

Wi-Fi Throughput with Block ACKs

Zanotto Enrico

Sommario

Introduzione	1
Funzionamento generale	2
Stadi della sessione	2.1
Composizione dei vari frames	2.2
Calcolo del throughput	3
Throughput traffico UDP senza Block Acknowledgment	3.1
Throughput con utilizzo di Block Acknowledgment	3.2
Test del throughput	4
Gestione dei dati	4.1
Test senza Block Ack	4.2
Test con Block Ack	4.3
Conclusioni	4.3

1. Introduzione

Block Acknowledgment è una tecnica introdotta inizialmente nello standard 802.11e e resa definitiva nella versione 802.11n. Il suo scopo è quello di incrementare l'efficienza nella trasmissione dei dati al livello MAC.

Ciò viene ottenuto raggruppando acknowledgment di più frame in un unico frame di risposta denominato BA (Block-Ack) diminuendo in generale il numero di frame trasmessi e aumentando l'effettivo utilizzo del canale di comunicazione.

Lo scopo di questo documento sarà quello di fornire una analisi del throughput effettivo riscontrato al livello applicazione in relazione all'utilizzo di questa tecnica e alle sue varianti.

2. Funzionamento generale

Prima di utilizzare un BA è necessario che i due host stabiliscano una sessione.

Questa sessione opera solo in una direzione: per utilizzare un BA nella direzione opposta è necessario crearne una nuova.

2.1 Stadi della sessione

1. Una delle due stazioni invia un frame *ADDBA request*. Tale frame verrà ricevuto dal destinatario dei frames, che risponderà con un *ADDBA response*. Questi due frame vengono confermati con un normale ack. Essi hanno lo scopo di concordare i vari parametri di sessione.

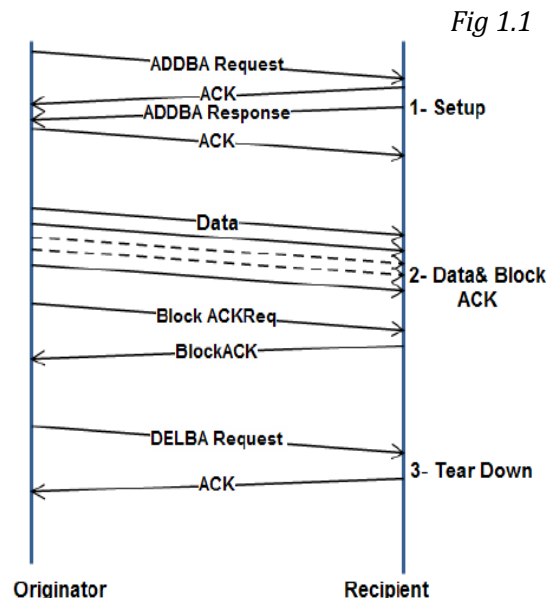
2. Il mittente quindi procede con l'invio dei vari frame separati solo dal SIFS time.

Una volta terminati i pacchetti (o raggiunto il valore di threshold) il mittente trasmette un *BA request* chiedendo essenzialmente all'host destinatario di inviare il BA contenente una bitmap rappresentante gli acknowledgment dei vari frame ricevuti.

3. A questo punto se i pacchetti da inviare sono terminati la trasmissione si conclude e la sessione viene chiusa dal mittente con l'invio di un *DELBA request*.

Se, invece, sono presenti nel buffer altri pacchetti da inviare si ricomincia dal punto 2 senza chiudere la sessione corrente.

I frame non confermati dal destinatario verranno subito ritrasmessi nel blocco di frame successivo.



2.1. Composizione dei vari frame

Frame di controllo

Ad ogni utilizzo del Block Acknowledgment necessario stabilire una **sessione**.

Questo viene fatto tramite l'invio del **ADDBA request** e del **ADDBA response** nella quale i due soggetti concordano i parametri per la trasmissione.

Parametri che includono BlockAck policy (immediate or delayed), TID, Buffer size, ecc...

Al termine della trasmissione la sessione verrà conclusa con l'invio del **DELBA request**.

Ognuno di questi tre tipi di frame viene confermato con un normale acknowledgment.

Data frame

I Data-frame sono dei normali frame contenenti il payload e incapsulati secondo lo standard 802.11 utilizzando la politica QoS, obbligatoria nelle sessioni di block ack.

Questi frame devono avere il flag "Ack policy" settato a "Block Ack" all'interno del QoS control field.

I data-frame vengono spediti in blocco distanziati dal SIFS time

Block ACKReq

BA Request è un frame inviato dal mittente del traffico per ordinare al destinatario di trasmettere il BlockAck. Questo frame viene generato nei due seguenti casi:

1. I pacchetti da trasmettere sono terminati e il buffer è vuoto.
2. Si è raggiunto il threshold di frame inviabili tra un BlockAck e l'altro. Questo limite può variare da 0 a 64, viene determinato all'inizio della sessione e resta invariato fino al termine della stessa.

BlockACK

Fig 1.2

Un BA è composto principalmente da una bitmap. Questa struttura dati contiene 64 entries, rappresentanti i 64 frame massimi inviabili tra un BlockAck e un altro.

