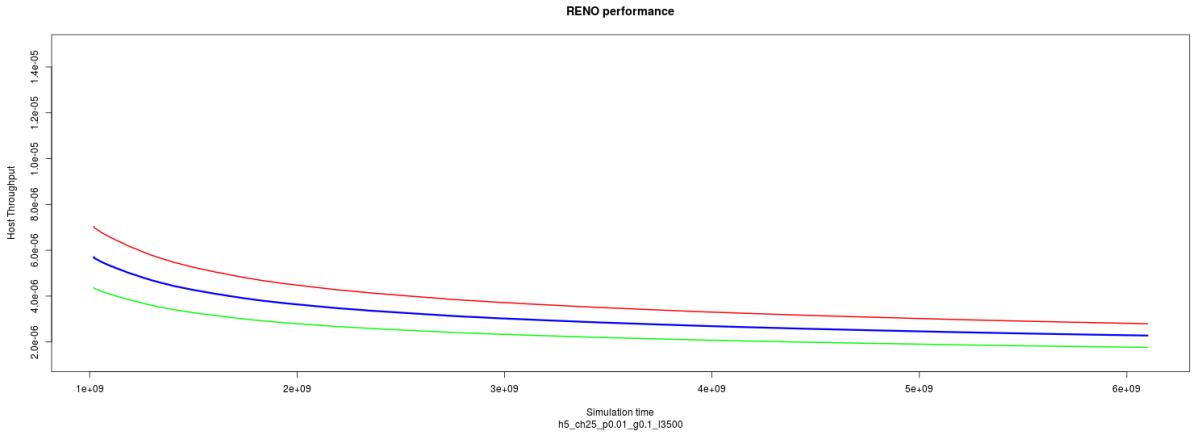

Figura 8: Host Throughput RENO K5

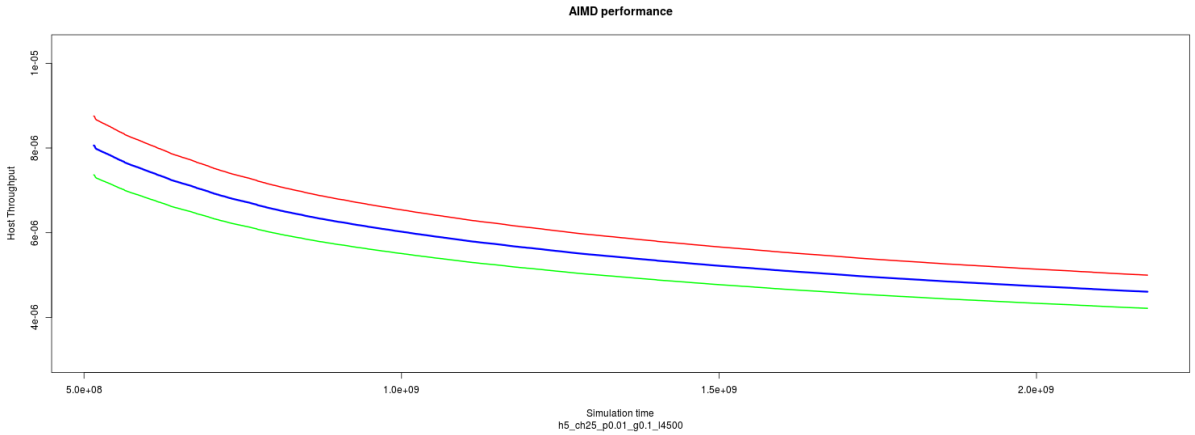

Figura 9: Host Throughput TAHOE K5

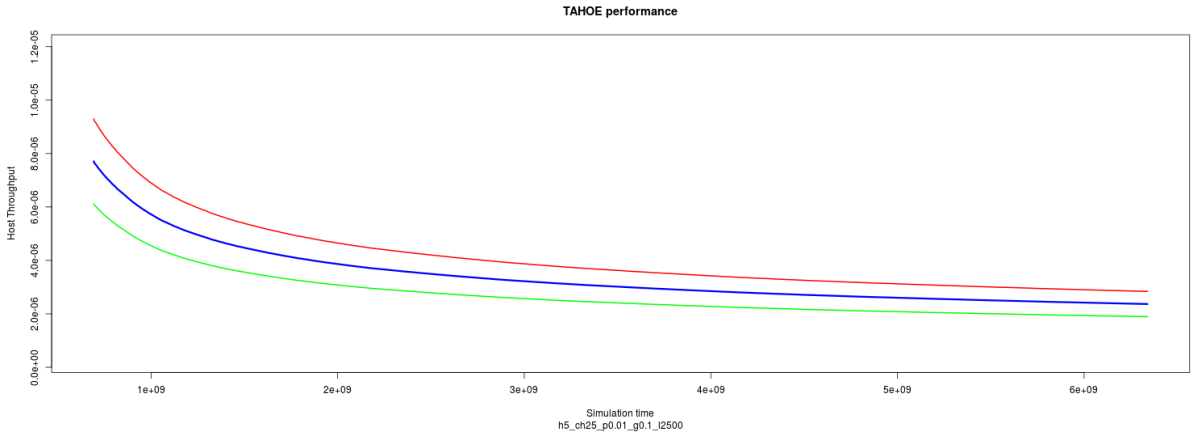

Figura 10: Host Throughput AIMD K5

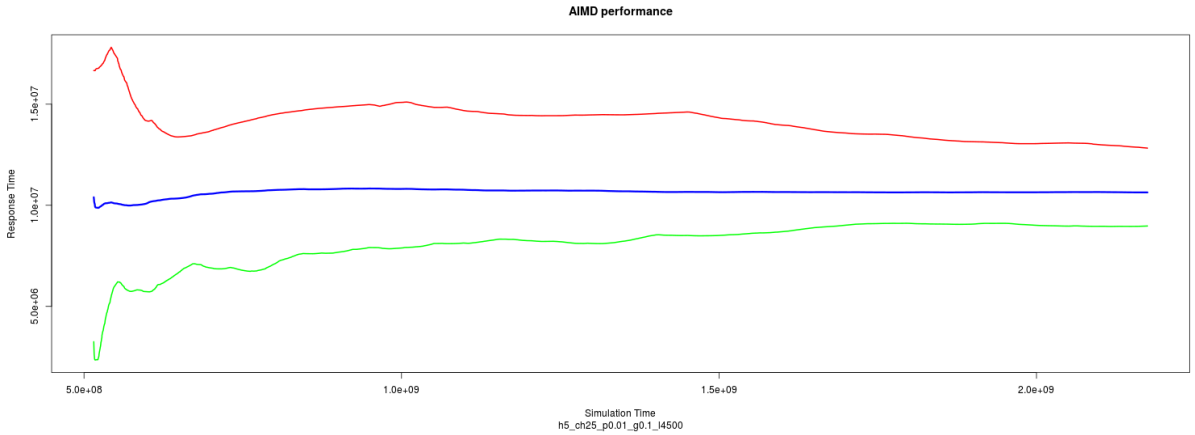

Figura 11: Response Time AIMD K5

Anche il response time ne risente della scarsa affidabilità del canale, infatti come si può notare i tempi di risposta sono veramente altissimi dunque dal primo invio di un segmento al ritorno del suo corrispettivo ack passerà molto tempo con conseguente abbattimento del throughput.

