Università degli Studi di Padova

Facoltà di Ingegneria

Corso di laurea magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Progetto e simulazione di un protocollo di accesso al mezzo per radio multi portante in reti di sensori

Relatore: Dott. Michele Rossi
Correlatore: Ing. Nicola Bui

Laureando: Riccardo Bonetto

05 10 2011

Questo documento é stato scritto in LaTeX su Debian GNU/Linux.

Dedico questa tesi alla mia famiglia che sempre mi ha garantito un incondizionato supporto.
Grazie.

Indice

Elenco delle figure

Sommario

In questa tesi si presenta un nuovo protocollo di accesso al mezzo, multicanale, a basso overhead e a ridotto consumo energetico specificamente progettato per l'applicazione nell'ambito delle reti wireless di sensori: Multi Channel Medium Access Control (McMac).

Nello svolgimento di questo documento viene presentata un'introduzione allo standard di riferimento per le tecnologie appartenenti alla categoria delle Low Rate Wireless Personal Area Nerwork (LRWPAN): lo standard IEEE 802.15.4. In seguito il lettore viene introdotto allo stato dell'arte dei protocolli di accesso al mezzo per le reti di cui sopra, evidenziando le linee guida che hanno indirizzato la progettazione di protocolli già affermati, sottolineando inoltre, con vigore, la necessità di soluzioni multicanale in grado di sfruttare le risorse, in termini spettrali, offerte dallo standard IEEE 802.15.4 mantendo allo stesso tempo ridotto il carico computazionale a cui di dispositivi sono soggetti.
Lo strumento utilizzato per l'implementazione e il testing del protocollo McMac è il framework NS-3 che, sempre più, si sta dimostrando, di fronte alla comunità scientifica, un valido strumento di supporto allo studio prestazionale delle più svariate architetture di rete.
Una volta introdotti il contesto operativo e gli strumenti utilizzati, si procederà nella descrizione del protocollo oggetto di questo lavoro, le scelte progettuali affrontate, le modifiche che è stato necessario apportare al framework NS-3, al fine di aggiungere il supporto, non presente, allo standard IEEE 802.15.4. Successivamente si discuterà l'organizzazione del processo di testing finalizzato all'ottenimento di risultati statisticamente validi per un'accurata analisi prestazionale.

In conclusione verranno presentati e commentati i dati ottenuti per mezzo di simulazioni, verrà evidenziata l'efficienza dimostrata dal protocollo e verranno discussi i punti di debolezza e le possibili soluzioni da implementare in un lavoro futuro che voglia proseguire sulla strada che, con questa tesi, si sta aprendo.

Capitolo 1

Introduzione

Le reti di sensori wireless (*Wireless Sensor Networks*, WSN nel seguito) sono una tecnologia che, nel corso dell'ultimo decennio, ha visto la propria progressiva affermazione e diffusione nell'ambito di numerosi campi applicativi quali per esempio il monitoraggio ambientale, la domotica, i sistemi anti intrusione e, in generale, qualunque progetto che richieda capacità di calcolo pervasiva a basso costo e comunicazioni senza fili.
La molteplicità delle possibilità applicative di questa tecnologia ha posto, e continua a porre, impegnative sfide a ricercatori e aziende in tutto il mondo, in particolare nel caso in cui una rete debba operare in condizioni sfavorevoli che ne impediscono la manutenzione e il controllo (si pensi ad esempio a sistemi di monitoraggio ambientale in ambienti estremi quali foreste, profondità marine o deserti). In queste situazioni si richiede che una rete sia in grado di operare in autonomia per lunghi periodi di tempo continuando a garantire un buon grado di affidabilità. Al fine di ottenere questi risultati, i dispositivi che, interconnessi, creano la rete devono essere progettati in modo da poter operare sfruttando fonti di alimentazione indipendenti quali batterie o fonti naturali (per mezzo di tecniche di *energy harvesting*). Un primo attacco al problema prevede la progettazione e realizzazione di hardware ad hoc per questo tipo di applicazioni, caratterizzato quindi da componenti a basso consumo di piccole dimensioni e relativamente economici; l'utilizzo di dispositivi elettronici dotati delle caratteristiche sopra elencate, tuttavia, non è sufficiente da solo ad assicurare l'estensione del periodo di funzionalità globale della rete ai livelli generalmente desiderati. Studi prestazionali sui vari dispositivi presenti nel mercato hanno evidenziato come la causa primaria di dispendio energetico sia l'attività del radiotrasmettitore, da queste considerazioni si evince, quindi, l'importanza che ha assunto lo studio di politiche di gestione delle comunicazioni tagliate sulle necessità operative di questa nuova tecnologia.
Lo studio di questi problemi ha portato alla definizione di svariati protocolli di ac-

cesso al mezzo ottimizzati per le esigenze delle WSN. Di alcuni di questi protocolli verrà data una descrizione nel seguito di questa trattazione al fine di consentire al lettore di conoscere lo stato dell'arte nel campo della progettazione di protocolli di accesso al mezzo per reti di sensori wireless. Il lavoro svolto per questa tesi si inserisce nel succitato contesto proponendo McMAC: un nuovo protocollo MAC completamente distribuito in grado di sfruttare la capacità di cambiare a *run time* la frequenza portante propria dei moderni radiotrasmettitori, permettendo in tal modo di utilizzare i molteplici canali previsti dallo standard IEEE 802.15.4 al fine di aumentare considerevolmente il *throughput* globale di una rete grazie alla possibilità di instaurare parallelamente più connessioni, fino a sfruttare l'intero spettro previsto per tali applicazioni dallo standard IEEE 802.15.4.
Il protocollo ivi proposto è stato implementato e testato nell'ambiente di simulazione *open source* NS-3, un simulatore di reti a eventi che si sta affermando sempre di più grazie alla sua elegante flessibilità e a un nutrito e attivo gruppo di utilizzatori e sviluppatori.

Requisiti necessari per la fruizione del lavoro oggetto di questa tesi sono l'aver compreso e fatti propri i concetti fondamentali dei protocolli di accesso al mezzo, una certa dimestichezza con i fenomeni elettromagnetici che entrano in gioco nel campo delle trasmissioni radio e una buona conoscenza del linguaggio C++.

Il capitolo 2 tratta dello standard IEEE 802.15.4. In questo capitolo viene descritta l'impostazione generale dello standard, vengono affrontate le motivazioni che hanno portato alla definizione delle sue principali caratteristiche e vengono presentate le strutture e le primitive previste per il funzionamento dei primi due livelli dello *stack* protocollare ISO/OSI per tecnologie appartenenti alla categoria delle *low rate wireless personal area network* (LR-WPAN), di cui le *wireless sensor network* sono un sottoinsieme.

Nel capitolo 3 si offre una panoramica sullo stato dell'arte dei protocolli di accesso al mezzo per le reti di sensori. Vengono ivi descritti diversi approcci proposti in letteratura, porgendo particolare attenzione alla dicotomia che sussiste tra protocolli basati su politiche di *scheduling* e quelli basati invece su tecniche di *preamble sampling*. Per ciascuno dei protocolli presentati viene data breve spiegazione del funzionamento e delle motivazioni che hanno portato alla sua realizzazione, ne vengono inoltre evidenziati i principali aspetti favorevoli e sfavorevoli.

Nel capitolo 4 viene introdotto il *framework* di simulazione NS-3, ne vengono spiegati, con esempi, il funzionamento e le peculiarità principali. Al termine del capitolo vengono infine introdotte le modifiche che è stato necessario apportare

al simulatore al fine di creare il supporto adatto al protocollo McMac, trattato nel capitolo 5.

Il capitolo 5 ha come oggetto il nucleo fondante del lavoro presentato in questa trattazione, il protocollo *Multi Channel Medium Access Control*: McMac. Nello svolgersi del capitolo verranno introdotte le idee che hanno portato alla definizione della struttura del protocollo, i problemi riscontrati nella fase di progettazione e le soluzioni adottate. Nella seconda parte del capitolo verranno, invece, descritti il processo di implementazione del protocollo all'interno del simulatore NS-3, le scelte architetturali adottate, e i miglioramenti apportati all'idea originale.

Il capitolo 6 introduce il successivo presentando il processo che ha portato all'ottenimento dei risultati. In particolare vengono spiegati l'impostazione delle simulazioni, i parametri caratteristici al variare dei quali si è ritenuto opportuno studiare il comportamento del protocollo e i risultati di interesse che si desiderava ottenere.

Nel capitolo 7 vengono presentati e commentati i risultati ottenuti per mezzo delle simulazioni. Viene inoltre presentato un confronto prestazionale col noto e affermato protocollo XMAC.

Il protocollo presentato in questa sede non rappresenta un prodotto finale, è, bensì, il risultato della progettazione di una nuova tecnologia in grado di sfruttare nella sua interezza lo spettro disponibile nel campo delle reti di sensori. Visti i primi risultati incoraggianti non si esclude che, nel futuro, questo progetto venga raffinato fino a divenire, sperabilmente, un prodotto completo utilizzabile in applicazioni reali.

Capitolo 2

Lo standard IEEE 802.15.4

Visto il diffondersi delle tecnologie legate all'ambito delle *Low Rate Wireless Personal Area Network* (LR-WPAN), più comunemente dette *Wireless Sensor Network* (WSN), è sorta la necessità di definire uno standard che specificasse le politiche di funzionamento di queste nuove reti in modo da garantire l'interoperabilità tra diverse tecnologie.
Lo standard IEEE 802.15.4 [80206] è il risultato di questo processo, esso definisce il comportamento del *physical layer* e del *MAC layer* che un dispositivo che si voglia far operare nell'ambito delle WSN deve mantenere.

Lo standard IEEE 802.15.4 è diventato il punto di riferimento sia dal punto di vista della ricerca sia dal punto di vista commerciale per la tecnologia oggetto di questo lavoro. Si è pertanto deciso di attenersi a tale standard.

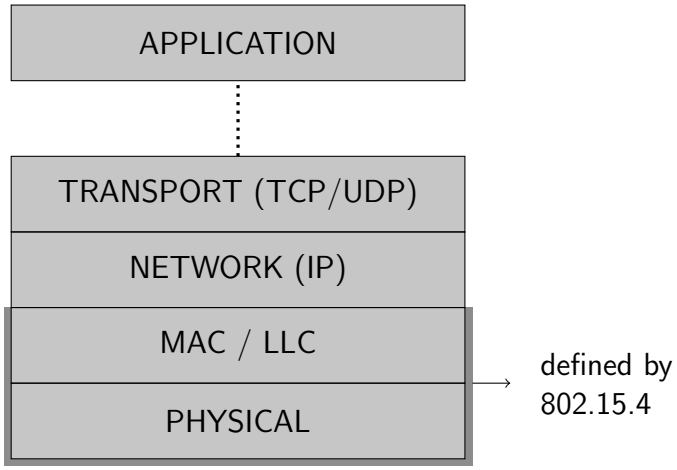


Figura 2.1: *Protocol stack* e campo di interesse dello standard IEEE 802.15.4

Un dispositivo IEEE 802.15.4 *compliant* può sfruttare tre diverse bande di frequenza a seconda dell'area geografica in cui opererà e della *bitrate* richiesta dalle applicazioni che la rete dovrà supportare:

- 868.0 - 868.6 MHz (Europa)
- 902 - 928 MHz (Nord America)
- 2400 - 2483.5 MHz

La scelta della banda in cui operare, oltre a influire sulla *bitrate* di trasmissione implica l'utilizzo di tre diverse forme di modulazione dei segnali radio:

- Per le bande 868.0-868.6 MHz e 902-928 MHz si hanno:
  - *Direct Sequence Spread Spectrum* con *Binary Phase Shift Keying* [Gol05];
  - *Direct Sequence Spread Spectrum* con *Offset Quadrature Phase Shift Keying* [Gol05];
  - *Parallel Sequence Spread Spectrum* con *Binary Phase Shift Keying* e *Amplitude Shift Keying* [Gol05].
- Per la banda 2400-2483.5 MHz, invece, è previsto l'utilizzo di un *Direct Sequence Spread Spectrum* associato a una modulazione *Offset Quadrature Phase Shift Keying* [Gol05].

All'interno di queste bande è possibile sfruttare un totale di 27 canali numerati da 0 a 26. Le frequenze centrali di tali canali sono definite come segue:

- $F_c = 868.3$ MHz (un solo canale disponibile) nella banda 868.0 - 868.6 MHz;
- $F_c = 906 + 2(k-1)$, $k = 1, 2, \ldots, 10$ nella banda 902 - 928 MHz;
- $F_c = 2405 + 5(k-11)$, $k = 11, 12, \ldots, 26$ nella banda 2400 - 2483.5 MHz.

La banda generalmente più utilizzata è quella ISM a 2400 - 2483.5 MHz e nel seguito di questa trattazione si farà riferimento esclusivamente a quest'ultima, se non diversamente specificato.

Verranno ora descritti i principali aspetti di funzionamento del *physical layer* e *medium access control sublayer* definiti nello standard in questione.
Nel seguito del capitolo, per qualunque definizione, si faccia riferimento a [80206].