Ogni entry della bitmap è composta da un array di 16 bit come conseguenza del fatto che ogni pacchetto può essere diviso in massimo 16 frames inviati e confermati separatamente.

Ogni pacchetto infatti, prima di essere incapsulato, viene frammentato in sezioni di dimensione uguale o inferiore ad un certo valore di **threshold** determinato in fase di creazione della sessione.

Nel caso la frammentazione non venga attivata o nel caso un pacchetto sia più piccolo del parametro di threshold, esso viene confermato con un singolo bit nella bitmap (frame 6 e 8 nella Fig. 1.2).

```
▼ blockAckBitmap[64] (inet::BitVector)
[0] 1111111111111111
[1] 1111111111111111
[2] 1111111111111111
[3] 1111111111111111
[4] 1111111111111111
[5] 1111111111111111
[6] 1000000000000000
[7] 0000000000000000
[8] 1000000000000000
[9] 0000000000000000
```

3. Calcolo del Throughput

Il Framework INET permette di simulare una comunicazione tra due host che utilizzano o meno questa tecnica analizzando il throughput a livello applicazione.

Lo showcase Throughput permette di comprendere in che modo questo valore può essere ottenuto e analizzato nel contesto di queste simulazioni.

Analiticamente è abbastanza semplice ottenere una stima delle prestazioni che ci si aspetta a livello applicazione di una data simulazione. Questo valore dipenderà dal tempo di trasmissione medio di ogni frame e dalla dimensione del payload del livello superiore:

$$throughput = \frac{payload}{tempoDiInvioDelFrame}$$

Il tempo di invio del frame in 802.11 è composto dagli intervalli di tempo occupati **in secondi** delle seguenti sezioni: DIFS, backoffTime, frameTime, SIFS, ACK

Il payload è invece composto dai dati che verranno poi passati al livello trasporto.

Essi costituiranno l'effettivo traffico che il livello applicazione sarà in grado di osservare.

Il throughput è quindi una grandezza misurabile in **bit al secondo bit/sec** e rappresenta l'effettiva velocità di trasmissione percepita al livello applicazione.

3.1. Throughput traffico UDP senza Block Acknowledgment

Il calcolo del throughput ottenuto da una comunicazione tra applicazioni che utilizzano UDP a livello trasporto risulta essere particolarmente affidabile nel caso siano presenti solo due dispositivi nella rete.

Inoltre, nel caso analizzato, solo uno dei due dispositivi invia i pacchetti mentre l'altro li riceve.

Di seguito il calcolo dei vari parametri che influiscono nella stima finale del throughput:

DIFS: $SIFS + 2 \times SlotTime$

SIFS: $16\mu s$

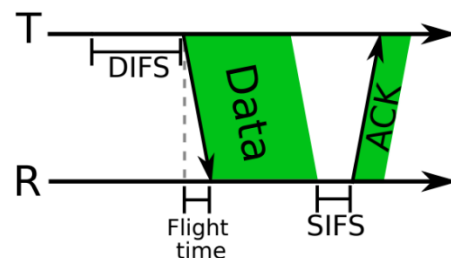
BackOffTime:

Tempo di attesa casuale scelto in una finestra che varia di dimensione in funzione delle collisioni avvenute nel canale di trasmissione secondo la seguente regola.

$$0 < time < 2^i CW_{min} \quad \text{Dove } i \text{ rappresenta il numero di collisioni}$$

Dato che in questi test non possono verificarsi collisioni e che questi tempi "estratti" seguono una distribuzione uniforme è possibile ottenere il tempo medio atteso:

$$BackoffTime = \frac{(0\mu s + 320\mu s)}{2} = 160\mu s \quad \text{con } CW_{min} = 320\mu s \text{ (dalla documentazione di inet)}$$



AckTime:

Ogni acknowledgment viene inviato con un bitrate pari a 24 Mbps (inferiore a quello utilizzato per i dati). Inoltre, dato che ogni frame di ack ha una dimensione di 27B è possibile determinare il tempo da esso occupato: $24/(27*8) = 9 \mu s$

FrameTime:

Al fine di stimare il throughput nel modo descritto nel paragrafo precedente è infine fondamentale definire la dimensione di ogni **frame** composto da: payload + headers aggiunti dai protocolli che stiamo utilizzando ai livelli 4, 3, 2 e 1.

Questo valore, diviso per la velocità di trasmissione del canale, darà il **frameTime**

Header lvl 4 (UDP)	Header lvl 3 (Ipv4)	Header lvl 2 (802.2 LLC)	Header + Trailer lvl 2 (802.11 MAC)	Header lvl 1 (preamble)
8 Byte	20 Byte	8 Byte	34Byte	22 Byte
TOTALE OVERHEAD: 92 Byte				

La dimensione effettiva del frame sarà quindi pari al **payload + 92 Byte**.

Dato che il bitrate massimo di 802.11 è pari a 54Mbps è possibile ottenere il FrameTime in una trasmissione ideale con la seguente formula:

$$FrameTime = \frac{(payload + 92) * 8}{54} \mu s$$

Sommando DIFS, SIFS, backoffTime, ACKtime al frameTime possiamo quindi stimare il throughput effettivo in funzione della dimensione del payload ottenendo i seguenti risultati:

$$throughput = \frac{payload * 8}{201 + FrameTime} mbps$$

payload (Byte)	100	400	600	800	1000	1500	2000
Frame Time (μs)	27,6	72	101,6	131,3	160,9	235	309
Tempo di invio totale (μs)	227,6	272	301,6	331,3	360,9	435	509
Throughput (Mbps)	3,5	9,3	13,9	17,6	22,1	27,5	31,4

Si può notare che il throughput viene influenzato enormemente dalla dimensione del payload iniziale determinato a livello trasporto.