I 3 protocolli non presentano valori tanto discostanti l'uno dall'altro da poter affermare la superiorità di uno rispetto all'altro in quanto con tasso cosi elevato di errori entrambi mantengono congestion window pari a 1 e non c'è utilizzo neanche di Fast Retransmit o Fast Recovery. Dall'analisi delle simulazioni si è pottuto notare che in tutti i casi comunque con il canale in condizioni pessime, il througput è tendente a zero. L'unica miglioria possibile, notando un piccolo incremento del througput, si ha solo variando il valore di g, riducendo il numero di segmenti da inviare per connessione, questo è possibile in quanto consideriamo che l'apertura e la chiusura della connessione non comportino dei costi di tempo, anche se nella realtà il protocollo lo prevede.

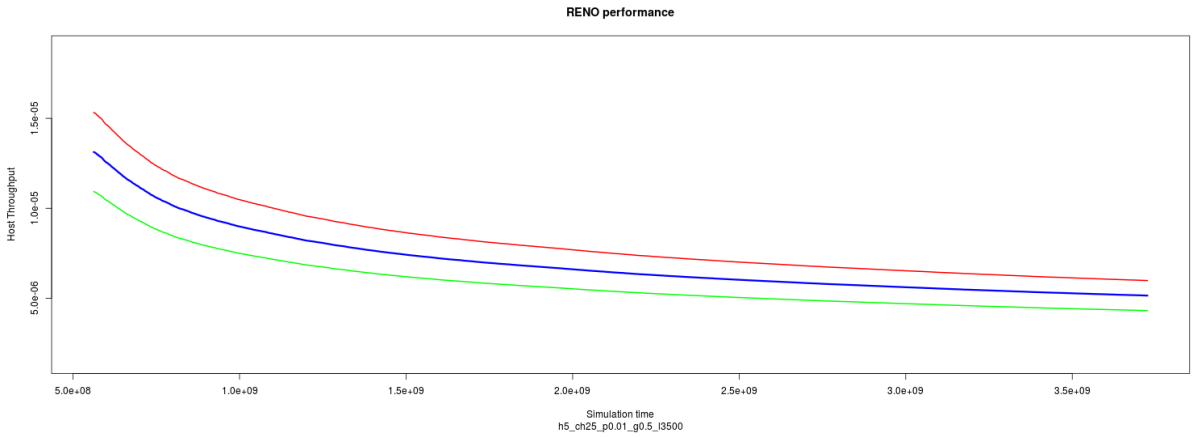

Figura 12: Host Throughput RENO K5

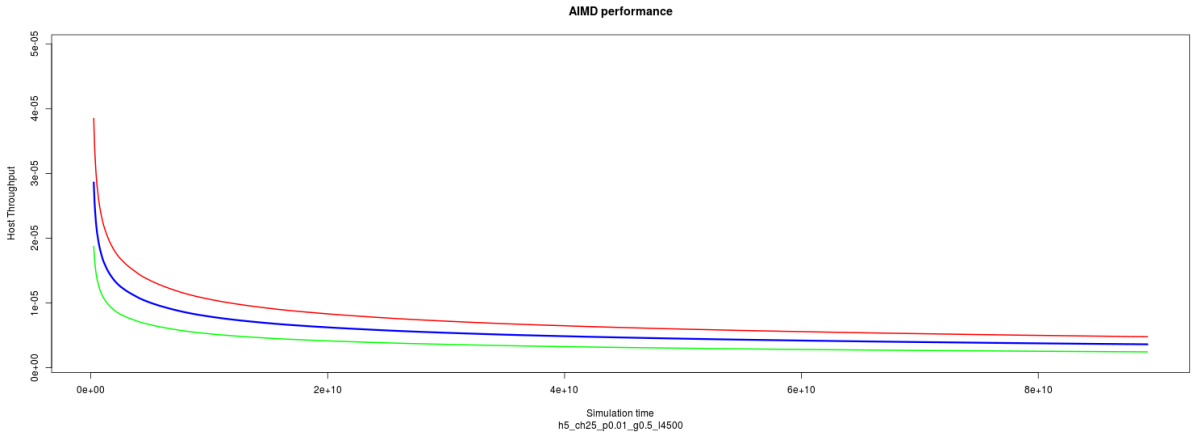

Figura 13: Host Throughput TAHOE K5

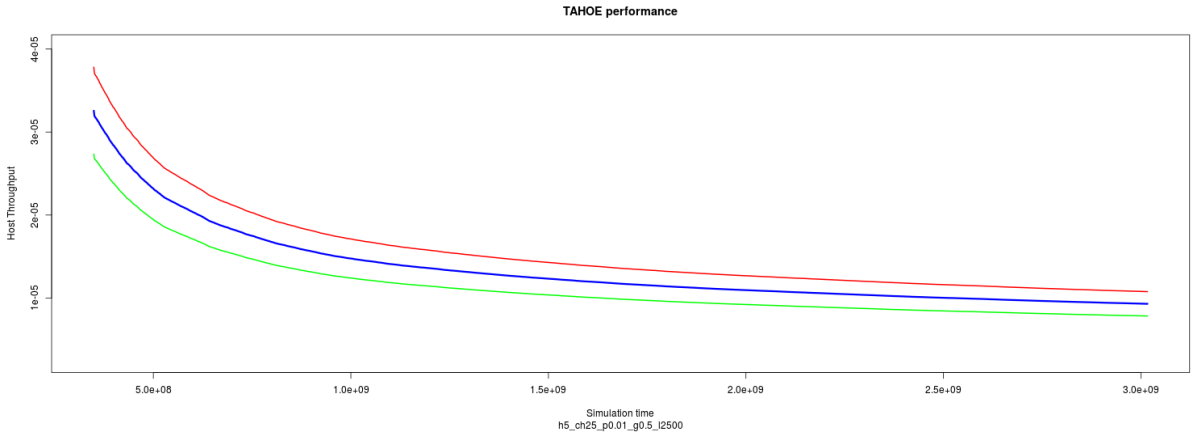

Figura 14: Host Throughput AIMD K5

Anche il response time ne ha risentito dell'aumento di G, infatti rispetto al caso precedente sono diminuiti ma non abbastanza da migliorare la situazione della rete.

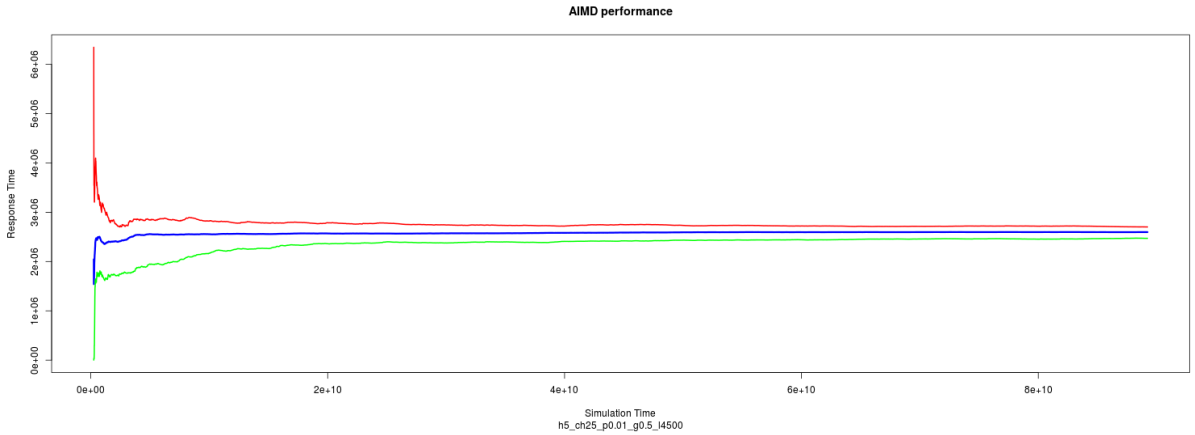

Figura 15: Response Time AIMD H5

Un esperimento che abbiamo voluto provare è stato quello di studiare il comportamento di una rete con scarsa affidabilità, scarso buffer e con molti host che condividono il canale. L'idea iniziale è che comunque nonostante la scarsa affidabilità ci si aspettava molti drop da parte del canale, ma alla fine dello studio abbiamo scoperto che in realtà il numero di drop non era cosi eccessivo.