2.1 Premesse generali

Lo standard individua due categorie di dispositivi:

- RFD (*Reduced Function Device*);
- FFD (*Full Function Device*).

Mentre i dispositivi appartenenti alla prima possono comunicare solo a un FFD e vengono utilizzati per applicazioni semplici, i *Full Function Device* sono invece più versatili. Essi possono operare in tre diverse modalità all'interno di una rete:

1. PAN *coordinator*: in questa modalità un FFD viene reso coordinatore di un'intera rete e avrà quindi il compito di gestire l'associazione alla rete, la deassociazione dalla stessa, la sincronizzazione dei nodi e le questioni relative alla sicurezza delle comunicazioni (in particolare distribuzione delle chiavi di cifratura e loro validazione);
2. *Coordinator*, in questo caso il nodo investito di questa carica sarà responsabile di una porzione di rete;
3. *Device*.

Lo standard prevede, inoltre, due possibili topologie per una LR-WPAN:

- *Star*;
- *Peer to peer*.

Nel primo caso i nodi appartenenti alla rete possono comunicare solamente col dispositivo facente funzione di PAN *coordinator*; in una topologia *peer to peer*, invece, pur essendo in ogni caso presente un PAN *coordinator*, è prevista anche l'esistenza di link tra nodi facenti funzione di *device* (si veda figura 2.2).

2.2 IEEE 802.15.4 Physical Layer

Al livello fisico sono demandati i seguenti compiti:

- Accensione e spegnimento del radiotrasmettitore;
- *Energy detection* o, in altri termini, l'ascolto del canale per rilevare l'entità della potenza presente in esso;
- Generazione dell'indicatore della qualità del *link* per i pacchetti ricevuti;

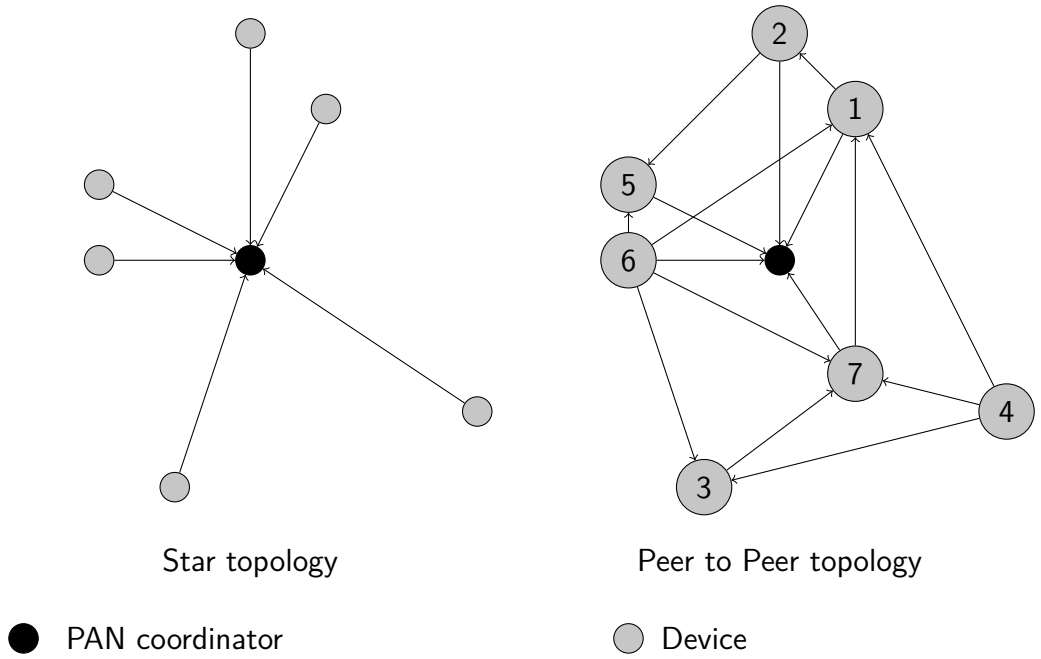


Figura 2.2: Esempi di *star topology* e *peer to peer topology*

- *Clear Channel Assessment* (necessario all'implementazione del protocollo CSMA-CA);
- Impostazione della frequenza portante relativa al canale desiderato;
- Trasmissione e ricezione dati.

Il *physical layer* fornisce principalmente due servizi a cui si può accedere attraverso due distinti *Service Application Provider* (SAP):

- Il PHY Data SAP (PD-SAP);
- Il PHY Management Service (PLME-SAP).

2.2.1 PD-SAP

Il *physical layer data service* è responsabile della trasmissione e ricezione dei dati. Esso prevede tre primitive e fornisce al livello superiore (il MAC) un'interfaccia verso il mezzo fisico di trasmissione.
Le primitive fornite sono:

- *PD-DATA.request*: attraverso questa primitiva il livello superiore comunica al livello fisico la volontà di iniziare una trasmissione.

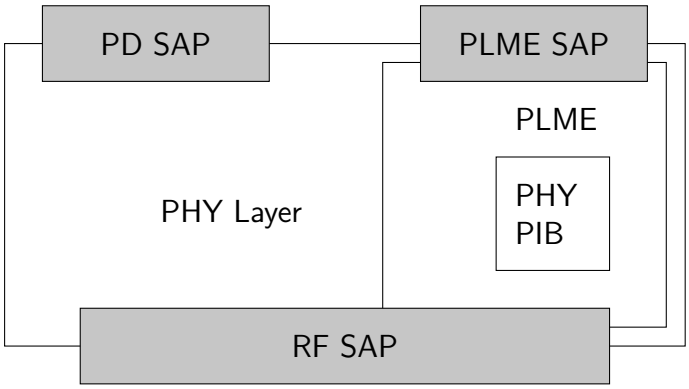


Figura 2.3: Modello di riferimento del *physical layer* IEEE 802.15.4

- *PD-DATA.confirm*: una volta elaborata una richiesta di trasmissione (pervenuta tramite *PD-DATA.request*), il livello fisico comunica al MAC l'esito della transazione. Questa primitiva viene chiamata sia nel caso in cui la trasmissione abbia avuto successo, sia nel caso in cui, invece, sia stata infruttuosa.

- *PD-DATA.indication*: questa primitiva viene invocata al momento di un'avvenuta ricezione per comunicare al livello MAC il completamento dell'operazione.

Attraverso queste tre primitive avvengono tutti gli scambi dati tra i dispositivi presenti nella rete.

2.2.2 PLME-SAP

Il *Physical layer management entity* mette a disposizione le primitive per l'impostazione dello stato e dei parametri caratteristici del *transceiver*. Le operazioni possibili attraverso questa interfaccia sono:

- PLME-CCA:
  - request: permette al MAC di richiedere l'inizio della procedura di *Clear Channel Assessment* che permette di conoscere lo stato del mezzo di trasmissione;
  - confirm: una volta terminata la procedura di CCA il livello fisico comunica al MAC il risultato. Quest'ultimo può essere *IDLE* o *BUSY*.

- PLME-ED:

  - request: permette al MAC di richiedere l'inizio della procedura di rilevazione di energia presente nel canale radio;
  - confirm: come nel caso della primitiva PLME-CCA, il risultato dell'energy detection viene comunicato al MAC attraverso la chiamata a PLME-ED.confirm.

- PLME-GET:
  - request: permette di richiedere il valore di uno dei parametri di impostazione del livello fisico (quali ad esempio il canale di trasmissione, la *bitrate*, lo stato del *transceiver* etc...);
  - confirm: una volta acquisito il valore del parametro richiesto, quest'ultimo viene comunicato al MAC attraverso questa primitiva.

- PLME-SET:
  - request: attraverso questa chiamata si richiede che il fisico imposti uno dei suoi parametri caratteristici a uno specifico valore passato come parametro;
  - confirm: il successo o meno dell'operazione richiesta viene, grazie a questa primitiva, comunicato al livello superiore.

2.2.3 Formato dei *frame* 802.15.4 a livello fisico

32 bit	8 bit	7 bit	1 bit	Variabile
Preamble	SFD	Frame Length	Reserved	PSDU

Figura 2.4: Struttura di un frame di livello fisico

- *Preamble*: il preambolo di livello fisico associato a ogni frame trasmesso permette la sincronizzazione del ricevitore rispetto ai simboli trasmessi, la sua durata è di $128\mu s$.

- *Start Frame Delimiter* (SFD): questo campo (come si può evincere dal nome) marca la separazione tra preambolo di sincronizzazione e inizio della porzione di *frame* contenente i dati utili. Preambolo e SFD formano il cosiddetto *synchronization header*.

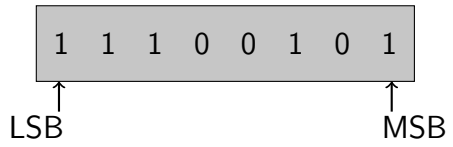


Figura 2.5: Start Frame Delimiter

- *Length*: il campo *length* specifica la lunghezza totale in *byte* del pacchetto trasmesso. Essendo la sua lunghezza pari a 7 bit, la lunghezza massima di un *frame* di livello fisico può essere dunque $2^7 - 1 = 127$ *byte*. Questo campo, inoltre, è l'unico elemento al livello fisico che consente un controllo sulla correttezza del *frame* da parte del ricevitore. Lo standard IEEE 802.15.4, infatti, non prevede procedure di correzione o rilevamento di errori a questo livello;

- *PSDU*: porzione del *frame* contente i dati.

2.2.4 Clear Channel Assessment

Lo standard prevede tre possibili metodi di CCA:

1. Energia al di sopra di una soglia fissata: in questo caso il canale viene dichiarato occupato se l'energia rilevata è al di sopra della soglia fissata, libero altrimenti;

2. *Carrier Sense*: il canale risulta occupato solo se viene rilevato un segnale corrispondente al tipo previsto dallo standard, le altre forme di energia vengono considerate rumore e non sono coinvolte nel processo di valutazione dello stato del canale;

3. *Carrier Sense* con energia al di sopra di una soglia fissata: in questo caso viene effettuata una combinazione logica dei primi due casi, l'operatore logico non viene specificato e può essere AND o OR.

La modalità da utilizzare per il CCA è specificata nell'attributo del livello fisico $phyCCAMode$ e può essere impostata attraverso la primitiva PLME-SET.

2.3 IEEE 802.15.4 MAC sublayer

Il livello MAC controlla tutti gli accessi al mezzo radio ed è responsabile delle seguenti operazioni:

- Generazione di *network beacon* nel caso in cui il device che implementa il livello sia un coordinatore;
- Sincronizzazione ai *network beacon* nel caso in cui, invece, il device che implementa il livello non sia un coordinatore;
- Supporto all'associazione e deassociazione;
- Supporto alla sicurezza;
- Implementazione del protocollo CSMA-CA per la regolamentazione degli accessi al mezzo radio;
- Mantenimento della temporizzazione globale della rete;
- Generazione di un *link* affidabile tra livelli MAC di nodi aventi le medesime funzioni;

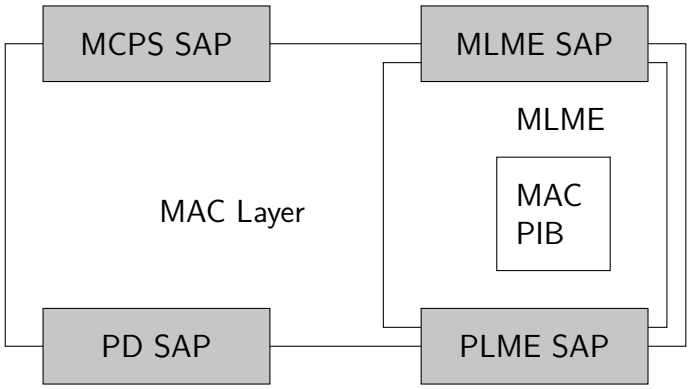


Figura 2.6: Modello di riferimento del *MAC sublayer* IEEE 802.15.4

Come nel caso del *physical layer*, il livello MAC è diviso in due unità funzionali:

- *MAC Common Part Sublayer* il cui scopo è fornire un'interfaccia per la trasmissione dei dati che permetta il collegamento del *layer* superiore al livello fisico;
- *MAC Layer Management Entity* il cui scopo è, invece, fornire al livello superiore un'interfaccia per la gestione e manipolazione dei parametri caratteristici del livello MAC.

Le primitive fornite per l'interazione con questo livello sono molteplici e una trattazione esaustiva di queste ultime esula dallo scopo di questo lavoro (per una trattazione esaustiva si veda [80206]).
Nel seguito verranno presentate le primitive principali che è necessario conoscere al momento di affrontare il progetto di un protocollo MAC.