Notiamo che anche con segmenti molto grandi (2000 Byte) il throughput effettivo risulta essere circa il 60% del bitrate reale utilizzato nella comunicazione.

Si scende invece ad un'efficienza del 6,5% utilizzando segmenti di 100 Byte.

3.2. Throughput con utilizzo di Block Acknowledgment

Il vantaggio di utilizzare il BlockAck è principalmente quello di diminuire in generale il numero di frame trasmessi raggruppando i vari acknowledgment in un unico frame, riducendo in questo modo il tempo di invio totale effettivo di ogni pacchetto.

Tuttavia nell'utilizzo di questa tecnica vengono introdotti 4 nuovi frame di controllo (descritti nel capitolo 2) che riducono l'efficienza della comunicazione.

ADDBA request, ADDBA response DELBA request.

Questi tre tipi di frame vengono utilizzati nella creazione e nella chiusura di una sessione di BlockAck.

Tuttavia se il traffico è continuo (come nel caso delle simulazioni fatte) la sessione viene aperta all'inizio e chiusa al termine dell'intera simulazione. Questo rende trascurabile l'overhead causato da questi 3 frame (e dai relativi ack) che vengono utilizzati una singola volta ciascuno.

I due frame di controllo che invece influiscono maggiormente sul throughput sono il **Block ACKReq** e il **BlockAck**.

Block ACKReq

La dimensione di questo frame varia intorno al valore di **55Byte** (headers e trailers compresi). Esso viene trasmesso ad un bitrate massimo di 24Mbps. Sommando il tempo di SIFS da attendere tra un invio e l'altro, il tempo occupato da questo frame risulta essere circa: **35 μs**

BlockACK

Il frame di BlockAck, contiene principalmente una bitmap rappresentante gli acknowledgment dei vari frame di dimensione 64x16 bit = 128 Byte.

Sono presenti inoltre altre informazioni tra cui il sequenceNumber e il fragmentNumber che portano il payload del frame a 130 Byte.

Aggiungendo infine i vari headers otteniamo una dimensione totale di circa **160 Byte**.

Il frame BlockAck viene trasmesso ad un bitrate di 24Mbps in un tempo di $(160 \cdot 8) / 24 = 53 \mu s$. A questo tempo va però aggiunto il SIFS time.

Il tempo totale di trasmissione di questo frame sarà quindi **70 μs** circa.

Data Frame

I vari frame di dati vengono inviati in serie separati solo dal **SIFS** time.

Il tempo di invio del frame è proporzionale alla dimensione del payload e possiede un'overhead identico a quello presente nei normali frames che utilizzano la normale tecnica di acknowledgment. Questo permette di applicare la medesima formula usata in precedenza

$$FrameTime = \frac{(payload + 92) \cdot 8}{54} \mu s$$

Contesa del canale

Infine è necessario considerare la modalità usata e il tempo impiegato nella contesa del canale. Durante l'utilizzo del BlockAck, al livello MAC, viene attivato il servizio di **QoS** basato su **EDCA** (Enhanced Distributed Channel Access) anziché DCF (Distributed coordination function).

EDCA assegna ad ogni host in attesa di trasmettere una finestra temporale denominata "Transmit opportunity" o "XTOP" la cui dimensione varia in base al **tipo di contenuto** presente dei frame.

Nel nostro caso **XTOP** è pari a **1500 µs** (assegnato da INET alle trasmissioni "VOICE")

All'interno di questa finestra è indifferente il tipo di frame inviato (Dataframe, BlockAck ...). Al termine di questo intervallo l'host cede il canale e attende il tempo **AIFS** (Arbitration inter-frame space) Di dimensione **≈450 µs**

Calcolo finale

Indico con:

- "**payload**" la dimensione del pacchetto in Byte
- "**threshold**" il numero di frame attesi prima di inviare il BlockAck
- "**BAtime**" tempo occupato dall'invio di BlockAckReq + BlockAck

$$FrameTime + SIFS = \frac{((payload + 92) * 8)}{54} \mu s + 25 \mu s$$

$$AIFSsWaitPerFrame = \frac{FrameTime + SIFS + \frac{BAtime}{threshold}}{1500}$$

$$throughput = \frac{payload * 8}{FrameTime + SIFS + \frac{BAtime}{threshold} + 450 * AIFSsWaitPerFrame} \text{ mbps}$$

payload (Byte)	100	300	500	700	1000	1500	2000
Frame Time (us)	28,4	58,1	87,7	117,3	161,8	235,9	309,9
Frame Time + SIFS (us)	53,4	83,1	112,7	142,3	186,8	260,9	334,9
Throughput (Mbps) threshold = 4	7	15,6	20,8	24,3	27,7	31,2	33,3
Throughput (Mbps) threshold = 16	9,9	20,1	25,3	28,5	31,5	34,2	35,8
Throughput (Mbps) threshold = 32	10,7	21,1	26,3	29,4	32,2	34,8	36,3
Throughput (Mbps) threshold = 64	11,1	21,6	26,8	29,8	32,6	35,1	36,5

4. Test del throughput

Al fine di verificare le stime fatte e analizzare il guadagno ottenuto dall'utilizzo del BlockAck in un ambiente reale sono stati effettuati diversi gruppi di esperimenti utilizzando il simulatore Omnet++. I test sono stati costruiti modificando ed espandendo lo showcase "throughput" all'interno del framework INET messo a disposizione da Omnet++.