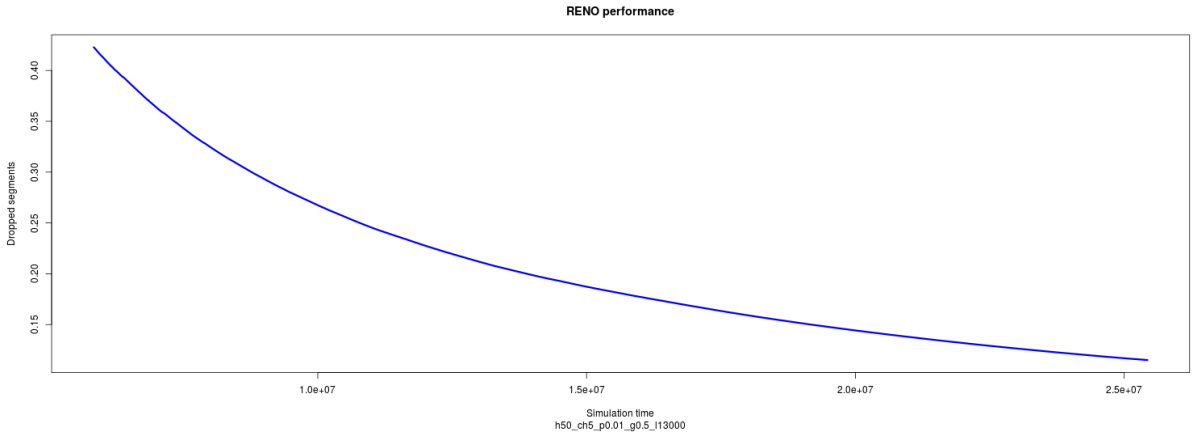

Figura 16: Dropping H50 T5

Anche throughput e response time sembrano aver avuto una miglioria, questo perchè solamente al servizio del segmento si decide se il segmento è integro o meno. Dunque mantenere un buffer piccolo permette di dare minor tempo di attesa ai segmenti e di sfoltire prima la coda da segmenti che verranno scelti come corrotti.

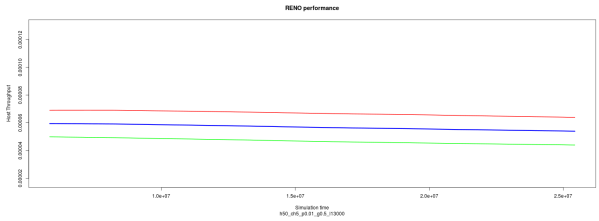

Figura 17: Throughput K50 T5

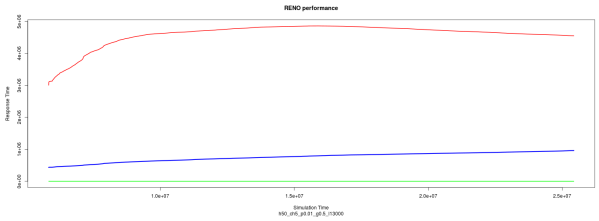

Figura 18: Response Time K50 T5

Studio extra Come caso di studio di nostro interesse abbiamo voluto anche analizzare la situazione in cui il timeout venga impostato ad un valore fisso, cercando un valore ottimale non troppo alto da portare a lunghe attese e non troppo baso da portare a continui invii sul canale. I risultati sembrano essere migliori perchè gli host non aspettavano troppo tempo a "vuoto" se il segmento non era più valido.

5.4 Punto 2 - Valutare il Througput in funzione della bontà del canale

Nel seguente punto si vuole studiare il throughput in funzione della bontà del canale, rappresentato dal valore p. Intuitivamente parlando bassi valori di p ci porteranno ad uno scarso throughput come nel caso precedente. Per analizzare questo punto abbiamo voluto simulare diverse situazioni, sempre messe in una tabella excel, di cui ne riporteremo alcuni riassunti. In ogni simulazione il valore del buffer, anch'esso importante, è stato posto ad un valore scelto alto a priori per evitare il dropping dei segmenti e valutare solamente timeout dovuti all'inaffidabilità del canale.

				TAHOE			RENO			AIMD		
HOST	T	P	G	Throughput	Throughput Host	ResponseTime	Throughput	Throughput Host	ResponseTime	Throughput	Throughput Host	ResponseTime
5	1000	1	0.05	39.03049	7.80386	0.2807718	39.04681	7.808409	0.2806459	41.22828	8.25175	0.2718363
5	1000	0.9	0.05	41.89157	8.375337	0.2442356	41.35192	8.265688	0.2510967	41.74878	8.346097	0.2513637
5	1000	0.8	0.05	18.42677	3.589049	8.9555	19.33007	3.816598	4.841852	17.98076	3.561385	4.941556
5	1000	0.7	0.05	0.3148529	0.06778786	170.1522	0.3294935	0.07502892	168.7552	0.3256481	0.06999502	172.2587
5	1000	0.5	0.05	0.005481945	0.001077102	1289.946	0.005487	0.001087875	1277.846	0.005464859	0.002270363	1261.578
5	1000	0.3	0.05	0.0007869388	0.0001560663	7566.426	0.00078722	0.0001559774	7563.313	0.0007825368	0.0002573791	7445.878
5	1000	0.1	0.05	0.000060033	0.00001197386	114717.2	0.00006024	0.00001194285	114232.7	5.983108e-05	1.383099e-05	112288.8

Figura 19: Valori simulati

				TAHOE			RENO			AIMD		
25	1000	1	0.2	38.08729	1.522407	0.7942255	38.26997	1.528666	0.7937811	43.36700	1.71526	0.7473516
25	1000	0.9	0.2	44.14728	1.764428	0.7325079	43.74867	1.748457	0.7396571	44.03587	1.759533	0.7208513
25	1000	0.8	0.2	37.30710	1.490285	0.9867093	37.21147	1.486988	0.9950442	37.12063	1.484012	0.9876311
25	1000	0.7	0.2	16.44505	0.6366355	12.13772	16.26224	0.6350007	11.63058	16.05441	0.6379895	13.65122
25	1000	0.5	0.2	0.2513280	0.009822888	657.49.00	0.2452248	0.01259505	675.27.00	0.2385030	0.01176497	611.8434
25	1000	0.3	0.2	0.01098399	0.0004320951	3807.56.00	0.01069862	0.0007681139	3840.56.00	0.01097584	0.0007162842	3838.417
25	1000	0.1	0.2	0.0004693788	0.00002551165	62521.2	0.0004656532	0.00008396364	122288.4	0.0004713746	0.000767464	61557.32

Figura 20: Valori simulati

Di seguito seguiranno i grafici e i risultati da loro forniti ed analizzati.