2.3.1 MCPS-SAP

Questo gruppo di primitive fornisce l'interfaccia per lo scambio di dati tra il MAC *sublayer* e il livello superiore. Le funzionalità supportate sono trasmissione e ricezione e, in più rispetto al livello fisico, l'annullamento di una trasmissione in corso.
La tassonomia delle funzioni standard per questo livello segue quella già presentata per il livello fisico:

- MCPS-DATA.request: con questa primitiva il livello superiore richiede al livello MAC di avviare la trasmissione di un *frame*;
- MCPS-DATA.confirm: in seguito a una MCPS-DATA.request, il MAC comunica al livello superiore l'esito della transazione;
- MCPS-DATA.indication: il livello MAC comunica al livello superiore l'avvenuta ricezione di un frame;
- MCPS-PURGE.request: questa primitiva viene chiamata dal livello superiore quando è necessario interrompere una trasmissione avviata tramite la chiamata a MCPS-DATA.request e prima che si sia avuta conferma di avvenuta transazione;
- MCPS-PURGE.confirm: l'esito della chiamata a MCPS-PURGE.request viene comunicato dal MAC al livello superiore per mezzo di questa funzione.

2.3.2 MLME-SAP

Come nel caso del livello fisico, questo gruppo di primitive fornisce un'interfaccia per la gestione dello stato e dei parametri caratteristici del livello MAC.
In particolare le funzionalità offerte da questo gruppo di primitive possono essere classificate nei seguenti contesti:

- Associazione/Deassociazione;
- Notifica di ricezione di *Network Beacon*;
- Scrittura/Lettura degli attributi PIB (*PAN Information Base*);
- Gestione GTS;
- Notifica di stato *orphaned*, stato in cui un device (non coordinatore) ha perso il contatto con il proprio coordinatore;
- Specifica del tempo di attività del ricevitore;

- Scansione dei canali;
- Aggiornamento della configurazione dei *superframe*;
- Sincronizzazione con un coordinatore;
- Richiesta di dati da un coordinatore;
- Reset del MAC.

Le primitive di interesse per la fruizione di questo lavoro, tuttavia, riguardano solamente le attività di scrittura/lettura degli attributi PIB e di *reset* del livello MAC.
Il gruppo di funzioni che permette di richiedere ed ottenere il valore di un determinato parametro del MAC è composto dalla coppia:

- MLME-GET.request: attraverso la chiamata a questa funzione si richiede il valore di un particolare attributo contenuto nel PIB *database* proprio del livello;
- MLME-GET.confirm: una volta completata l'elaborazione della richiesta il livello MAC utilizza la chiamata a questa funzione per rispondere al livello chiamante.

Analogamente al caso precedente, per l'impostazione dei parametri caratteristici, del livello si hanno:

- MLME-SET.request;
- MLME-SET.confirm.

Per quanto riguarda, invece, le primitive che forniscono al livello superiore l'interfaccia necessaria a forzare un reset del livello MAC sono:

- MLME-RESET.request: la richiesta di reset forza il livello MAC a impostare il *transceiver* a *OFF* e a riportare i parametri propri del livello a quelli di default;
- MLME-RESET.confirm: il risultato della richiesta, come ormai risulterà evidente, viene comunicato al chiamante della MLME-RESET.request attraverso questa funzione.

2.3.3 L'algoritmo CSMA-CA secondo 802.15.4

Lo standard prevede due casi:

- *Slotted* nel caso in cui siano abilitati i *network beacon*;
- *Unslotted* altrimenti.

Ai fini di questa trattazione risulta utile lo studio del secondo caso (si veda figura 2.7), mentre lo studio del primo si applica principalmente alle situazioni in cui i nodi della rete utilizzino un protocollo MAC che necessita di esplicita sincronizzazione con il *clock* di un coordinatore attraverso periodici *network beacon*. Si suppone inoltre che il lettore abbia familiarità con i concetti fondamentali del protocollo CSMA-CA.
Ogni dispositivo deve mantenere tre variabili relative a ciascun tentativo di trasmissione:

- *NB*: corrisponde al numero di *backoff* che il MAC ha dovuto eseguire i relazione alla trasmissione corrente. All'inizio di ogni nuova trasmissione il valore di questa variabile deve essere impostato a zero;
- *CW*: utilizzata solo nel caso di *slotted* CSMA-CA, rappresenta l'intervallo di tempo durante il quale, pur avendo rilevato il canale *idle*, un nodo può incorrere in collisioni dovute al tempo di propagazione di segnali appena trasmessi da altri dispositivi;
- *BE*: rappresenta il cosiddetto *backoff exponent* ed indica quanti periodi di *backoff* il trasmettitore deve attendere prima di eseguire un nuovo *channel sense*.

2.3.4 Formato dei frame a livello MAC IEEE 802.15.4

Il livello MAC IEEE 802.15.4 prevede quattro tipi di *frame*:

- BEACON;
- DATA;
- ACKNOWLEDGMENT;
- COMMAND.

La struttura generale di un *frame* è presentata in figura 2.8. I tre bit meno significativi del campo *frame control* (rispettivamente di indici 0, 1, 2, si veda figura 2.9) determinano il tipo di pacchetto:

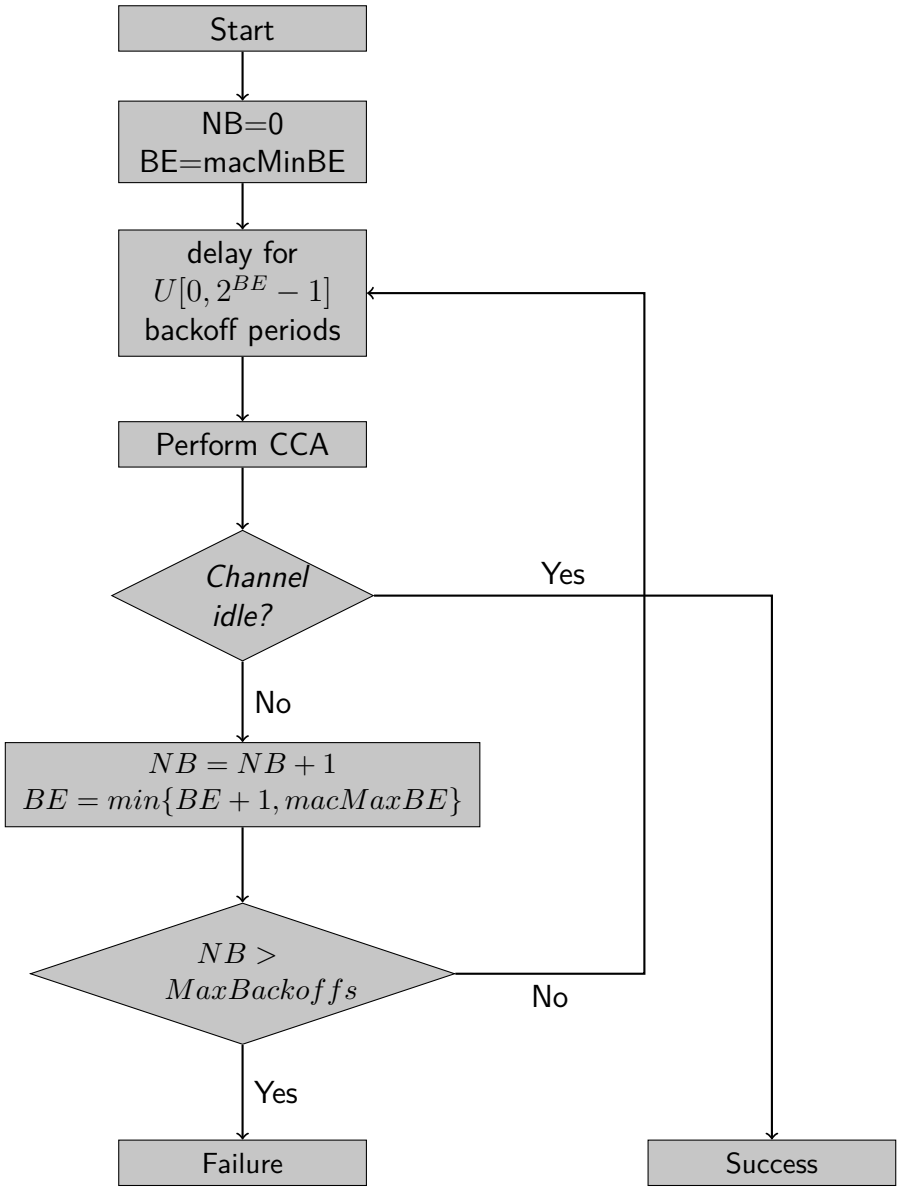


Figura 2.7: Diagramma di flusso del protocollo *unslotted* CSMA come definito in IEEE 802.15.4

1. 000 $\Rightarrow$ BEACON;
2. 001 $\Rightarrow$ DATA;
3. 010 $\Rightarrow$ ACKNOWLEDGMENT;
4. 011 $\Rightarrow$ COMMAND.

size in byte 2	1	0/2	0/2/8	0/2	0/2/8	0/5/6/10/14	variable	2
Frame control	Sequence number	Destination PAN id	Destination address	Source PAN id	Source address	Security Header	Payload	FCS
		Addressing fields						
MHR							Payload	MFR

Figura 2.8: Struttura di un frame di livello MAC in IEEE 802.15.4

bits 0-2	3	4	5	6	7-9	10-11	12-13	14-15
Frame type	Security enabled	Frame pending	Ack req	PAN ID compression	Reserved	Dst addr mode	Frame version	Src addr mode

Figura 2.9: Struttura del *frame control field* di un generico pacchetto MAC IEEE 802.15.4

Gli altri bit, invece, hanno i seguenti significati:

- Il terzo bit meno significativo è un *flag* che determina se la sicurezza è attivata o meno;
- Il quarto bit determina se vi è in attesa un pacchetto successivo a quello corrente (*frame pending*);
- Il quinto determina se è richiesto o meno un *acknowledgment*;
- Il sesto determina se la compressione dell'identificativo della *Personal Area Network* è abilitato;
- I bit dal 7 al 9 sono riservati;
- I bit 10 e 11 determinano il tipo di indirizzo della destinazione:
  1. 00 $\Rightarrow$ Non è specificato il PAN Id;
  2. 01 $\Rightarrow$ riservato;
  3. 10 $\Rightarrow$ indirizzo a 16 bit (*short address*);
  4. 11 $\Rightarrow$ indirizzo a 64 bit (*extended address*).
- I bit 12 e 13 determinano la versione del *frame*;
- I bit 14 e 15 determinano il tipo di indirizzo del mittente.

Gli altri campi che compongono un *frame* di livello MAC IEEE 802.15.4 non sono determinanti per questo lavoro e si rimanda il lettore a [80206].

Capitolo 3

Protocolli MAC per WSN: stato dell'arte

Generalmente un nodo di una WSN consiste in un microcontrollore equipaggiato con un processore a basso consumo, un radiotrasmettitore che implementa lo standard IEEE 802.15.4 e una serie di sensori scelti a seconda dell'utilizzo che si prospetta per la rete. In molti scenari operativi tali nodi vengono alimentati per mezzo di batterie e devono operare in totale indipendenza.
Nonostante l'utilizzo di componenti hardware progettati ad hoc per essere utilizzati in reti di sensori, analisi prestazionali e prove sul campo hanno evidenziato come ciò non sia sufficiente a garantire un adeguato periodo di attività delle reti destinate ad operare in completa autonomia.
Viste le succitate limitazioni la ricerca si è presto indirizzata verso lo studio di politiche di accesso al mezzo e gestione del *transceiver* (individuato come principale responsabile di dispendio energetico) specifiche per l'applicazione in reti di sensori ed in grado di ridurre ulteriormente il consumo energetico relativo alle comunicazioni. Sono state quindi individuate quattro principali cause di deperimento delle fonti di alimentazione:

- Collisioni;
- *Overhearing*: quando, cioè, un nodo riceve pacchetti che non gli competono;
- *Overhead*: la necessità di inviare e ricevere pacchetti di controllo aumenta i consumi limitando nel contempo la possibilità di trasmettere dati utili;
- *Idle listening*: questa condizione si verifica quando il *transceiver* di un nodo è acceso senza però che siano in corso trasmissioni che lo coinvolgono. L'*idle listening* è stato individuato come la principale causa di dispendio energetico.

Tutti i protocolli di accesso al mezzo fin'ora proposti hanno l'esplicito intento di mitigare l'influenza dei quattro fattori sopra descritti allo scopo di aumentare l'aspettativa di vita di ciascun sensore appartenente a una rete.
In prima analisi le soluzioni proposte possono essere suddivise in due categorie:

- *Schedule based*;
- *Contention based*.

Nel primo caso i periodi di *idle listening* e *overhearing* vengono in larga misura ridotti per mezzo di una accurata sincronizzazione tra i nodi. Queste tecniche permettono di eliminare le collisioni al prezzo, però, di un maggiore *overhead* necessario a mantenere sincronizzati i *clock* dei nodi che, data la necessità di hardware economico, tendono ad essere imprecisi. Data la natura fortemente gerarchica delle politiche basate su *scheduling* queste mal si adattano a casi in cui la topologia della rete sia soggetta a frequenti variazioni (dovute ad esempio a mobilità dei nodi) e, inoltre, richiedono la presenza di un coordinatore (in genere dotato di potenza di calcolo superiore a quella degli altri nodi) in grado in ogni momento di definire lo *scheduling* ottimo - secondo il particolare algoritmo implementato - basandosi, in genere, su informazioni aggregate provenienti dai vari dispositivi presenti nella rete.