Lo showcase inizialmente prevedeva una rete ad-hoc tra 2 dispositivi che utilizzano la normale tecnica di acknowledgment. Al fine di adattare la simulazione alle nostre necessità sono state effettuate le seguenti alterazioni al file **.ini**

```
# simulations are composed of 100 runs of 1 second each
sim-time-limit = 1s
repeat = 100

# enabling the use of QoS data frames (necessary in BlockAck session)
*.Host.wlan[*].mac.qosStation = true
*.Host.wlan[*].classifier.typename = "QosClassifier"
```

Sono state inoltre create **4 diverse configurazioni** per analizzare separatamente in che modo diversi parametri, come il FrameThreshold o la lunghezza del pacchetto, influiscono sul traffico.

```
[Config NormalRun]
description = "1500B packets, no fragmentation, no BlockAck "
*.sourceHost.app[*].messageLength = ${packetLength = 1500}byte
*.sourceHost.app[*].sendInterval = ${packetLength} * 8 / ${bitrate} * 1us
*.Host.wlan[*].mac.hcf.isBlockAckSupported = false

[Config BlockAck]
description = "1500B packets, no fragmentation, BlockAck with threshold of 16"
*.sourceHost.app[*].messageLength = ${packetLength = 1500}byte
*.sourceHost.app[*].sendInterval = ${packetLength} * 8 / ${bitrate} * 1us
*.Host.wlan[*].mac.hcf.isBlockAckSupported = true
*.Host.wlan[*].mac.hcf.originatorAckPolicy.blockAckReqThreshold = 16

[Config blockAckThresholds]
description = "comparing different frames thresholds before sending the BA"
*.sourceHost.app[*].messageLength = ${packetLength = 1500}byte
*.sourceHost.app[*].sendInterval = ${packetLength} * 8 / ${bitrate} * 1us
*.Host.wlan[*].mac.hcf.isBlockAckSupported = ${isBlockAckSupported=true}
*.Host.wlan[*].mac.hcf.originatorAckPolicy.blockAckReqThreshold =
${blockAckReqThreshold=4,8,12,16,20,24,28,32,36,40,44,48,52,56,60,64}

[Config blockAckFragmentations]
description = "comparing BlockAck vs NoBlockAck with different frames length"
*.sourceHost.app[*].messageLength = ${packetLength =
100,300,500,700,1000,1500,2000}byte
*.sourceHost.app[*].sendInterval = ${packetLength} * 8 / ${bitrate} * 1us
*.Host.wlan[*].mac.hcf.isBlockAckSupported = ${isBlockAckSupported=true,false}
*.Host.wlan[*].mac.hcf.originatorAckPolicy.blockAckReqThreshold = 64
```


4.1. Gestione dei dati

Le informazioni raccolte nei vari esperimenti sono state estratte e salvate in dei file .csv.

Grazie a alle librerie di Python questi dati sono stati poi elaborati e formattati correttamente al fine di trarre informazioni coerenti che permettono un'analisi corretta delle performance

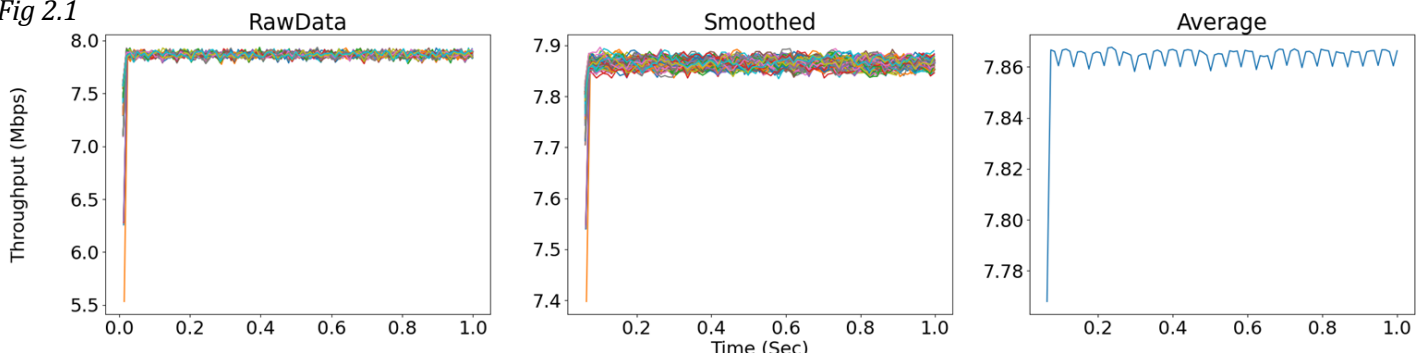
Welch method

In primo luogo è fondamentale applicare il metodo di Welch per stabilire da che istante la simulazione è effettivamente entrata a regime raggiungendo una situazione di equilibrio.