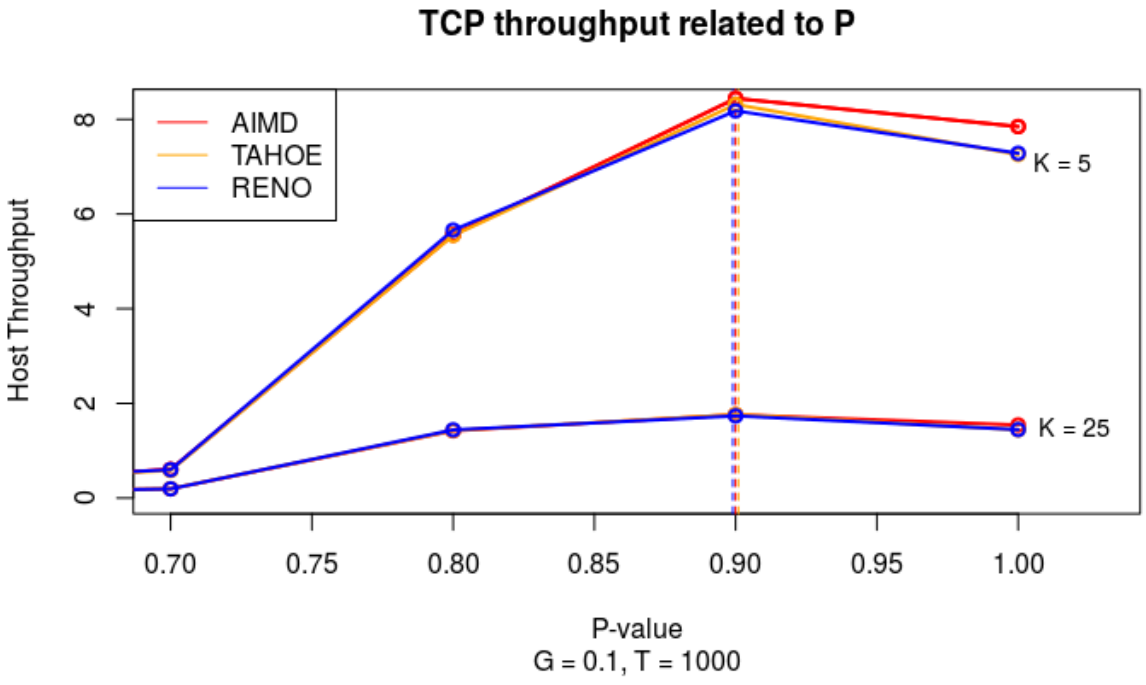

Figura 21: Throughput in funzione del numero di K e P

Da un analisi apportata da questi grafici non sembra esserci molta evidenza tra i protocolli Reno e Tahoe, mentre l'AIMD con valori di p più alti mantiene un throughput leggermente più

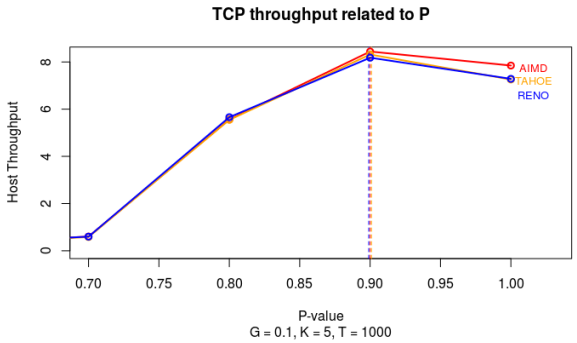

Figura 22: Throughput in funzione di P e K5

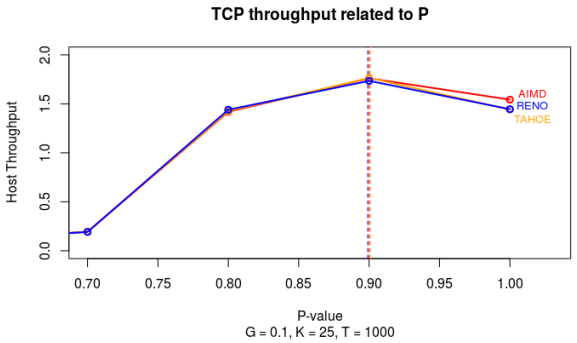

Figura 23: Throughput in funzione di P e K25

alto rispetto agli altri due. Di seguito verranno mostrati altri grafici analizzando solamente il protocollo Tahoe, in quanto per gli altri i risultati non si discostano di molto. Caratteristiche in comune tra i 3 protocolli:

- ◇ All'aumentare del numero di host, il throughput diminuisce.
- ◇ Quando l'affidabilità del canale è bassa (p <= 0.5) l'impatto del numero di host sulla rete è quasi irrevalente. Infatti come si può vedere graficamente il problema non starà sul numero di host ma proprio sulla scarsa affidabilità del canale.

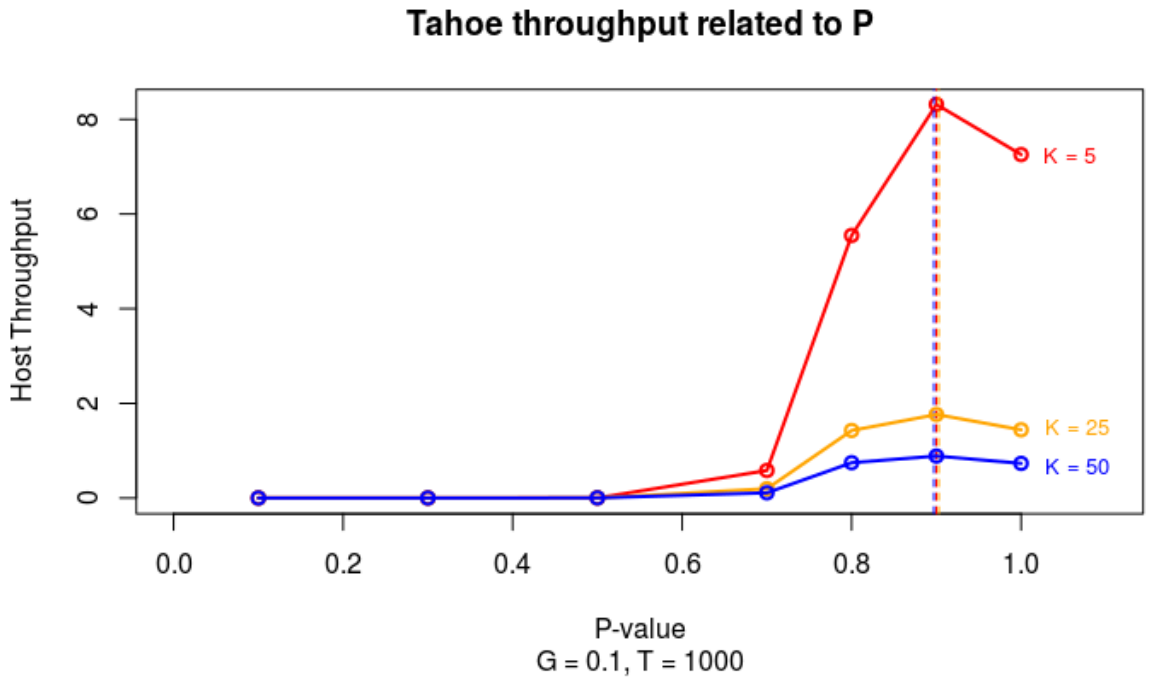

Figura 24: Throughput in funzione del numero di P e K

- ◇ Contrariamente all'idea iniziale non sempre p perfetto (p = 1) aiuta il throughput del canale, infatti dai risultati possiamo vedere come per tutti e 3 il throughput massimo l'abbiamo con p = 0.95. Questo avviene perchè con quel 5% di errore permette di mandare in timeout gli host qualche volta al fine da ridurgli la congestion window e diminuire di conseguenza il numero medio di segmenti in coda ed il loro tempo di attesa.