Le tecniche basate su contesa, invece, si basano in generale su varianti del

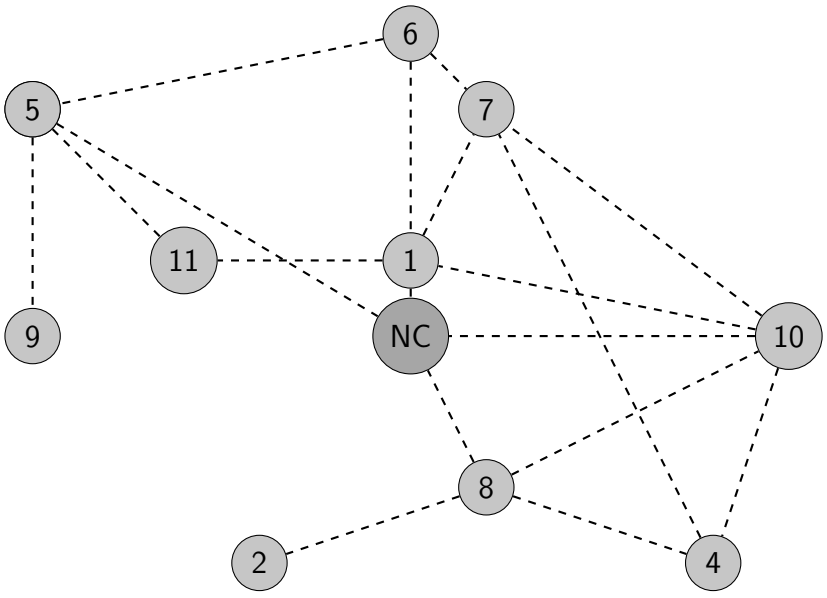


Figura 3.1: Tipica struttura di una WSN

famoso protocollo CSMA (*Carrier Sense Multiple Access*) e non richiedono che

vi sia una esplicita politica di sincronizzazione tra i nodi della rete rendendosi, in questo modo, più adatte alla gestione della mobilità rispetto alle precedenti. La mancanza di uno *scheduling* globale, tuttavia, rende le soluzioni basate su contesa prone a fenomeni quali collisioni e *idle listening* non permettendo, infatti, di avere informazioni precise sullo stato globale della rete.
Nel seguito di questo capitolo verranno proposti alcuni protocolli che rappresentano lo stato dell'arte relativamente alle categorie sopra descritte.

Per entrambe le categorie sopra descritte, esiste la possibilità di implementare protocolli che, anziché sfruttare un singolo canale di comunicazione, sono in grado di utilizzare parallelamente più canali. Questa capacità, pur introducendo ulteriori complicazioni alla progettazione di protocolli siffatti, permette di aumentare il *throughput* globale della rete sfruttando la possibilità di avviare più comunicazioni in parallelo.

In questo lavoro l'attenzione è stata posta sulle tecniche multifrequenza applicate alla realizzazione di un protocolo appartenente alla seconda delle categorie sopra elencate.

3.1 Protocolli basati su *scheduling*

Questi protocolli sono particolarmente adatti nel caso in cui la rete sia soggetta ad alti livelli di traffico fortemente periodico. In genere sfruttano tecniche TDMA (*Time Division Multiple Access*) e FDMA (*Frequency Division Multiple Access*) allo scopo di ottenere uno *scheduling* ottimo nello spazio definito da tempo/frequenza utilizzando a tal fine la capacità dei *transceiver* di modificare a *run time* la propria frequenza portante.
Tre sono le principali tecniche usate per organizzare l'attività dei nodi in modo

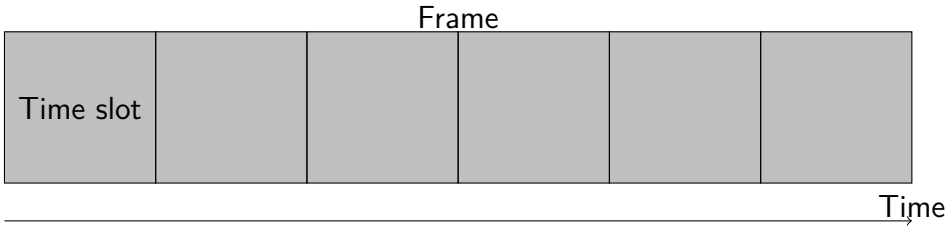


Figura 3.2: Ripartizione degli istanti temporali nel TDMA

efficiente:

- *schedule* dei collegamenti: generalmente il tempo viene suddiviso in frame e ciasun *frame* viene a sua volta suddiviso in *slot*. Ad ogni coppia di nodi connessi (in grado cioè di instaurare una comunicazione punto punto) viene

assegnato uno *slot* e una frequenza eliminando in tal modo le collisioni e limitando fortemente *overhearing* e *idle listening* al costo, però, di una fortemente ridotta flessibilità della rete.

- *schedule* dei trasmettitori: a ciascun nodo viene assegnato uno *slot* temporale durante il quale è abilitato a trasmettere. Con questo approccio all'inizio di ogni slot tutti i nodi non abilitati a trasmettere devono essere in ricezione e, nel caso in cui siano implementate tecniche di *header filtering*, possono, eventualmente, spegnere il *transceiver* per la durata dello slot, una volta assicuratisi di non essere i destinatari del pacchetto in corso di trasmissione.
- *schedule* dei ricevitori: questa tecnica è il caso duale della precedente. In questo caso ad ogni nodo viene assegnato uno slot durante il quale esso può ricevere, i nodi che desiderano trasmettere informazioni al dispositivo abilitato alla ricezione devono contendersi l'uso del canale.

Unendo le tecniche di TDMA e FDMA il tempo viene dapprima suddiviso in slot come in figura 3.2, in seguito ciascuno slot viene moltiplicato per il numero di frequenze di trasmissione disponibili ottenendo una struttura che può essere graficamente rappresentata come in figura 3.3

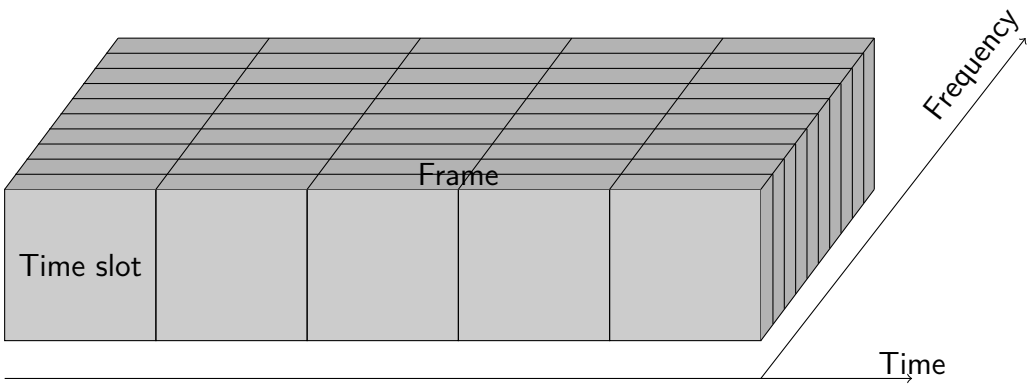


Figura 3.3: Ripartizione degli istanti temporali nel caso di combinazione di TDMA e FDMA

3.1.1 Arisha

Il protocollo Arisha [Doh10] prevede la presenza di un *network coordinator* (generalmente identificato nel nodo *sink*) il quale, acquisendo informazioni topologiche (in particolare l'esistenza di *link* tra coppie di nodi) dai dispositivi attivi, crea un albero (con radice il *network coordinator*) relativo alla rete. Una volta acquisite tali informazioni ad ogni nodo viene assegnato uno *slot* secondo due possibili algoritmi:

- *Breadth first search*.

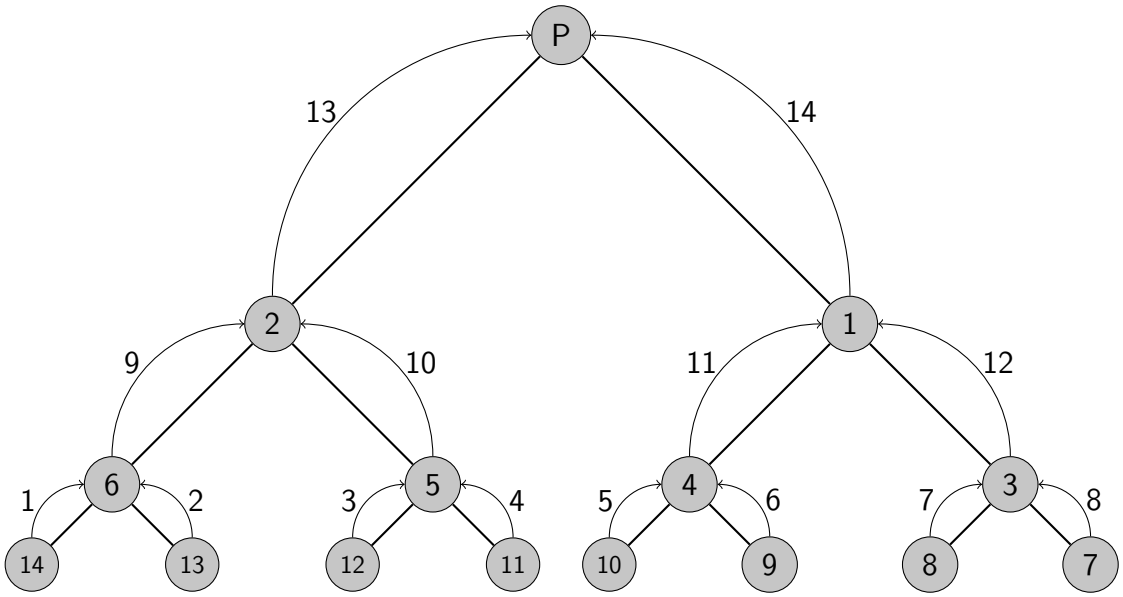


Figura 3.4: Esempio di *breadth first*: le frecce numerate indicano l'ordine di assegnazione degli slot.

- *Depth first search*.

Nel primo caso l'assegnazione degli *slot* inizia dalle foglie: a nodi aventi il medesimo padre vengono assegnati *slot* consecutivi, una volta terminate le foglie la procedura viene nuovamente applicata in modo ricorsivo ai nodi padre fino a giungere al *sink*. Tale politica di assegnazione permette di minimizzare il numero di volte in cui il *transceiver* di un nodo appartenente a un livello interno dell'albero deve commutare il proprio stato Tuttavia tende ad aumentare, nel contempo, il ritardo a cui sono soggetti i pacchetti.
Nel secondo caso, invece, l'assegnazione degli *slot* inizia da un nodo foglia per poi proseguire, secondo legami di parentela diretta (due nodi sono legati da parentela diretta quando nel grafo che rappresenta la connettività della rete esiste un arco che li unisce), verso la radice. Una volta terminata, la procedura ricomincia a partire dalla foglia immediatamente successiva a quella appena processata. Questa politica, pur riducendo il ritardo a cui sono soggetti i pacchetti, aumenta il numero di commutazioni di stato dei *trasciever* dei nodi interni dell'albero.
Questo protocollo soffre di una ridotta scalabilità dal momento che non vi è riuso degli slot né nel caso che, una volta assegnati, risultino inutilizzati. Inoltre non viene tenuto conto del raggio di trasmissione dei dispositivi, infatti una volta assegnato uno slot ad un arco del grafo che rappresenta la rete, questo non viene più riutilizzato anche se un'altra coppia di nodi fuori dal raggio di interferenza dei primi due potrebbe comunque utilizzare lo slot in questione. La

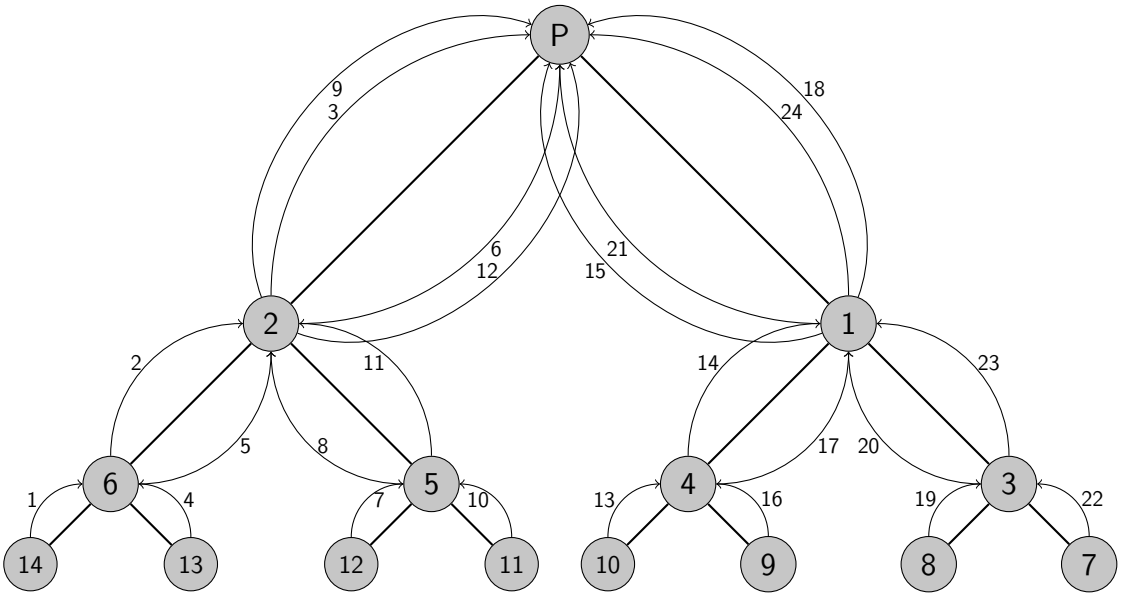


Figura 3.5: Esempio di *depth first*: le frecce numerate indicano l'ordine di assegnazione degli slot.

sua natura intrinsecamente centralizzata, inoltre, lo rende decisamente inadatto nel caso siano richieste capacità di *fault tolerance*, o di supporto alla mobilità, dal momento che, nel caso di *crash* del *sink* non vi è modo di ripristinare la funzionalità della rete se non inserendo un nuovo *network coordinator*.