Sono state quindi eseguite delle simulazioni che fanno utilizzo del Block Ack e delle simulazioni che non ne fanno uso. Ogni simulazione dura 1000ms e contiene 100 run separate.

Dalle prove si è ricavato che le varie simulazioni basate sul normale sistema di ack entrano quasi istantaneamente in una situazione di equilibrio. Tuttavia nelle simulazioni con il BlockAck è ben visibile una fase di transizione iniziale.

Fig 2.1



Dai grafici (Fig. 2.1) si può notare che questa fase iniziale ha una durata inferiore ai 100ms. Ipotizzo che questo intervallo di transizione sia dovuto a due principali fattori:

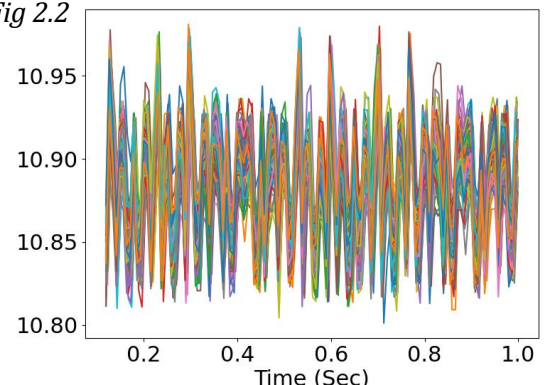
1. La creazione della sessione di BlockAck richiede del tempo
2. Fino a che il primo BlockAck non viene inviato i frame non vengono considerati "ricevuti" e di conseguenza nei primi istanti il throughput risulterà essere zero.

Media e mediana

Nelle varie analisi dei dati che seguiranno, si è scelto di utilizzare principalmente il valore di media.

Come è possibile vedere dal grafico (Fig. 2.2), una volta rimossi i primi 50ms il throughput non assume mai valori estremi che potrebbero influenzare negativamente la media.

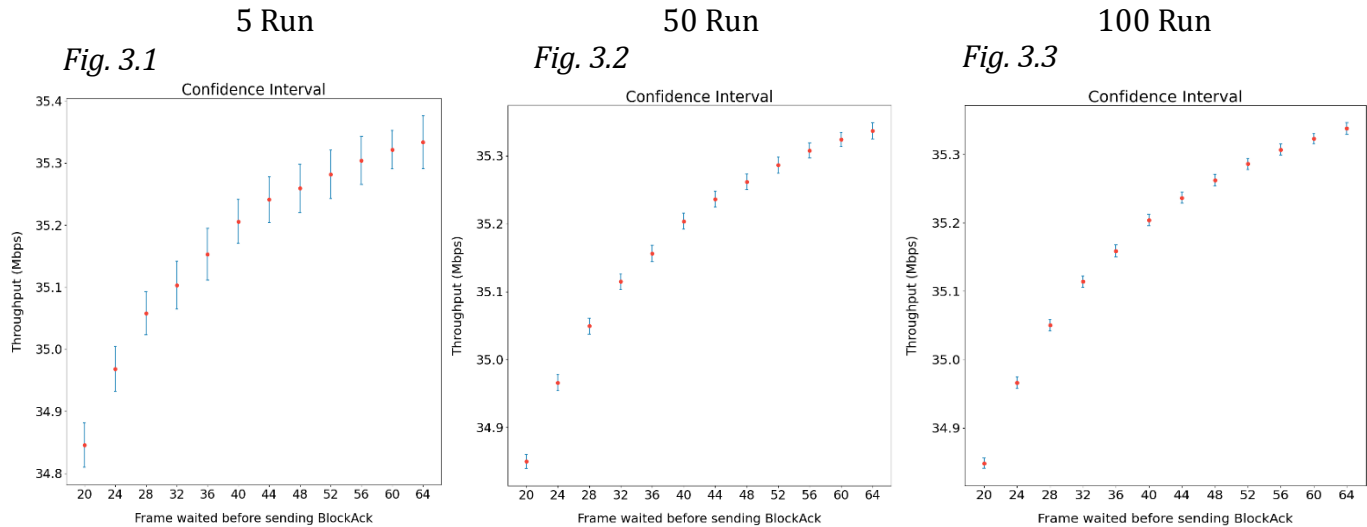
Fig 2.2



Intervalli di confidenza

Di seguito sono riportati 3 grafici (*Fig. 3.1 3.2 3.3*) rappresentanti gli intervalli di confidenza riguardo al throughput di esperimenti dove sono state eseguite rispettivamente 5, 50 e 100 run. In questi esperimenti si è andati a variare il numero di frame attesi prima dell'invio del BlockAck. In questo esempio si è scelta una dimensione del payload pari a 1500B.

I vari intervalli di confidenza sono stati calcolati con una precisione del 95%.



Assumendo che la dimensione del payload non influisca più di tanto sulla precisione dei risultati è chiaro che un numero di run pari a ~100 sia più che sufficiente al fine di ottenere dei valori accurati.