Altro studio di nostro interesse è stato quello di valutare il throghput anche in funzione del parametro G. Come si può notare, in concordanza a quanto detto al Punto1, un valore alto di G permette agli host di ridurre i loro timeout, perchè le spedizioni finiscono prima, per riaprire una nuova connessione (per cui non abbiamo costi in termini temporali). Quando invece il canale aumenta di affidabilità non vi è grande differenza tra i valori di G.

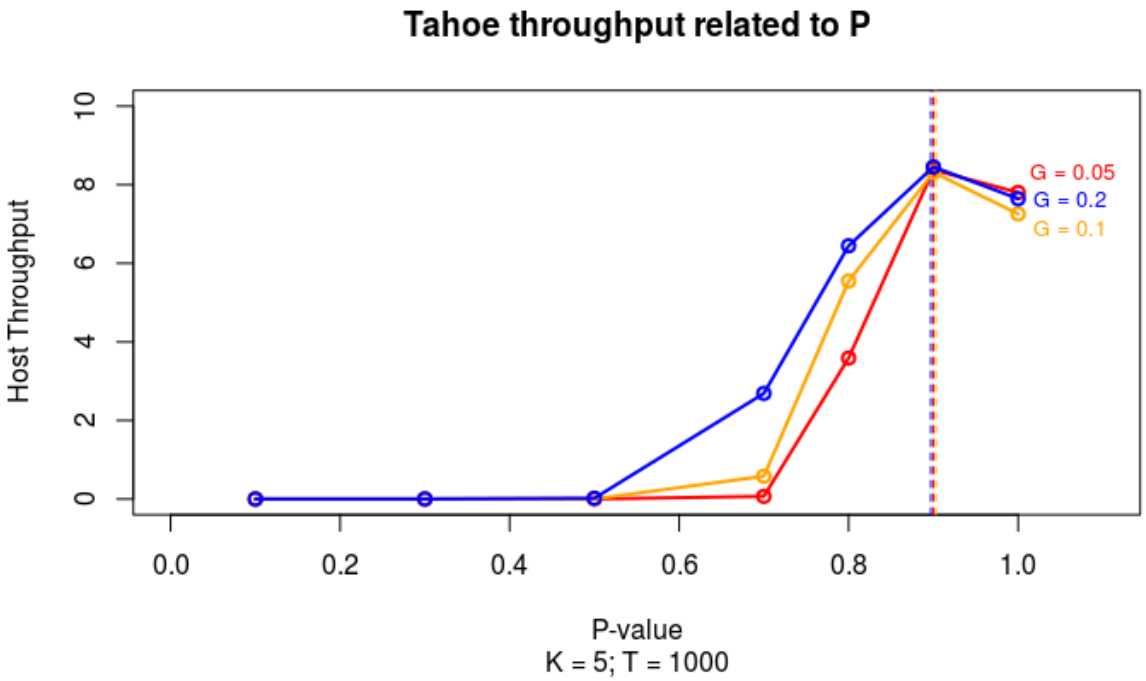

Figura 25: Throughput in funzione del numero di P e G

5.5 Punto 3 - Valutare le dimensioni del Buffer per massimizzare il Throughput

Nel seguente punto lo scopo è studiare il comportamento del throughput degli host ed il response time medio in base alla dimensione del buffer. Esso infatti è un valore molto importante in quanto:

- ◇ T piccolo, porterebbe il canale a saturarsi subito causando dropping dei segmenti.
- ◇ T grande, gli host trovano spazio per poter inviare segmenti sul canale. Questo porterebbe però a lunghi tempi di attesa per i segmenti.

Le simulazioni effettuate sono state riportate anche in questo caso in una tabella excel per poterne infine studiare il comportamento al variare dei diversi parametri, di seguito vengono riportate alcune tabelle per dare un esempio dei valori raccolti.

				TAHOE			RENO			AIMD		
HOST	T	P	G	Throughput	Throughput Host	ResponseTime	Throughput	Throughput Host	ResponseTime	Throughput	Throughput Host	ResponseTime
5	1000	1	0.05	39.03049	7.80386	0.2807718	39.04681	7.808409	0.2806459	41.22828	8.25175	0.2718363
5	1000	0.9	0.05	41.89157	8.375337	0.2442356	41.35192	8.265688	0.2510967	41.74878	8.346097	0.2513637
5	1000	0.8	0.05	18.42677	3.589049	8.9555	19.33007	3.816598	4.841852	17.98076	3.561385	4.941556
5	1000	0.7	0.05	0.3148529	0.06778786	170.1522	0.3294935	0.07502892	168.7552	0.3256481	0.06999502	172.2587
5	1000	0.5	0.05	0.005481945	0.001077102	1289.946	0.005487	0.001087875	1277.846	0.005464859	0.002270363	1261.578
5	1000	0.3	0.05	0.0007869388	0.0001560663	7566.426	0.00078722	0.0001559774	7563.313	0.0007825368	0.0002573791	7445.878
5	1000	0.1	0.05	0.000060033	0.00001197386	114717.2	0.00006024	0.00001194285	114232.7	5.983108e-05	1.383099e-05	112288.8

Figura 26: Valori simulati K5

HOST	T	P	G	TAHOE			RENO			AIMD		
25	1000	1	0.2	38.08729	1.522407	0.7942255	38.26997	1.528666	0.7937811	43.36700	1.71526	0.7473516
25	1000	0.9	0.2	44.14728	1.764428	0.7325079	43.74867	1.748457	0.7396571	44.03587	1.759533	0.7208513
25	1000	0.8	0.2	37.30710	1.490285	0.9867093	37.21147	1.486988	0.9950442	37.12063	1.484012	0.9876311
25	1000	0.7	0.2	16.44505	0.6366355	12.13772	16.26224	0.6350007	11.63058	16.05441	0.6379895	13.65122
25	1000	0.5	0.2	0.2513280	0.009822888	657.49.00	0.2452248	0.01259505	675.27.00	0.2385030	0.01176497	611.8434
25	1000	0.3	0.2	0.01098399	0.0004320951	3807.56.00	0.01069862	0.0007681139	3840.56.00	0.01097584	0.0007162842	3838.417
25	1000	0.1	0.2	0.0004693788	0.00002551165	62521.2	0.0004656532	0.00008396364	122288.4	0.0004713746	0.000767464	61557.32

Figura 27: Valori simulati K25

Di seguti vengono riportati i grafici riassuntivi che mostrano l'andamento del throughput al variare dei parametri T,p,g,h:

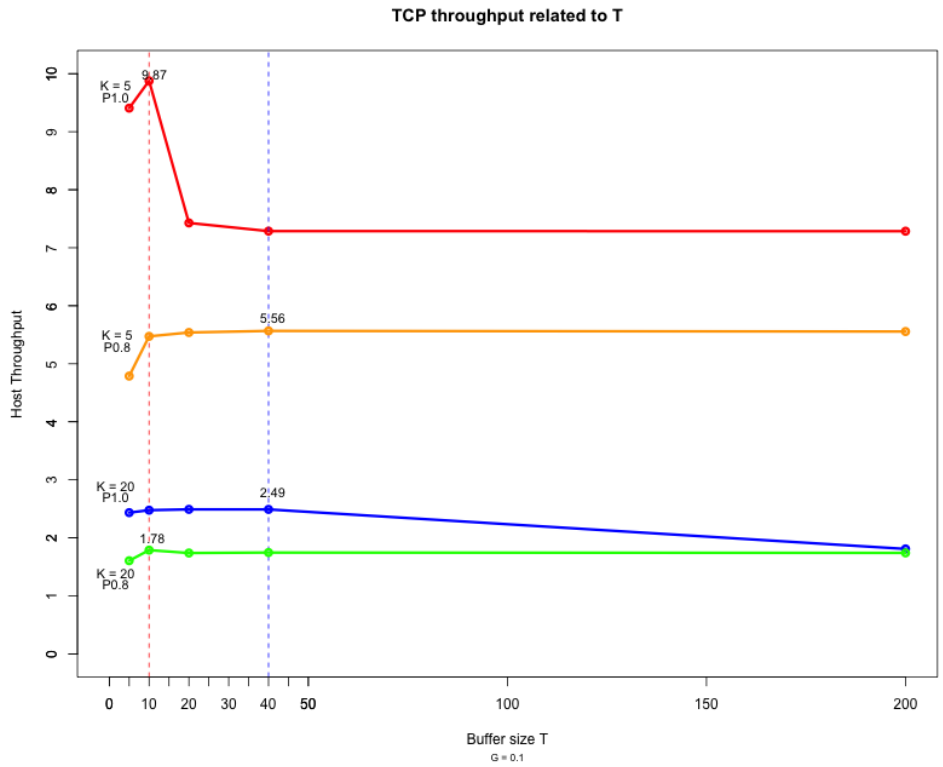

Figura 28: Throughput in relazione al numero di host e P

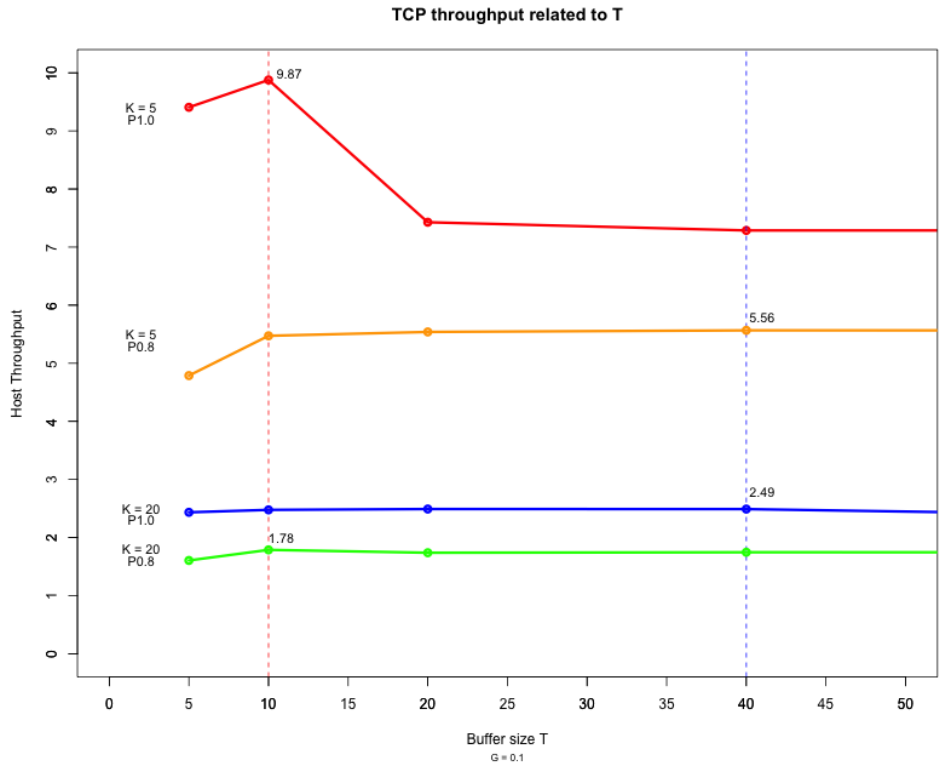

Figura 29: Throughput in relazione al numero di host e P

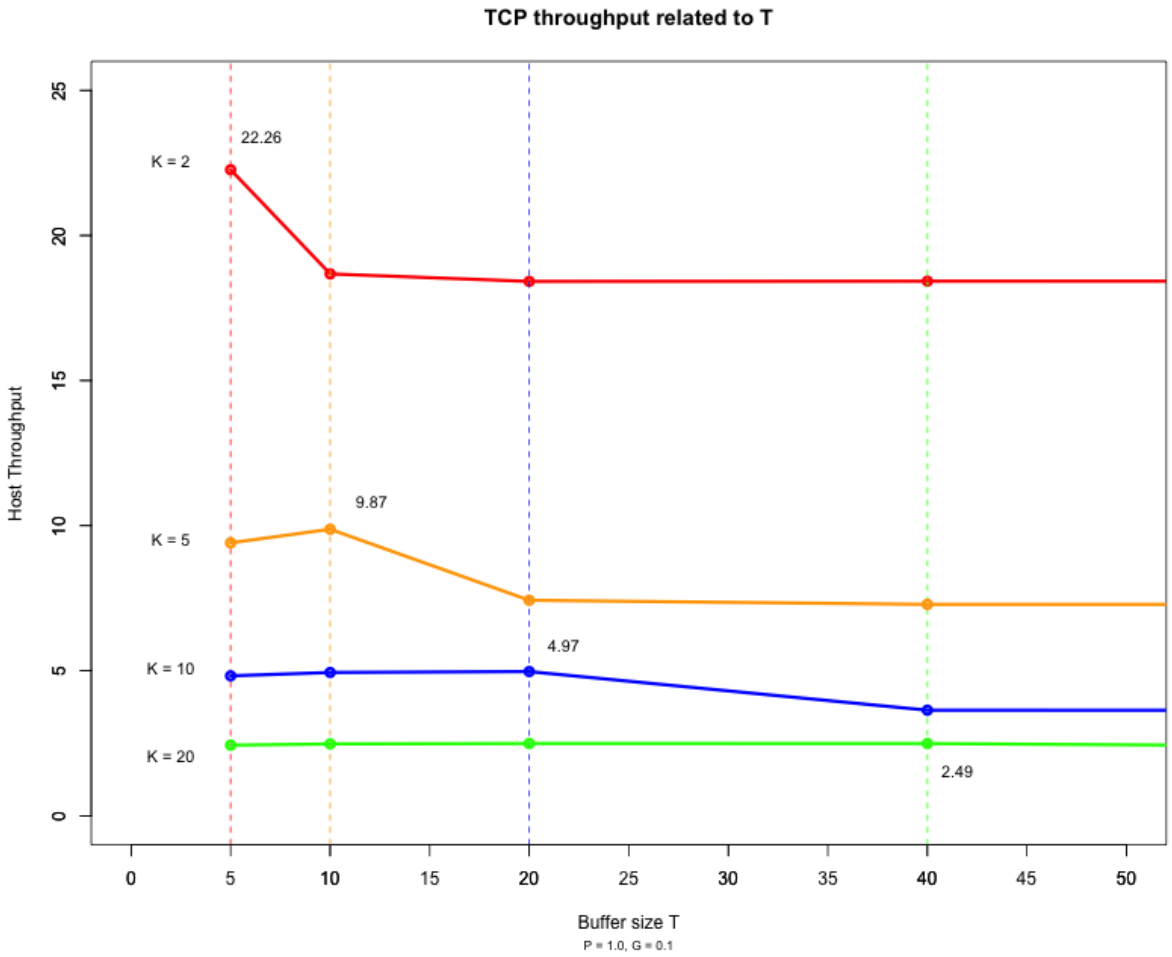

Figura 30: Throughput in relazione al numero di host