3.1.2 PEDAMACS

Questo protocollo (si veda [Doh10]) prevede una fase di setup durante la quale il *sink* raccoglie informazioni sulla topologia della rete e sul traffico presente in essa. Basandosi su tali informazioni il *sink* (che si suppone avere potenza di calcolo e di trasmissione sufficiente) avvia la seconda fase generando uno *schedule* globale e inoltrandolo in *broadcast* a tutti i dispositivi attivi.
La prima fase in cui i nodi trasmettono al *sink* le proprie informazioni locali è basata su una politica CSMA, in seguito i nodi comunicano con il *network coordinator* secondo lo schema TDMA generato durante la seconda fase attraverso una tecnica di *graph coloring* applicata al grafo delle interferenze costruito sulle informazioni raccolte in precedenza.

Rispetto ad Arisha, Pedamacs presuppone che il nodo *sink* sia sufficientemente potente da raggiungere tutti i nodi della rete con un solo *hop*, inoltre, il *pattern* di trasmissione è totalmente *collision free* dal momento che esso vie-

ne calcolato in modo centralizzato dal coordinatore dopo che quest'ultimo ha acquisito informazioni su tutti i componenti della rete.

Il principale svantaggio di PEDAMACS è che il pattern di trasmissione è sempre di tipo *convergecast* e, per come come questo protocollo è strutturato, non ne sono supportati altri. Inoltre, il vincolo che il *sink* sia sufficientemente potente da avere un raggio di trasmissione sufficiente a comprendere tutti i nodi della rete non è spesso rispettato in situazioni reali.

3.1.3 SMACS

SMACS [Doh10] è un protocollo localizzato e distribuito, non prevede quindi la necessità di un *network coordinator*. Il suo funzionamento si articola in due fasi:

- *Neighbour discovery*;
- *Channel assignment*.

Durante la fase di *neighbour discovery* un nodo pone il proprio *transceiver* in ricezione e aspetta di ricevere *invitation packets* provenienti da altri dispositivi, nel caso questa eventualità non si verifichi, il nodo inizia a mandare lui stesso *invitation packets*. Durante questo processo i nodi, al fine di ottenere maggior risparmio energetico, spengono e accendono in modo casuale il radiotrasmettitore (in questo modo, tuttavia, esiste una probabilità non nulla che due nodi non si incontrino mai pur essendo vicini). Ad ogni nodo scoperto viene assegnato un canale e vengono definiti *slot* di ricezione e trasmissione.
Nonostante la sua semplicità, questo metodo soffre di vari problemi quali:

- Elevato consumo energetico nella fase di *neighbour discovery*;
- Basso grado di connettività della rete che pone serie difficoltà nel creare instradamenti ottimali per i pacchetti a livello di *routing*.

3.1.4 EMACs

In EMACs vengono definite tre categorie di nodi:

- Attivi;
- Passivi;
- Dormienti.

I nodi attivi possono partecipare a tutte le operazioni in corso nella rete (e in qualche modo sono assimilabili ai *full function device* previsti da IEEE 802.15.4), mentre i nodi passivi, invece, possono solamente comunicare con un nodo attivo.

Solo i nodi appartenenti alle prime due categorie vengono coinvolti nelle comunicazioni. I nodi dormienti, invece, sono nodi che hanno esaurito le batterie o le stanno ricaricando.

I nodi attivi formano un dominating set che funge da backbone per la rete

Il tempo viene suddiviso in *frame* a loro volta suddivisi in *slot*, ogni slot prevede tre sezioni:

- CR: *Communication Request*;
- TC: *Topology Control*;
- DATA.

Ad ogni nodo è assegnato uno *slot*, durante tale intervallo di tempo sono autorizzati a comunicare solo quei nodi che hanno intenzione di avviare una trasmissione verso l'assegnatario dello slot in questione. Nella prima fase di CR il proprietario dello *slot* corrente ascolta il canale in attesa di richieste di comunicazione, durante la seconda fase invia eventuali *acknowledgment* o informazioni di controllo. I nodi passivi devono essere in collegamento diretto con almeno un nodo attivo al quale possono trasmettere dati annunciandosi preventivamente durante la fase di CR e trasmettendo il pacchetto durante la fase DATA. Le comunicazioni da un nodo attivo a un nodo passivo, invece, vengono annunciate durante la fase TC e portate a termine sempre nella fase DATA.

Per assicurare la connessione della rete, i nodi attivi devono formare un *dominating set* in modo che, per essere connesso alla rete, un nodo passivo debba conoscere almeno un nodo attivo.

Il principale inconveniente di questo protocollo deriva dal fatto che i nodi attivi devono necessariamente mantenere in stato di attività il proprio *transceiver*, componente che, come già accennato, è la principale fonte di consumo energetico.

3.2 Protocolli basati su *preamble sampling*

I protocolli appartenenti a questa categoria non prevedono forme di arbitraggio del mezzo di comunicazione e generalmente si basano su tecniche derivanti dai noti algoritmi CSMA e CSMA-CA.

I nodi che implementano tali politiche sfruttano *duty cycle* (cioè il rapporto tra tempo in cui il *transceiver* è acceso e tempo in cui invece è spento) relativamente bassi (valori piuttosto comuni vanno dal 1% al 10%) senza che vi sia alcun tipo di sincronizzazione tra i *clock* dei dispositivi. Qualora un nodo volesse trasmettere

(si veda la figura 3.6) un pacchetto, esso dovrà inviare un preambolo della durata pari o superiore al periodo di inattività previsto dal *duty cycle* del dispositivo di destinazione in modo da avere la certezza di notificare con successo la volontà di iniziare una trasmissione.

Questa categoria di protocolli si adatta a reti con mobilità e nel caso in cui il traffico non abbia forti periodicità, tuttavia la necessità di trasmettere preamboli lunghi quanto la durata del periodo di *sleep* del destinatario comporta principalmente due problemi:

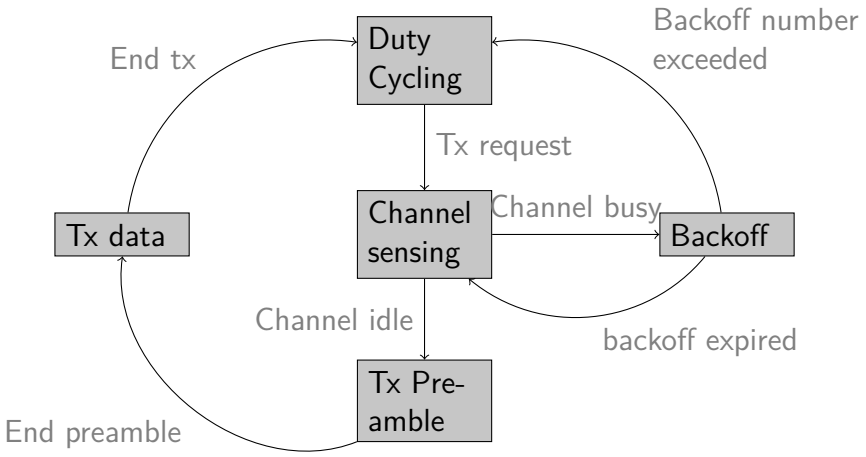


Figura 3.6: *State machine* di un trasmettitore che implementa un protocollo basato su *preamble sampling*

Un nodo che non sta servendo nessuna richiesta resta impegnato ad eseguire il proprio duty cycle. Quando dal layer superiore riceve una Tx Request, il nodo blocca il duty cycle ed esegue la procedura di clear channel assessment. Se il canale risulta libero si comincia a trasmettere un preambolo necessario ad agganciare la ricezione del destinatario, una volta terminato il preambolo i dati vengono inviati e al termine della comunicazione gli agenti di quest'ultima possono tornare ad eseguire i propri duty cycle

- Aumento del costo delle collisioni: infatti, nel caso avvenga una collisione durante la trasmissione di un pacchetto, non si avrà uno spreco di energia relativo esclusivamente al pacchetto soggetto alla collisione, ma anche al preambolo inviato in precedenza per svegliare il destinatario. Le collisioni, dunque, saranno tanto più costose quanto più lungo sarà il preambolo necessario per iniziare la trasmissione;

- Necessità di mantenere *sleep time* relativamente ridotti: dal momento che la lunghezza del preambolo che precede ogni pacchetto è direttamente proporzionale alla durata del tempo di *sleep* del destinatario è necessario trovare un compromesso tra la possibilità di risparmio energetico ottenuto utilizzando *duty cycle* sempre più bassi, e il consumo relativo alla necessità di trasmettere preamboli sempre più lunghi all'aumentare del tempo di inattività del *transceiver* del destinatario.

3.2.1 LPL [PHC04]

Low Power Listening [PHC04] realizza i concetti basilari dei protocolli basati su *preamble sampling*.
I nodi operano in totale autonomia alternando stati di attività a stati di inattività secondo un *duty cycle* fissato al momento del *deployment* della rete.
Al momento di trasmettere un nodo esegue un *sampling* del canale sfruttando il pin CCA di cui à dotata la maggior parte dei *transceiver*, nel caso non venga rilevata attività al di sopra di una soglia predefinita inizia la trasmissione del preambolo lungo quanto il tempo di *sleep* previsto dal *duty cycle* (come si può vedere in figura 3.7).

Quando un nodo che esegue il proprio duty cycle entra in stato di ricezione e si sincronizza su un preambolo, interrompe il duty cycle e si prepara a ricevere i dati a lui destinati. Terminata la trasmissione, il duty cycle viene ripristinato

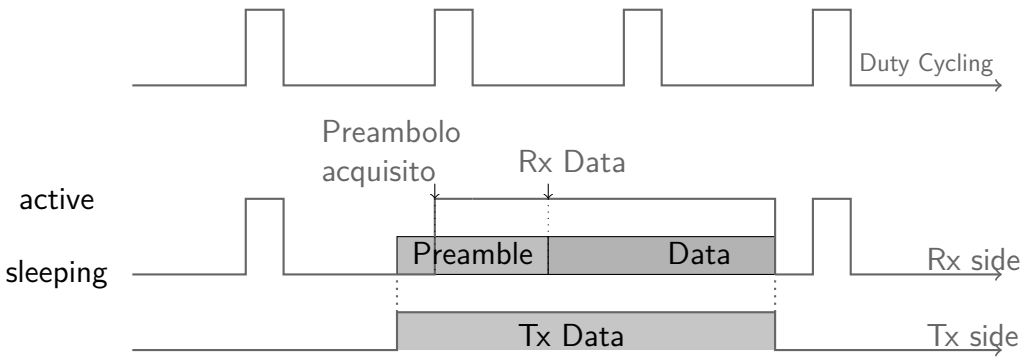


Figura 3.7: *Preamble sampling* come implementato in LPL

Questo protocollo, pur ponendo le basi per le successive tecniche che sfruttano il *preamble sampling*, soffre, senza possibilità di mitigarli, dei due problemi descritti in precedenza (si veda figura 3.7). Verranno ora descritti alcuni dei protocolli più diffusi che sono stati sviluppati sulla base di LPL e che ne migliorano le prestazioni introducendo accorgimenti atti a mitigare l'influenza dei due problemi congeniti a questa categoria di protocolli.

3.2.2 BMAC [PHC04]

BMAC [PHC04] sfrutta le stesse idee che hanno portato alla definizione di *Low Power Listening* del quale, però, migliora il processo di *Clear Channel Assessment*.
A differenza di LPL in BMAC il CCA non viene fatto per mezzo del *tresholding*, ma su base statistica. Durante il periodo di *sampling* del canale vengono rilevate

tutte le anomalie al di sotto della soglia di rumore, nel caso tali anomalie siano presenti il canale viene dichiarato libero dal momento che in un segnale regolare la probabilità di rilevare componenti al di sotto della soglia di rumore è bassa. Se al contrario non vengono rilevate anomalie del tipo sopra descritto il canale viene dichiarato occupato e si procede a un *backoff* come specificato nel CSMA definito nello standard IEEE 802.15.4.