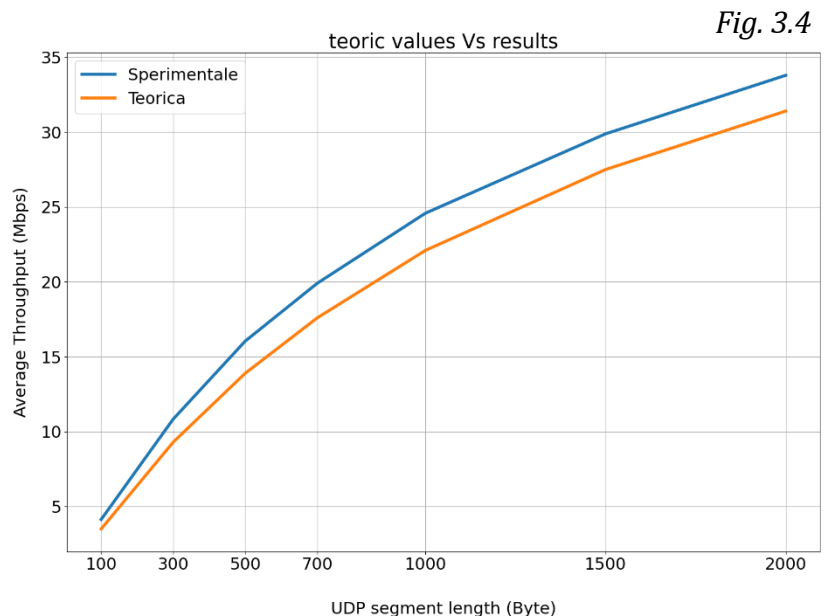
4.2. Test senza BlockAck

In seguito è stata effettuata una serie di prove per comparare il throughput stimato con i risultati ottenuti dal simulatore senza l'utilizzo di BlockAck.

Le prove sono state suddivise in funzione della dimensione del payload di livello trasporto.

Si può notare dal grafico (*Fig. 3.4*) come l'errore relativo non sia costante ma vari in funzione del payload:

- ~15% per payload di 100B
- ~11% per payload di 700B
- ~7% per payload di 2000B



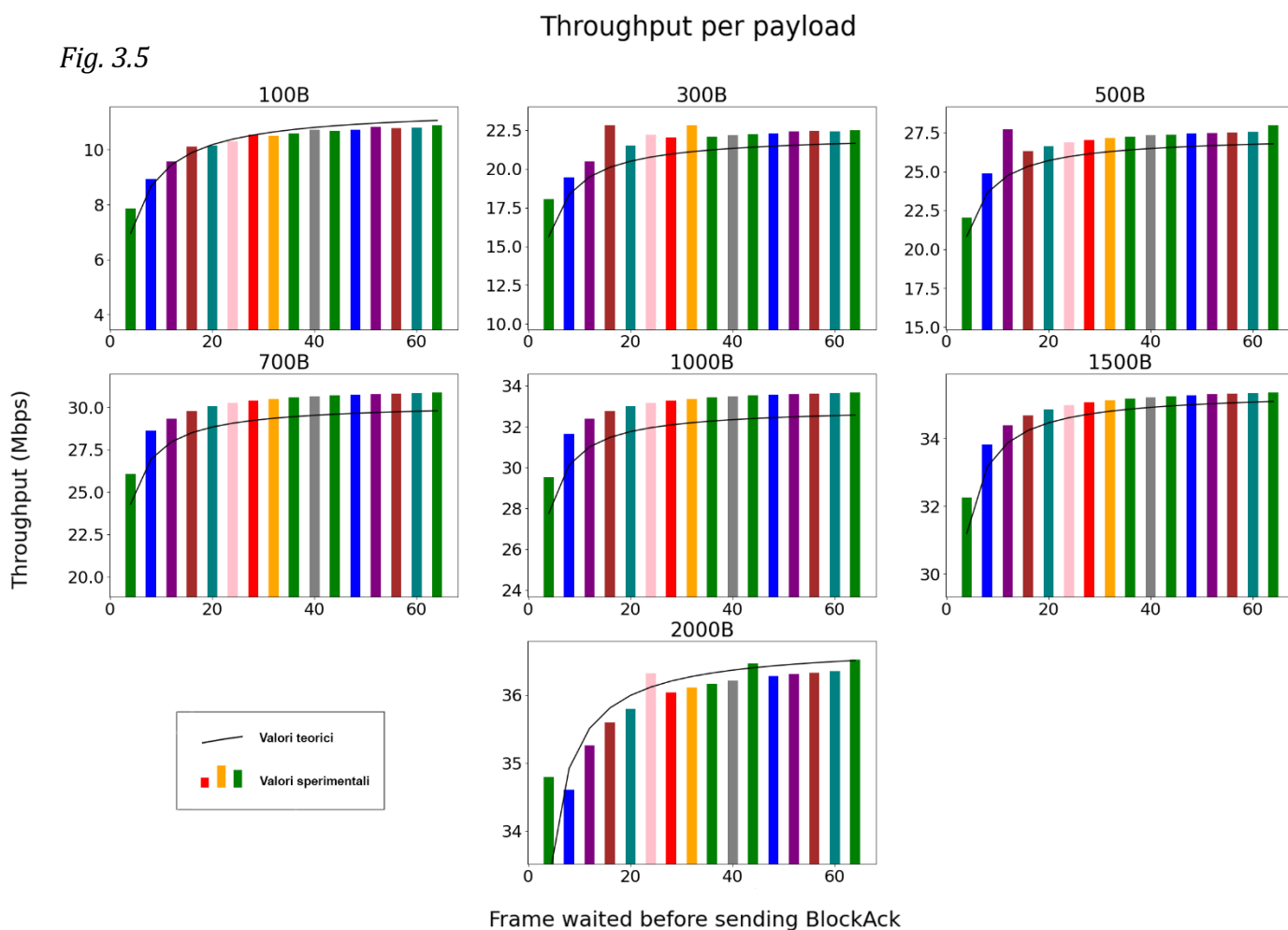
4.3. Test con BlockAck

Successivamente è stata introdotta la tecnica del BlockAcknowledgment alternando il file .ini come precedentemente descritto.

Sono stati quindi eseguiti diversi esperimenti alterando 2 parametri:

- Dimensione del Payload (100B, 300B, 500B, 700B, 1000B, 1500B, 2000B)
- BATHreshold: Frame attesi prima dell'invio del BlockAck (4,8,12, ... 64)

La *fig. 3.5* rappresenta i risultati ottenuti dai vari esperimenti comparandoli ai valori teorici attesi.



Si può notare che in alcuni casi i valori ottenuti dal simulatore sono leggermente al di fuori della norma.

Questo fenomeno è particolarmente visibile nei grafici rappresentanti le simulazioni con 300, 500 e 2000 Byte

Nel resto degli esperimenti invece (*Fig3.6*) la contesa del canale viene fatta, di volta in volta, in una fase diversa della sessione di BlockAck.

Entrambe le Figure 3.6 e 3.7 rappresentano un test con l'utilizzo di un payload da 500B e valori di threshold rispettivamente di 16 e 12 frame.

Osservando i tempi di arrivo dei vari frame si può risalire al tempo impiegato nella contesa del canale.

Nel caso in cui la contesa avvenga tra due data-frame il tempo di attesa introdotto risulta essere pari a **239us**.

Particolarmente inferiore è invece il tempo dedicato alla contesa del canale nel caso “fortunato” dove si attenderebbe solamente **108μs**.

Il fenomeno che rende questi tempi intra-frame più lunghi del previsto riguarda un NAV-time inaspettato che la stazione mittente decide di attendere al termine di ogni Transmit-Opportunity. In questo intervallo di tempo il canale resta completamente inutilizzato.

Come si può leggere dal log (Fig. 3.8) i parametri per la contesa del canale vengono decisi ma la contesa effettiva non inizia fino a che questo NAV-time non scade (Fig. 3.9)

```

** Event #83569 t=0.130073452724 Throughput.sourceHost.wlan[0].radio (Ieee80211ScalarRadio, id=29) on selfmsg transmissionTimer (onnetpp::CMessage, id=27)
INFO: Transmission ended: (inet::physicalLayer::WirelessSignal)UDPData--8244 (54 us 192 B) (inet::Packet)UDPData--8244 (192 B) (inet::SequenceChunk) length = :
INFO: Changing radio transmission state from TRANSMITTING to IDLE.
DETAIL (Tx)Throughput.sourceHost.wlan[0].mac.tx: Tx: radioTransmissionFinished()
INFO (Hcf)Throughput.sourceHost.wlan[0].mac.hcf: Finishing last frame sequence step: history = ((DATA) (DATA) (DATA) (DATA) (DATA) (DATA) (DATA) (DATA))
INFO (Hcf)Throughput.sourceHost.wlan[0].mac.hcf: Processing transmitted frame UDPData--8244 as originator in frame sequence.
INFO (Hcf)Throughput.sourceHost.wlan[0].mac.hcf: Starting next frame sequence step: history = ((DATA) (DATA) (DATA) (DATA) (DATA) (DATA) (DATA) (DATA))
INFO (Hcf)Throughput.sourceHost.wlan[0].mac.hcf: Frame sequence finished.
INFO (Edcaf)Throughput.sourceHost.wlan[0].mac.hcf.edca.edcaf[3]: Channel released.
INFO (TxopProcedure)Throughput.sourceHost.wlan[0].mac.hcf.edca.edcaf[3].txopProcedure: Txop ended.
DETAIL (Contention)Throughput.sourceHost.wlan[0].mac.hcf.edca.edcaf[3].contention: Starting contention: cw = 3, slots = 0, slotTime = 0.000009, ifs = 0.000028
INFO (Rx)Throughput.sourceHost.wlan[0].mac.rx: Setting NAV to 0.000152
INFO: Changing radio transmitted signal part from DATA to NONE.
```

Risulta inoltre che il NAV viene calcolato in funzione dell'ultimo frame inviato da quella stazione. Questo influenza il throughput ulteriormente in funzione della dimensione del payload motivando anche i "picchi" di throughput presenti nei grafici nella *Fig. 3.5*.