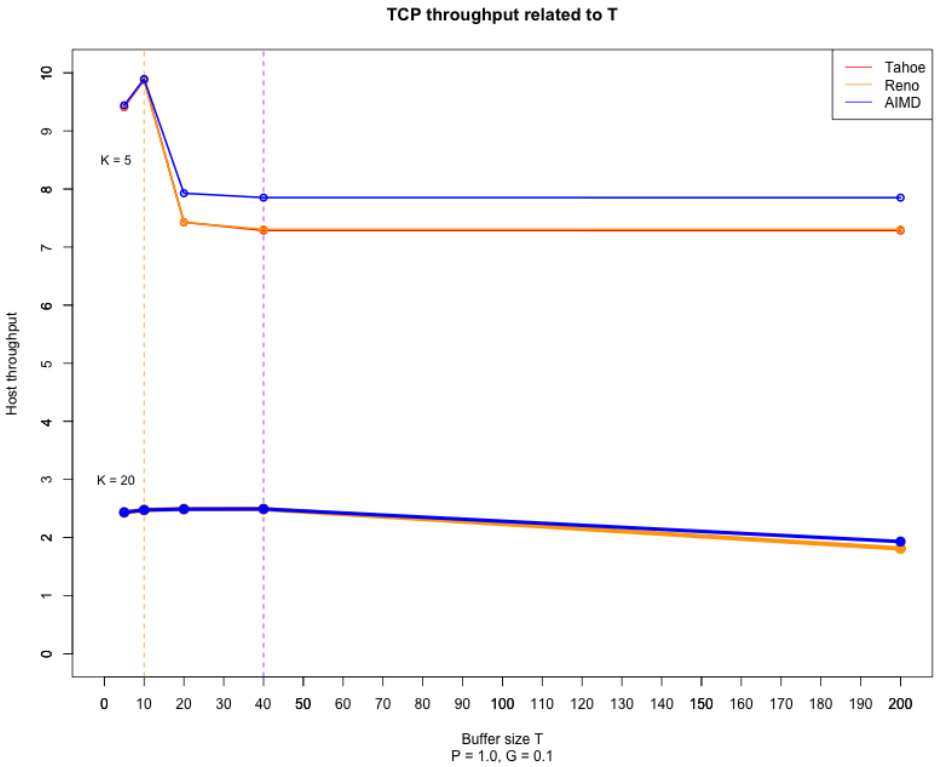

Figura 31: Protocolli a confronto

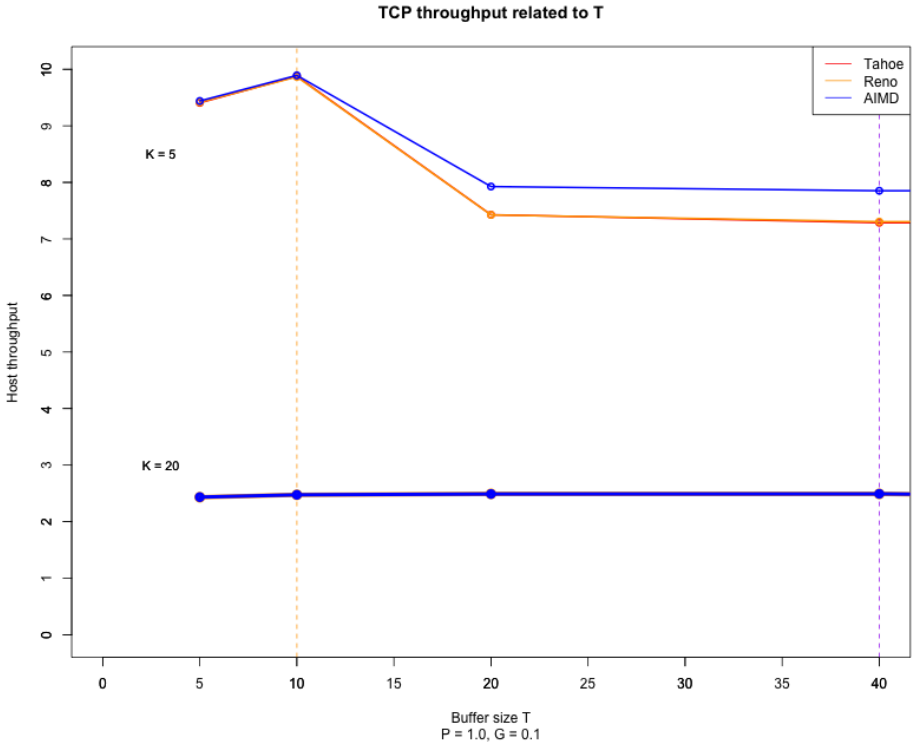

Figura 32: Protocolli a confronto, zoom

Di seguito vengono riportati i grafici riassuntivi che mostrano l'andamento del response time al variare dei parametri T,p,g,h:

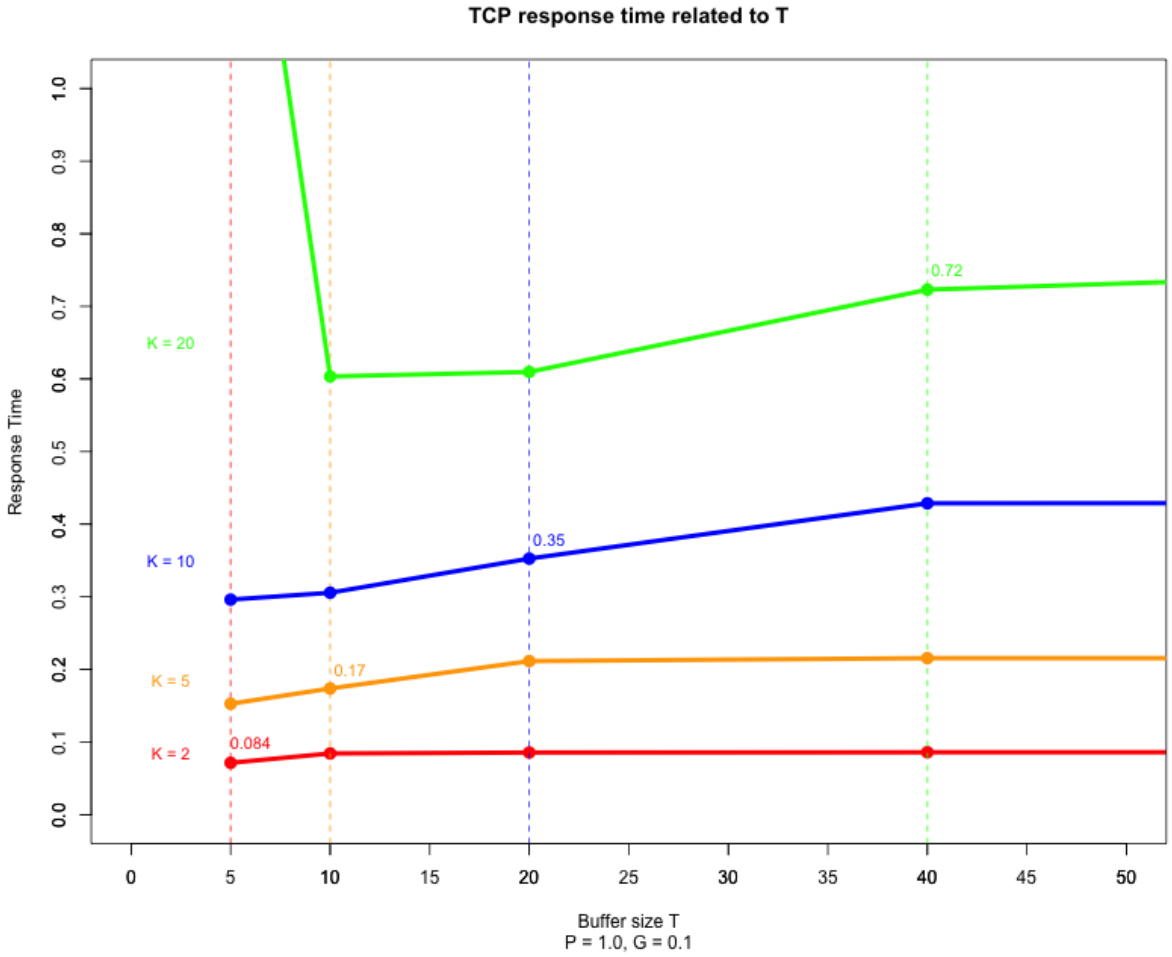

Figura 33: Response Time medio

Attraverso i grafici sopra riportati abbiamo potuto concludere diverse cose:

- ◇ I protocolli Reno e Tahoe sembrano dare all'incirca le stesse performance, dimostrazione del fatto che le congestion window in queste simulazioni non crescono di molto lasciando i due protocolli molto simili. Il protocollo AIMD sembra invece comportarsi meglio tenendo congestion window poco più ridotte e con tempi medi di risposta più piccoli. Queste conclusioni sono molto più evidenti quando siamo in presenza di un canale condiviso da pochi host (2-5), invece non notiamo molta differenza quando il numero di host aumenta ancora di più, in quanto il throughput per singolo host si restringe mantenendo le congestion window dei 3 protocolli molto basse.
- ◇ In presenza di un canale con alti valori di affidabilità, P tendente o uguale ad 1, il valore ottimale per la grandezza del buffer è pari a 2*K. Anche in questo caso il risultato tende ad essere meno influente quando il numero di host cresce, infatti come si può notare dai grafici sovrastanti la retta di ottimizzazione tende ad appiattirsi fino a diventare quasi una retta orizzontale. Mentre per la scelta della grandezza del buffer in funzione del response time, sembra che il valore ottimale si pari a K.
  Errore di valutazione: Un errore che si potrebbe fare è quello di pensare che avendo un

canale perfetto il throughput migliore lo si ottiene con buffer infinito. Questo però non è vero in quanto più aumentano il numero di segmenti presenti sul canale più aumenta il loro tempo di attesa, di conseguenza anche il loro tempo di risposta, portando ad un throughput più basso. Ricordiamo che il path per payload e ack è lo stesso, dunque immaginando di avere 40 segmenti in buffer avremmo che il primo ack arriverebbe dopo l'elaborazione dell'ultimo payload.

- ◇ Con situazioni in cui l'affidabilità del canale è meno buona, sappiamo già dal punto precedente che il throughput diminuisce però anche la grandezza del buffer ha la sua influenza. Infatti contrariamente a prima l'aumento della dimensione del buffer non influenza moltissimo i valori del throughput. Notiamo che i valori ottimali di throughput anche in questo caso dipende dal numero di host, infatti ci aspettiamo che il canale non essendo perfetto porti le congestion window degli host molto vicine al 1-2. Con numero di host pari a 2-5-10 immaginiamo di avere circa 20 segmenti sul canale, per cui i tempi di risposta riescono ad essere ancora accettabili tanto da portare ad un buon throughput. La differenza tra avere un buffer da 10, 20 o 40 non è cosi influente proprio per il valore ridotto delle cw. Con numero di host pari a 20 la situazione cambia. Infatti anche mantendendo cw pari a 2 il canale si ritroverebbe ad avere 40 segmenti sul canale, in questo caso si preferisce droppare qualche segmento per dare minor tempo di risposta ad altri.
- ◇ Comune a tutti i punti è il fatto di non tenere la dimensione del buffer troppo ridotta perchè porterebbe a molti drop che aumenterebbero il response time medio e la riduzione del throughput. Inoltre si cerca di non tenere troppi segmenti sul canale in quanto solo al loro servizio si sceglierà se il segmento è corrotto o meno. Dunque immaginando la situazione precedente con 40 segmenti sul canale avremmo che l'ultimo segmento, dopo aver aspettato 0.4ms per la sua elaborazione, risulta essere corrotto, portando ad una perdita di tempo per gli altri segmenti.

Un'altra analisi appurtata è stata anche quella di valutare il throughput massimo al variare del numero di stazioni che condividono il canale.

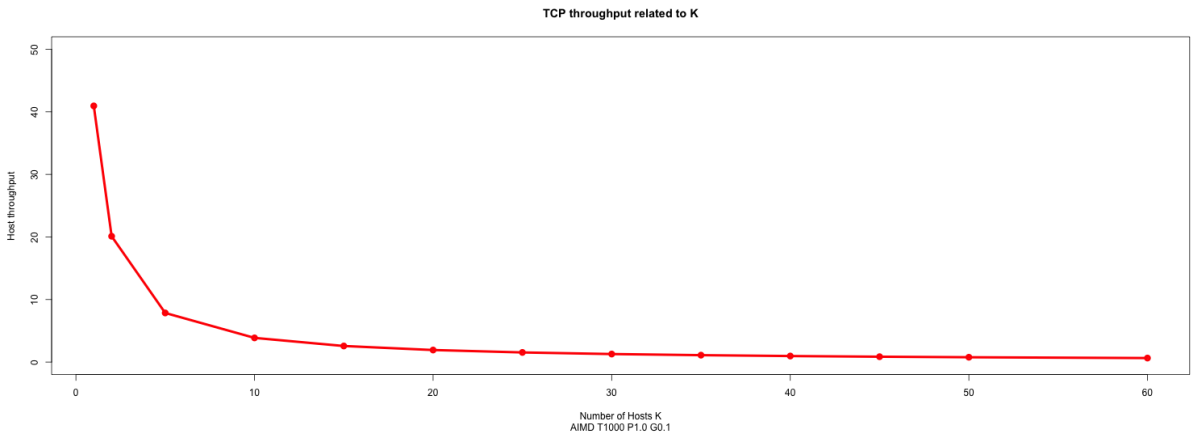

Figura 34: Throughput al variare del numero di host sulla rete

La capacità totale del canale è di 100000 segm/s, di cui gli host ne utilizzano 50000 segm/s considerando l'invio del payload e l'invio dell'ack. Più aumenta il numero di host più la banda totale viene divisa tra essi in quanto la capacità del canale è limitata a quel valore e le stazioni sono greedy.