3.2.3 WISEMAC [EHD04]

WISEMAC [EHD04] migliora le prestazioni di BMAC dando ai nodi la possibilità di raccogliere informazioni sui *wake up time* dei vicini e permettendo, in questo modo, di tarare la lunghezza del preambolo sulla stima dell'istante di risveglio del nodo destinatario.
Le informazioni sui tempi di risveglio vengono trasmesse come campo aggiuntivo dei pacchetti di *acknowledgment* e conservate in una *neighbour table*, esse devono comunque essere aggiornate periodicamente a causa delle derive dei *clock* dei singoli trasmettitori.
Nel caso la sincronizzazione venga persa o non sia disponibile l'informazione relativa a un vicino, durante la trasmissione viene utilizzato un preambolo di lunghezza massima tornando di fatto al caso di LPL.

3.2.4 XMAC [BYAH06]

Questo protocollo, rispetto a WISEMAC, riduce ulteriormente l'energia consumata nella trasmissione del preambolo,

La figura 3.8 mostra come la necessità di trasmettere per intero il preambolo porti a un possibile consistente spreco di energia sia in ricezione che in trasmissione

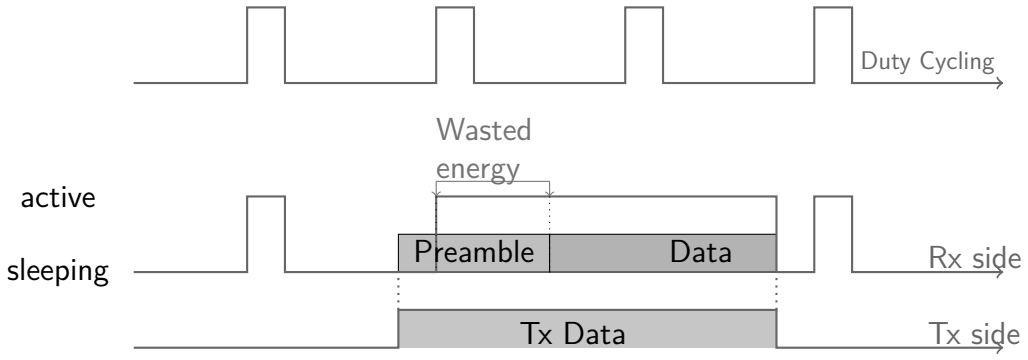


Figura 3.8: Inefficienza dovuta alla trasmissione dell'intero preambolo in LPL

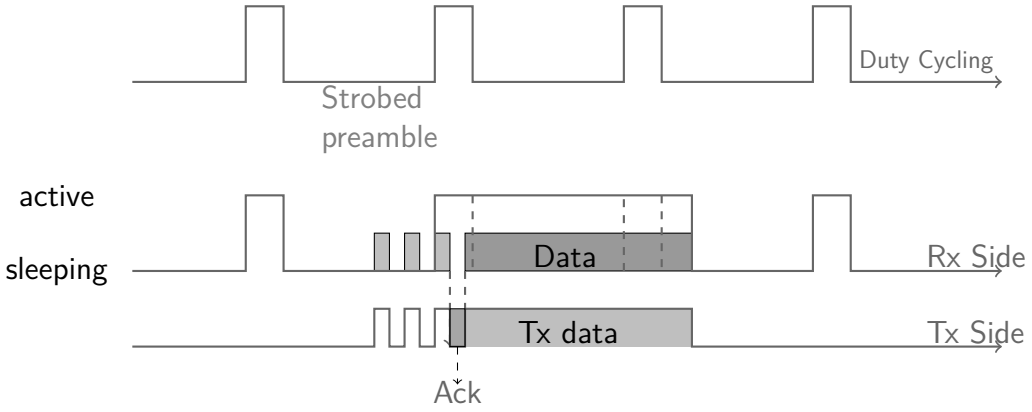


Figura 3.9: Utilizzo di *strobed preamble* in XMAC

La figura **3.9** mostra come in XMAC lo spreco di energia proprio di LPL e BMAC venga ridotto per mezzo di strobed preamble

quest'ultimo, infatti, viene spezzato in una sequenza di *strobed preambles* in ognuno dei quali vengono inseriti gli indirizzi di sorgente e destinazione. In questo modo, qualora un nodo diverso dal destinatario ricevesse uno di questi preamboli potrebbe agevolmente trascurarlo e tornare ad eseguire il proprio *duty cycle*, il destinatario invece, una volta ricevuto correttamente uno di questi preamboli potrà immediatamente inviare un *acknowledgment* al nodo sorgente permettendogli in questo modo di cominciare la trasmissione dei dati.
Tra i protocolli finora descritti basati su tecniche di *preamble sampling*, XMAC si è rivelato il più performante dal punto di vista energetico. Esso è inoltre apprezzato per la sua caratteristica di essere particolarmente adatto all'implementazione in radio basate su pacchetti (anziché su *stream* di bit).

3.3 Protocolli multi portante

Sebbene i letteratura siano presenti molti protocolli di accesso al mezzo dedicati alle reti di sensori, la maggior parte di questi fa affidamento su un singolo canale e pone la propria attenzione sull'ottimizzazione dell'arbitraggio di un'unica risorsa.

In letteratura, la presenza di protocolli multi portante non è consistente come ci si aspetterebbe visti i vantaggi che politiche di questo tipo potrebbero offrire

Lo standard IEEE 802.15.4, tuttavia, mette a disposizione sedici canali logici dedicati alla tecnologia LR-WPAN come ripartizione della banda ISM a 2.4 GHz.

La possibilità di sfruttare nella sua interezza lo spettro disponibile è di grande attrattiva dal momento che potrebbe permettere di raggiungere *throughput* consistentemente più elevati rispetto agli standard attuali., ma l'introduzione della dimensione frequenziale, oltre a quella temporale, introduce ulteriore complessità al già non banale compito di progettare protocolli di accesso al mezzo efficienti.

In questa tesi si è deciso di indagare il campo dell'accesso al mezzo multi-frequenza al fine di sfruttare i vantaggi di cui sopra, a tal proposito si ritiene opportuno presentare alcuni casi presenti in letteratura di protocolli MAC multi frequenza in modo da rendere noti, tramite esempi, i principali problemi in cui si incorre durante la progettazione di questo tipo politiche.

3.3.1 TF-MAC [JD07]

In *Time Frequency MAC* (TFMAC) il tempo viene diviso in *slot* di lunghezza prefissata, ciascuno *slot* è suddiviso in due parti:

1. *Contention access period*: utilizzato per lo scambio di messaggi necessari al mantenimento del protocollo;

2. *Contention free period*: utilizzato per lo scambio di dati.

Durante il *contention access period* i nodi devono ascoltare una frequenza prestabilita di *rendez vous* su cui tutte le trasmissione di servizio devono avvenire. L'accesso a questo canale è basato su contesa e viene utilizzata una tecnica simile alla *distributed coordination function*, propria dello standard IEEE 802.11, che prevede lo scambio di messaggi detti *Request To Send* (RTS) per avviare una comunicazione e *Clear To Send* (CTS) affinché il destinatario possa comunicare al mittente il proprio consenso all'avvio della comunicazione.

Ogni *contention free period* viene poi, a sua volta, suddiviso in N_t *time slot*. Ciascun *time slot* può essere di tre tipi:

1. T: slot di trasmissione;

2. R: slot di ricezione;

3. I: slot idle.

Ogni nodo mantiene una tabella locale in cui ad ogni *time slot* viene assegnata una tripla: (i, t, f) dove:

1. i: rappresenta l'indice dello slot al'interno del *contention free period*;

2. t: rappresenta il tipo assegnato allo slot i;

3. f: rappresenta la frequenza su cui il nodo che mantiene la tabella deve trasmettere durante lo slot i.

Ad ogni nodo vengono assegnati tanti slot di trasmissione quante sono le frequenze disponibili per la rete, mentre ad esso viene assegnato una sola frequenza di trasmissione.

TFMAC consiste in due aspetti principali: il primo è l'assegnazione delle frequenze di ricezione, durante questa fase ciascun nodo sceglie in modo casuale una delle frequenze disponibili e la comunica in broadcast, sul canale di *rendez vous* a tutti i vicini; il secondo aspetto, invece, è quello dell'accesso al mezzo.
Al termine della fase di selezione delle frequenze di ricezione, al fine di garantire un accesso al mezzo privo di collisioni, i nodi della rete provvedono al *setup* delle proprie tabelle. Durante questa fase ciascun nodo sceglie in modo aleatorio se essere attivo (e quindi partecipare alle successive comunicazioni) o passivo (e quindi restare escluso da tutto il traffico successivo alla scelta, ma essere obbligato all'ascolto del canale di controllo al fine di acquisire informazioni relative ai propri vicini). Ciascun nodo trasmette in broadcast la richiesta di ottenere dai propri vicini le rispettive tabelle, quando queste vengono ricevute, il nodo che aveva mandato la richiesta provvede a determinare quali slot sono liberi da collisioni.

Il principale problema di questo protocollo è la presenza di un canale di controllo, il cui accesso è regolamentato da politiche basate sulla contesa, che rappresenta, nel caso di reti dense, un notevole collo di bottiglia, inoltre il calcolo degli slot liberi da collisioni che ciascun nodo deve eseguire per determinare la propria tabella può diventare computazionalmente esoso, specialmente per dispositivi a ridotta capacità di calcolo.

Nonostante i problemi sopra evidenziati, i risultati mostrati in [JD07] dimostrano come l'utilizzo di più canali in parallelo possa giovare notevolmente al *throughput* di una rete.

3.3.2 YMAC [KSC08]

YMAC è un protocollo multi frequenza basato sulla tecnica TDMA, ad ogni nodo vengono assegnati, per mezzo di un algoritmo distribuito, uno slot di ricezione e uno di trasmissione. Questi due slot devono risultare assegnati ad un solo nodo nel raggio di 2 *hop* di trasmissione al fine di garantire l'assenza di collisioni.
Le trasmissioni vengono avviate in un canale di controllo prefissato e comune per tutti i nodi, dopo che il primo pacchetto è stato inviato e il destinatario ha comunicato l'avvenuta ricezione, i due nodi coinvolti nella comunicazione provvedono ad impostare la propria frequenza portante su quella centrale di un canale concordato tramite lo scambio dei primi due pacchetti.

La necessità di avviare tutte le comunicazioni partendo dallo stesso canale di controllo può, in caso di traffico intenso, comportare forti ritardi nelle trasmis-

sioni, questo problema è solo mitigato dal successivo cambio di canale.

Come detto per il protocollo TFMAC, tuttavia, il throughput che i risultati presentati in [KSC08] evidenziano, dimostra come la strada della progettazione di protocolli di accesso al mezzo multi portante sia interessante.

Capitolo 4

Network Simulator 3: NS-3

NS-3 è un simulatore di rete a eventi. Esso è nato con lo scopo di unificare i processi di creazione di nuovi moduli e implementazione delle relative simulazioni che nel precedente NS-2 erano nettamente separati, visti, però, i numerosi *feedback* positivi ricevuti dalla comunità di utilizzatori di NS-2 e il sempre crescente numero di sviluppatori che decisero di contribuire al progetto, NS-3, da possibile estensione di NS-2, diventò un nuovo - e sempre più apprezzato - simulatore di rete. La sua modularità, il fatto di essere interamente realizzato in C++ (a differenza di NS-2 che invece, pur essendo il suo *core* parimenti implementato in C++, prevede, per quanto riguarda lo *scripting* della topologia relativa alle simulazioni desiderate dall'utente, l'utilizzo del linguaggio OTcl) e l'ampia e attiva comunità di sviluppatori stanno rendendo il progetto NS-3 uno dei simulatori di rete più diffusi e apprezzati tra gli specialisti del settore.

La scelta è ricaduta sul simulatore oggetto di questo capitolo principalmente a causa della sua modularità e flessibilità. Il fatto di essere open source e la vasta comunità di utenti e sviluppatori impegnati nel progetto NS-3 non ha fatto che incentivare all'uso di questo framework

4.1 Da NS-2 a NS-3

NS-2 è un simulatore di reti ad eventi sviluppato dalla *University of California, Berkeley* (UC Berkeley) e mantenuto dalla *University of Southern California* di Los Angeles.
Gli scopi principali di questo progetto erano (e, nonstante la diffusione del più moderno NS-3, ancora sono):

- supportare la ricerca e la didattica nell'ambito delle reti di calcolatori, in particolare favorendo le possibilità di:
  - sviluppo di nuovi protocolli di rete;
  - confronto tra protocolli;
  - sviluppo di nuove architetture di rete.

- fornire un ambiente di sviluppo collaborativo che sia *open source* e liberamente redistribuibile e che, nel contempo, permetta di aumentare la fiducia nei risultati provenienti da simulazioni piuttosto che da reali prove sul campo (spesso di più ardua realizzazione sia dal punto di vista implementativo che economico).