Dato che questo NAV viene calcolato in base all'ultimo frame inviato, nelle simulazioni dove il termine della *Transmission-Opportunity* coincide ogni volta con l'invio del BlockAck, è normale aspettarsi un sostanziale guadagno in performance. (date le dimensioni ridotte del frame di BA)

Fig. 3.6

```

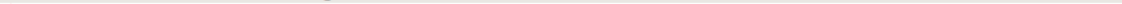
0.261 645 995 500 sourceHost → destinationHost UDPData--3325
0.261 769 995 500 sourceHost → destinationHost UDPData--3327
0.261 893 995 500 sourceHost → destinationHost UDPData--3330
0.262 017 995 500 sourceHost → destinationHost UDPData--3332
0.262 141 995 500 sourceHost → destinationHost UDPData--3334
0.262 504 995 500 sourceHost → destinationHost UDPData--3335
0.262 628 995 500 sourceHost → destinationHost UDPData--3337
0.262 752 995 500 sourceHost → destinationHost UDPData--3339
0.262 876 995 500 sourceHost → destinationHost UDPData--3344
0.263 000 995 500 sourceHost → destinationHost UDPData--3345
0.263 124 995 500 sourceHost → destinationHost UDPData--3347

```

Fig. 3.7

```
0.290 421'446'992 sourceHost → destinationHost UDPData--3727 10
0.290 545'446'992 sourceHost → destinationHost UDPData--3728 10
0.290 669'446'992 sourceHost → destinationHost UDPData--3730 10
0.290 793'446'992 sourceHost → destinationHost UDPData--3732 10
0.290 917'446'992 sourceHost → destinationHost UDPData--3733 10
0.291 041'446'992 sourceHost → destinationHost BasicBlockAckReq
0.291 093'454'974 destinationHost → sourceHost BasicBlockAck 0A
0.291 289'446'992 sourceHost → destinationHost UDPData--3735 10
0.291 413'446'992 sourceHost → destinationHost UDPData--3737 10
0.291 537'446'992 sourceHost → destinationHost UDPData--3738 10
```

Fig. 3.8



DETAIL (Hcf)Throughput.sourceHost.wlan[0].mac.hcf: Requesting channel for access category Voice
 ** Event #83644 t=0.130225452724 Throughput.sourceHost.wlan[0].mac.rx (Rx, id=37) on selfmsg NAV (omnetpp::cMessage, id=22)
 INFO: The radio channel has become free according to the NAV
 INFO (Contention)Throughput.sourceHost.wlan[0].mac.hcf.edca.edcaf[3].contention: Scheduling contention end: backoffslots = 0, slotTime = 0.000009,

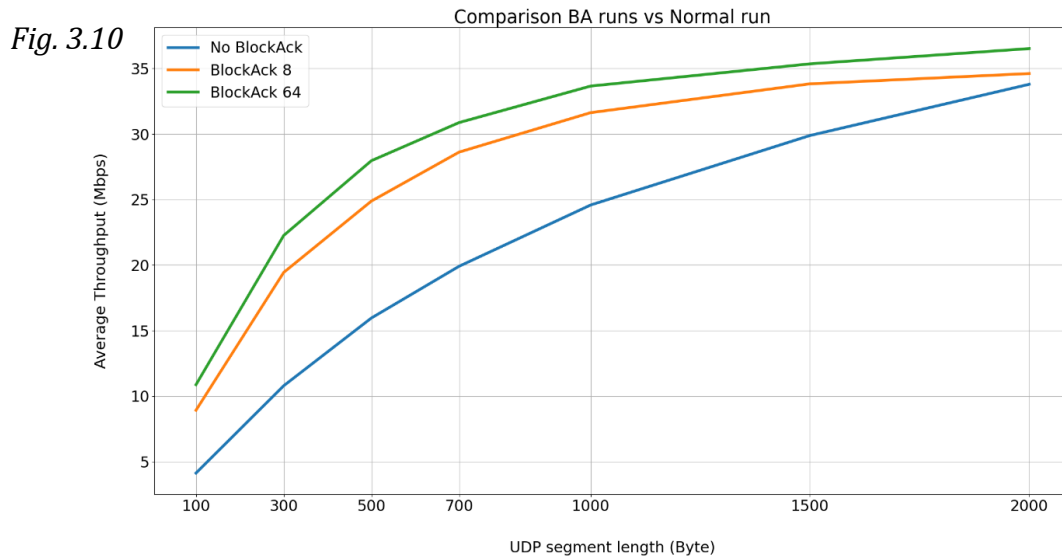
Fig. 3.9

Fig. 3.9

BlockAck Vs No BlockAck

Il seguente grafico (*Fig. 3.10*) permette di comparare il throughput di una normale trasmissione con quello ottenuto introducendo il BlockAcknowledgment.

Si può notare come il guadagno in prestazioni diminuisca con l'aumentare della dimensione del payload.



Block Ack gains per packet length

Fig. 3.11

Un altro modo per visualizzare i medesimi risultati è tramite il seguente grafico a barre (*Fig. 3.11*).

La figura rappresenta il guadagno di throughput ottenuto dall'utilizzo del BlockAck in rapporto a quello ottenuto senza di esso.

I risultati sono espressi in percentuali.

I dati utilizzati per rappresentare l'efficacia del BlockAck sono stati presi dagli esperimenti nella quale il threshold di frame di attesi è pari a 64 (valore massimo)