Seguendo le linee guida tracciate dagli obiettivi sopra esposti, NS-2 è stato progettato in modo da scindere chiaramente le necessità di efficienza computazionale e di semplicità nell'impostazione della struttura delle simulazioni. Per quanto riguarda la prima necessità, la soluzione adottata è stata quella di implementare il *core* del simulatore e i vari protocolli di comunicazione in C++, per quanto riguarda, invece, il secondo requisito si è scelto il liguaggio OTcl (estensione a oggetti del linguaggio Tcl/Tk sviluppata presso il prestigioso *Massachusetts Institute of Technology*) la cui stesura risulta più rapida e meno impegnativa rispetto al C++ al costo, tuttavia, di prestazioni meno elevate in termini di velocità di esecuzione.
NS-3, nato come *fork* di NS-2 e diventato quasi subito indipendente, mantiene le finalità del suo predecessore pur introducendo significativi cambiamenti allo scopo di ottenere migliori prestazioni rendendo, allo stesso tempo, più agevole la realizzazione di *script* di simulazione. Al fine di ottenere questi risultati, la scissione tra *core* e *scripting* in termini di linguaggi di programmazione è stata eliminata in favore dell'uso globale del linguaggio C++, tuttavia, per dare comunque la possibilità di implementare agevolmente simulazioni con un linguaggio di *scripting*, viene fornito un *frontend* per l'utilizzo del *core* NS-3 da parte di applicazioni implementate in Python.

4.2 Pseudo Random Number Generator

Il generatore di numeri pseudocasuali utilizzato in NS-3 è il *Combined Multiple-Recursive Generator* (CMRG) MRG32k3a proposto da Pierre L'Ecuyer in [LSCK02]. Questo generatore ha il pregio di permettere l'utilizzo di $1.8 * 10^{19}$ flussi indipendenti di numeri casuali, ognuno dei quali composto da $2.3 * 10^{15}$ sotto-flussi indipendenti di periodo $7.6 * 10^{22}$ ottenendo in questo modo un periodo complessivo pari a $3.1 * 10^{57}$.
L'interfaccia fornita per l'utilizzo del generatore di numeri pseudocasuali è definita dall classe `ns3::RandomVariable` la quale permette di creare oggetti che rappresentano variabili aleatorie con distribuzione determinata dall'utente all'atto della creazione dell'oggetto. Ad ogni oggetto di tipo `RandomVariable` viene associato uno dei flussi indipenti di numeri casuali e, per ciascuno di questi ultimi, può essere impostato il sotto-flusso desiderato.

Grazie all'architettura sopra descritta ciascuna simulazione può utilizzare contemporaneamente $1.8 * 10^{19}$ variabili aleatorie indipendenti, ciascuna con una propria distribuzione definita dall'utente, e per ciascuna simulazione si possono eseguire $2.3 * 10^{15}$ *run* indipendenti.
Le *API* fornite dalla classe `ns3::RandomVariable` sono:

- `double GetValue(void) const;` che ritorna un valore in virgola mobile a doppia precisione secondo la particolare distribuzione di probabilità implementata;
- `uint32_t GetInteger(void) const;` che ritorna, invece, un valore intero eseguendo un *cast* ad *integer* del valore ritornato dalla precedente funzione.

Tutte le variabili aleatorie che realizzano particolari distribuzioni di probabilità devono derivare dalla classe `RandomVariable` ed implementare opportunamente le succitate *API*.
Le principali distribuzioni implementate in NS-3 sono:

- Uniforme: `class UniformVariable : RandomVariable`;
- Esponenziale: `class ExponentialVariable : RandomVariable`;
- Di Pareto: `class ParetoVariable : RandomVariable`;
- Normale: `class NormalVariable : RandomVariable`;
- LogNormale: `class LogNormalVariable : RandomVariable`.

Inoltre viene lasciata libertà agli sviluppatori di realizzare qualunque distribuzione ritengano necessaria e di integrarla nel codice del simulatore.
La possibilità di realizzare un grande numero di esecuzioni indipendenti per ogni simulazione è di fondamentale importanza nelle campagne di raccolta dati con scopo la validazione di modelli teorici. La disponibilità di una grande quantità di dati provenienti da iterazioni indipendenti, infatti, permette di aumentare la robustezza dei risultati nel processo di valutazione di un modello teorico e di ottenere stime con appurata validità statistica.

4.3 Object Model

NS-3 prevede tre classi di base:

- `class Object`;

- `class ObjectBase`;
- `class SimpleRefCount`.

Sebbene non sia necessario che le classi definite dagli utenti derivino da una delle tre succitate, quelle che lo fanno acquisiscono particolari capacità che ne agevolano la gestione all'interno del *framework* di NS-3.
Le classi che derivano dalla classe `Object` acquisiscono le seguenti proprietà:

- il sistema di tipizzazione e di attributi proprio di NS-3;
- un sistema di aggregazione degli oggetti dello stesso tipo;
- un sistema di *smart pointer* associato al conteggio dei riferimenti attivi.

Le classi che derivano da `ObjectBase` acquisiscono solo le prime due proprietà tra quelle sopra elencate, mantre le classi che derivano da `SimpleRefCount` possono utilizzare solamente il sistema di *smart pointer*.

4.3.1 Smart pointer

Il sistema di *smart pointer* e *reference counting* permette allo sviluppatore di non dover gestire esplicitamente il complesso processo di liberazione della memoria allocata e non più referenziata (operazioni queste che, se non svolte consistentemente, sono fonte di errori a *run time* nel migliore dei casi, mentre nel peggiore dei casi causano *memory leak* di difficile individuazione e responsabili spesso di *performance* scadenti)

In C++, Ogni oggetto mantiene una variabile di stato che gestisce il conteggio dei riferimenti attivi verso l'oggetto stesso. Ogni qualvolta si ottenga un puntatore verso un determinato oggetto, il conteggio dei riferimenti relativo all'elemento referenziato viene incrementato tramite la chiamata al metodo `Ref()`, il compito di notificare all'oggetto il fatto che un puntatore non è più utilizzato viene, invece, demandato all'utente attraverso l'esplicita chiamata al metodo `Unref()`.
Appurato che questa tecnica è decisamente prona ad errori relativi alla gestione della memoria (soprattutto nel caso di progetti di medie o grandi dimensioni), in NS-3 è stata implementata la classe `Ptr` che, facendo da *wrapper* al puntatore desiderato, gestisce automaticamente le chiamate a `Ref()` e `Unref()`. La sintassi per la creazione di oggetti attraverso il sistema di *smart pointer* è:

```
Ptr<Type> pointerName = CreateObject<Type>();
```

Nel caso in cui la classe che definisce l'oggetto di tipo `Type` derivi da `Object`. Nel caso in cui, invece, l'oggetto desiderato derivi dalla classe `SimpleRefCount`, la sintassi corretta è:

```
Ptr<Type> pointerName = Create<Type>();
```

In entrambi i casi, i parametri delle funzioni `CreateObject()` e `Create()` sono gli stessi che andrebbero forniti al costruttore dell'oggetto di tipo `Type` nel caso si volesse ottenere esplicitamente un puntatore ad esso.

4.4 Callbacks

Una *callback* nell'accezione più generale del termine altro non è che un puntatore a funzione utilizzato quando si ha la necessità, in seguito ad un evento, di chiamare una funzione della quale si conoscono i parametri (in particolare la firma), ma della quale non si conosce l'oggetto che la implementa.
Nella cossiddetta *event driven programming* l'utilizzo di *callbacks* è una pratica estremamente diffusa e finanche, in alcuni casi, necessaria. In particolare nella programmazione di protocolli di rete, ambito in cui la modularità è una caratteristica essenziale per il buon funzionamento di un sistema, la tecnica delle *callback* è sfruttata intensamente ogni qualvolta un *layer* debba comunicare con quello superiore o inferiore nello *stack* protocollare.
In NS-3 è implementato un sistema che, consistentemente con la filosofia di non lasciare all'utente la gestione diretta dei puntatori, gestisce automaticamente le *callback* fornendo la classe *wrapper* `Callback`.
Attraverso l'utilizzo di questa classe lo sviluppatore non deve gestire esplicitamente la creazione dei vari puntatori a funzione necessari alla realizzazione del sistema di interconnessione tra i vari livelli dello *stack*.
Una *callback* viene inserita nel sistema di NS-3 attraverso la chiamata al metodo:

```
Callback<R>
ns3::MakeCallback (R(T::*memPtr)([...]), OBJ objPtr);
```

dove:

- `R`: tipo di dato ritornato dalla *callback*;
- `T`: classe di appartenenza del metodo da chiamare;
- `*memPtr`: puntatore al metodo che deve essere chiamato;

- `([...])`: eventuali parametri presenti nella firma del metodo;
- `OBJ objPtr`: riferimento all'oggetto specifico del quale è necessario invocare il metodo puntato da `*memPtr`.

una volta creata una *callback*, essa può essere chiamata come un qualunque metodo di una classe C++.

4.5 Integrazione con Python

NS-3 fornisce i *bindings* per Python che consentono a *script* realizzati in questo linguaggio di utilizzare il codice C++ del simulatore. Lo scopo di fornire un'integrazione con il linguaggio Python è duplice:

- permettere agli sviluppatori di realizzare *script* di simulazione interamente in Python;
- permettere la prototipazione di nuovi modelli.

Sebbene le premesse lascino ben sperare nel futuro di questa integrazione che dovrebbe coniugare la potenza del C++ con la flessibilità propria del Python, la realizzazione dei *binding* è ancora in corso di sviluppo e, di conseguenza, il supporto alle *API* di NS-3 non è ancora completo. Per questo motivo, al fine di uno studio approfondito dell'integrazione tra i linguaggi sopra menzionati si rimanda il lettore al manuale ufficiale di NS-3.

4.6 Simulare con NS-3

Essendo NS-3 un simulatore discreto, una simulazione realizzata con questo *framenwork* altro non è se non una sequenza di eventi a ognuno dei quali viene associato l'istante temporale in cui esso dovrà essere gestito. Terminata la gestione dell'evento corrente (in particolare la porzione di codice necessaria a gestire l'avvenimento), lo *scheduler* del simulatore avanzerà istantaneamente al nuovo evento registrato e provvederà alla gestione di quest'ultimo.
Quanto detto permette di intuire due questioni di fondamentale importanza per l'utilizzo consapevole di uno strumento di simulazione siffatto:

- esiste una netta separazione tra tempo di simulazione e tempo realmente trascorso durante l'esecuzione della simulazione stessa;

- ogni porzione di codice viene eseguita con tempo di simulazione nullo, di conseguenza ogni qualvolta si desideri rappresentare la durata di un'operazione è necessario specificarne gli istanti di inizio e di fine ed inserirli nel *database* degli eventi proprio della simulazione di modo che il simulatore possa far avanzare il proprio conteggio temporale.

L'inserimento di un evento nel *database* della simulazione si ottiene attraverso la chiamata a:

```
static EventId
Schedule (Time const &time, MEM mem_ptr, OBJ obj, [...]);
```

dove:

- `EventId`: è un *handle* univoco all'*entry* nel *database* di eventi della simulazione. Lo scopo di questo valore di ritorno è dare la possibilità di cancellare l'evento dalla simulazione in un secondo momento (comunque prima che il relativo codice venga eseguito);
- `Time`: la classe `Time` è un *wrapper* per il tipo di dato `int64_t` che permette di rappresentare il tempo di simulazione in nanosecondi. Fornisce, inoltre, metodi per la conversione da nanosecondi ad altra unità di misura temporale:
  - `timeInstance.GetMicroSeconds();`
  - `timeInstance.GetMilliSeconds();`
  - `timeInstance.GetSeconds()`.
- `mem_ptr`: puntatore all'area di memoria contente il codice da eseguire all'avvenire dell'evento, generalmente è un puntatore a funzione;
- `([...])`: eventuali parametri presenti nella firma del metodo;
- `obj`: riferimento alla particolare istanza dell'oggetto responsabile della gestione dell'evento.

Varianti della funzione sopra descritta sono:

```
static EventId
ScheduleNow (MEM mem_ptr, OBJ obj, [...]);
```

che permette lo *scheduling* di un evento all'istante corrente di simulazione e:

```
static void
ScheduleWithContext (
                uint32_t context,
                Time const &time,
                MEM mem_ptr,
                OBJ obj,
                [...]);
```

che permette di specificare un parametro di contesto da associare all'evento.

Una volta eseguito il *setup* della configurazione della simulazione, questa viene avviata tramite la chiamata al metodo:

```
static void Simulator::Run();
```

È poi necessario fermare la simulazione all'istante desiderato:

```
static void Simulator::Stop();
```

oppure

```
static void Simulator::Stop(Time const &time);
```

che permette di specificare in modo esplicito l'istante di terminazione. Infine si deve procedere alla liberazione della memoria ancora allocata tramite la chiamata al metodo:

```
static void Simulator::Destroy();
```

4.7 Il framework Spectrum

Il progetto e la simulazione di un protocollo di accesso al mezzo multi portante, richiedono una rappresentazione precisa dello spettro su cui si effettuano le trasmissioni al fine di poter modellare con accuratezza fenomeni quali l'autointerferenza che affligge trasmissioni contemporanee su canali adiacenti. Nell'ambito del simulatore NS-3, il framework Spectrum è risultato essere il miglior strumento disponibile per la rappresentazione del mezzo trasmissivo.

Il framework di modellazione di fenomeni dipendenti dalla frequenza di trasmissione *Spectrum*, sviluppato da Nicola Bado e Marco Miozzo [BM09], aggiunge al *core* di NS-3 una funzionalità molto importante nello studio dei protocolli di comunicazione per sistemi *wireless*. L'utilizzo di Spectrum permette di modellare un segnale da trasmettere nei termini della sua *Power Spectral Density* (PSD). A questo scopo lo spettro viene suddiviso in un insieme finito di bande di frequenza, entro ciascuna delle quali si considera che i fenomeni dipendenti dalla frequenza agiscano in modo costante.
Più rigorosamente, un modello dello spettro S viene definito come:

$$S = \{B_i\},\ i = 1, \ldots, N,\ B_i = [f_{i,l}, f_{i,h}]$$

dove $f_{i,l}$ e $f_{i,h}$ sono gli estremi (relativamente inferiore e superiore) in frequenza della relativa banda B_i. All'aumentare di N dunque aumenterà la granularità della rappresentazione dello spettro rendendo la simulazione dei processi che, in frequenza, agiscono sul segnale trasmesso sempre più accurata. L'entità di N deve essere decisa dallo sviluppatore sulla base di un compromesso tra complessità computazionale e accuratezza dei risultati desiderati.
Una volta definita la struttura dello spettro, che come si può intuire è implementata tramite un vettore le cui celle rappresentano le singole bande in frequenza e i valori contenuti, invece, rappresentano i livelli di PSD presenti in ciascuna banda, i fenomeni dipendenti dalla frequenza vengono modellati come operazioni algebriche tra livelli di PSD (rappresentati per mezzo di valori in virgola mobile a doppia precisione) applicate a ciascun elemento del vettore rappresentante lo spettro.
Nell'ambito di questo lavoro, il framework Spectrum si è rivelato di fondamentale importanza nella modellazione delle interferenze tra canali adiacenti nella porzione di spettro relativa allo standard IEEE 802.15.4 dovute a trasmissioni contemporanee e, inoltre, nell'implementazione di un protocollo CSMA-CA che realizzi il processo di CCA per mezzo di *energy detection* come previsto dallo standard succitato.

4.8 I moduli IEEE 802.15.4

Lo standard IEEE 802.15.4 non è, al momento, ufficialmente supportato da NS-3, la comunità di sviluppatori, tuttavia, ha avviato diversi progetti atti a realizzare lo *stack* protocollare proprio di questa tecnologia.
In particolare è stata rilasciata di recente una versione non ufficiale, né definitiva, dei primi due livelli previsti dallo standard: il *physical layer* e il *medium access control sublayer*. Questa *release*, seppur chiaramente ancora in fase di sviluppo, si è rivelata un utile supporto al processo di modellazione del comportamento del

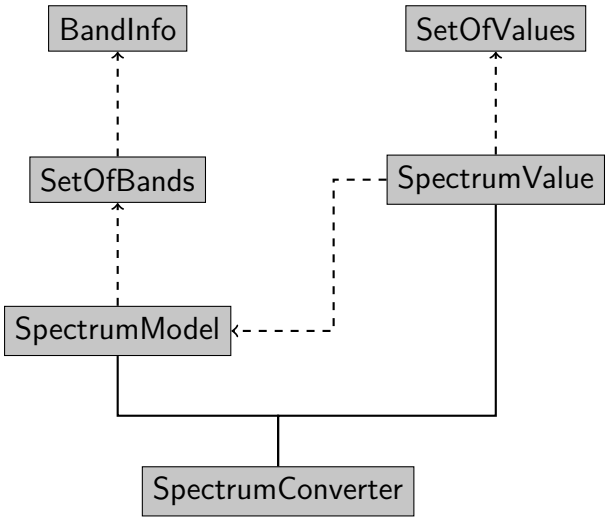


Figura 4.1: Diagramma delle classi del framework Spectrum

livello fisico di un *transceiver* IEEE 802.15.4 ideale.
Verrà ora presentata una descrizione dei moduli che si sono rivelati utili e delle modifiche apportate a questi ultimi con lo scopo di ottenere simulazioni coerenti con le necessità del progetto oggetto di questa tesi.

4.8.1 Lr-wpan-phy

La classe `LrWpanPhy` implementa il comportamento di un radiotrasmettitore ideale che implementa lo standard IEEE 802.15.4. I principali metodi che permettono il controllo dei parametri caratteristici del *transceiver* sono:

```
LrWpanPhy::PlmeSetTRXStateRequest(
        LrWpanPhyEnumeration);

LrWpanPhy::PlmeSetAttributeRequest(
        LrWpanPibAttributeIdentifier id,
        LrWpanPhyPIBAttributes *attribute);
```

I valori di stato previsti per descrivere lo stato del *transceiver* sono contenuti nella definizione:

```
typedef enum
{
        IEEE_802_15_4_PHY_BUSY  = 0x00,
```

```
        IEEE_802_15_4_PHY_BUSY_RX = 0x01,
        IEEE_802_15_4_PHY_BUSY_TX = 0x02,
        IEEE_802_15_4_PHY_FORCE_TRX_OFF = 0x03,
        IEEE_802_15_4_PHY_IDLE = 0x04,
        IEEE_802_15_4_PHY_INVALID_PARAMETER = 0x05,
        IEEE_802_15_4_PHY_RX_ON = 0x06,
        IEEE_802_15_4_PHY_SUCCESS = 0x07,
        IEEE_802_15_4_PHY_TRX_OFF = 0x08,
        IEEE_802_15_4_PHY_TX_ON = 0x09,
        IEEE_802_15_4_PHY_UNSUPPORTED_ATTRIBUTE = 0xa,
        IEEE_802_15_4_PHY_READ_OLNY = 0xb,
        IEEE_802_15_4_PHY_UNSPECIFIED = 0xc
} LrWpanPhyEnumeration;
```

Mentre gli identificatori degli attributi del livello fisico sono così definiti:

```
typedef enum
{
        phyCurrentChannel = 0x00,
        phyChannelsSupported = 0x01,
        phyTransmitPower = 0x02,
        phyCCAMode = 0x03,
        phyCurrentPage = 0x04,
        phyMaxFrameDuration = 0x05,
        phySHRDuration = 0x06,
        phySymbolsPerOctet = 0x07
} LrWpanPibAttributeIdentifier;
```

La struttura che invece contiene i valori attuali associati a ciascun attributo è:

```
typedef struct
{
        uint8_t phyCurrentChannel;
        uint32_t phyChannelsSupported[32];
        uint8_t phyTransmitPower;
        uint8_t phyCCAMode;
        uint32_t phyCurrentPage;
        uint32_t phyMaxFrameDuration;
        uint32_t phySHRDuration;
        double phySymbolsPerOctet;
} LrWpanPhyPIBAttributes;
```

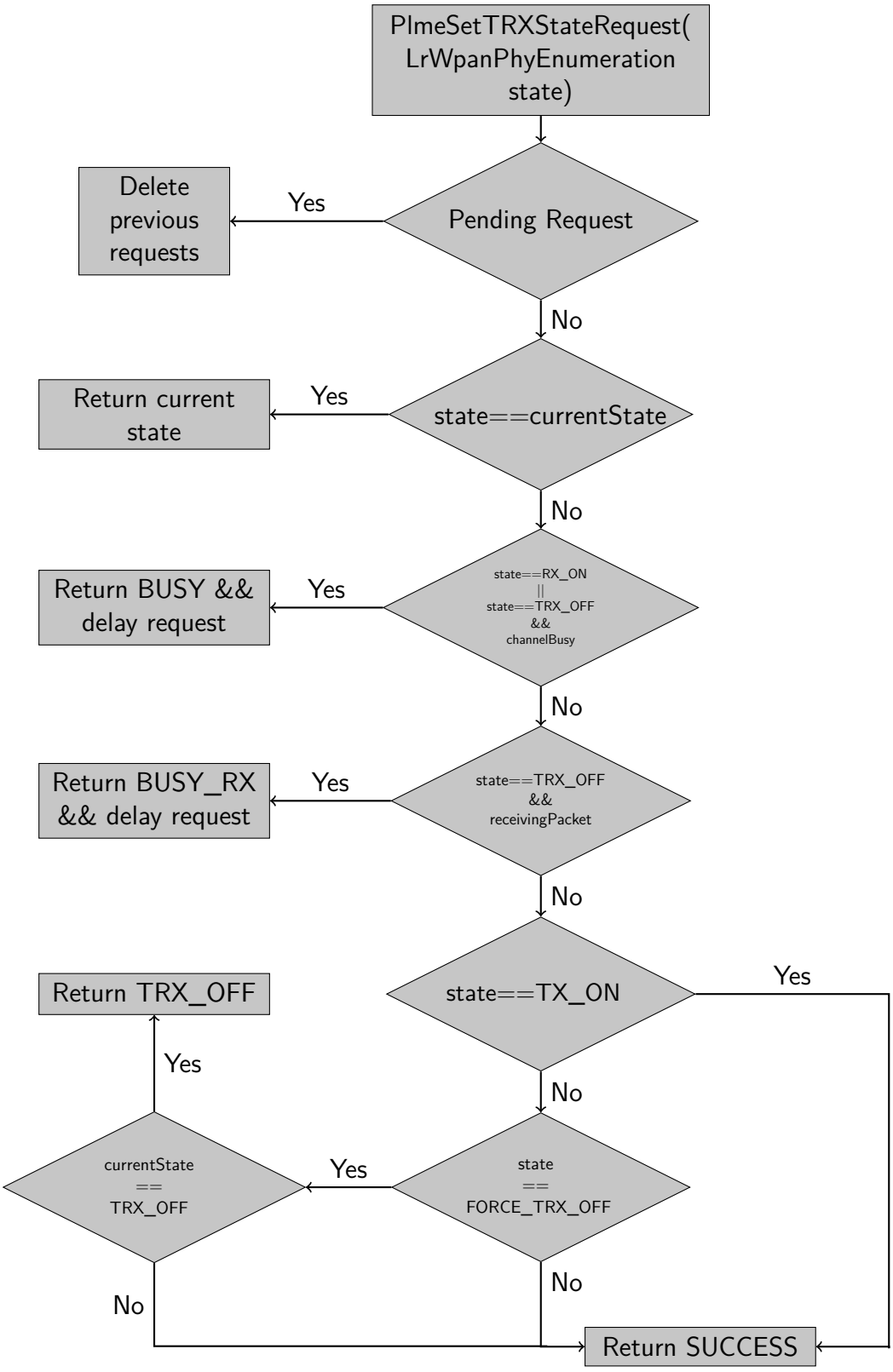


Figura 4.2: Diagramma di flusso per il metodo PlmeSetTRXStateRequest

Coerentemente con quanto prescritto dallo standard, la funzione che aggiorna lo stato operativo del radiotrasmettitore ha il comportamento descritto in figura 4.2. Il metodo che, invece, permette di impostare i parametri che determinano il comportamento del livello fisico ha il comportamento descritto in figura 4.3.
La classe in oggetto, implementa inoltre la funzione di *Clear Channel Assessment*, sempre coerentemente con le prescrizioni dello standard, tuttavia, pur essendo modellato tramite l'ausilio del *framework* Spectrum descritto in precedenza, il controllo dello stato del mezzo di trasmissione non viene eseguito rilevando la potenza effettivamente presente sul canale, bensì si basa sui livelli `booleani` `true` e `false`. Questa semplificazione, di fatto, impedisce di avere una simulazione coerente con una situazione reale per quanto riguarda i valori di *loss rate* dovuti a interferenze interne (nel caso di nodi che trasmettano su canali IEEE 802.15.4 adiacenti a quello di interesse) ed esterne (nel caso di dispositivi IEEE 802.11 o comunque irradianti potenza nella banda ISM 2.4GHz) e per quanto riguarda lo stato del canale che, in una situazione reale potrà assumere valori intermedi (rispetto agli estremi identificabili in termini di canale libero e canale occupato) che debbono essere gestiti sulla base delle performance che si vogliono ottenere e della *loss rate* che si considera accettabile. Viste queste limitazioni, in questo lavoro si propone, e nel seguito ne verrà data descrizione, un modello di canale alternativo che tenga conto del livello di potenza presente nel mezzo di trasmissione in ogni istante della simulazione.

4.8.2 Lr-wpan-spectrum-value-helper

La descrizione dello spettro relativo alla banda ISM 2.4GHz è realizzata per mezzo degli strumenti resi disponibili dal *framework* Spectrum. Il mezzo di trasmissione è rappresentato da un vettore con granularità di 1Mhz, di conseguenza per ognuno dei sedici canali IEEE 802.15.4 si hanno a disposizione cinque diverse bande di frequenze che permettono di dare una descrizione piuttosto accurata degli eventuali fenomeni dipendenti dalla frequenza che influiscono sulle trasmissioni:

```
for (int i = -1; i < 85; i++)
{
        BandInfo bi;
        bi.fl = 2400.5e6 + i * 1.0e6;
        bi.fh = 2400.5e6 + (i + 1) * 1.0e6;
        bi.fc = (bi.fl +  bi.fh) / 2;
        bands.push_back (bi);
}
```

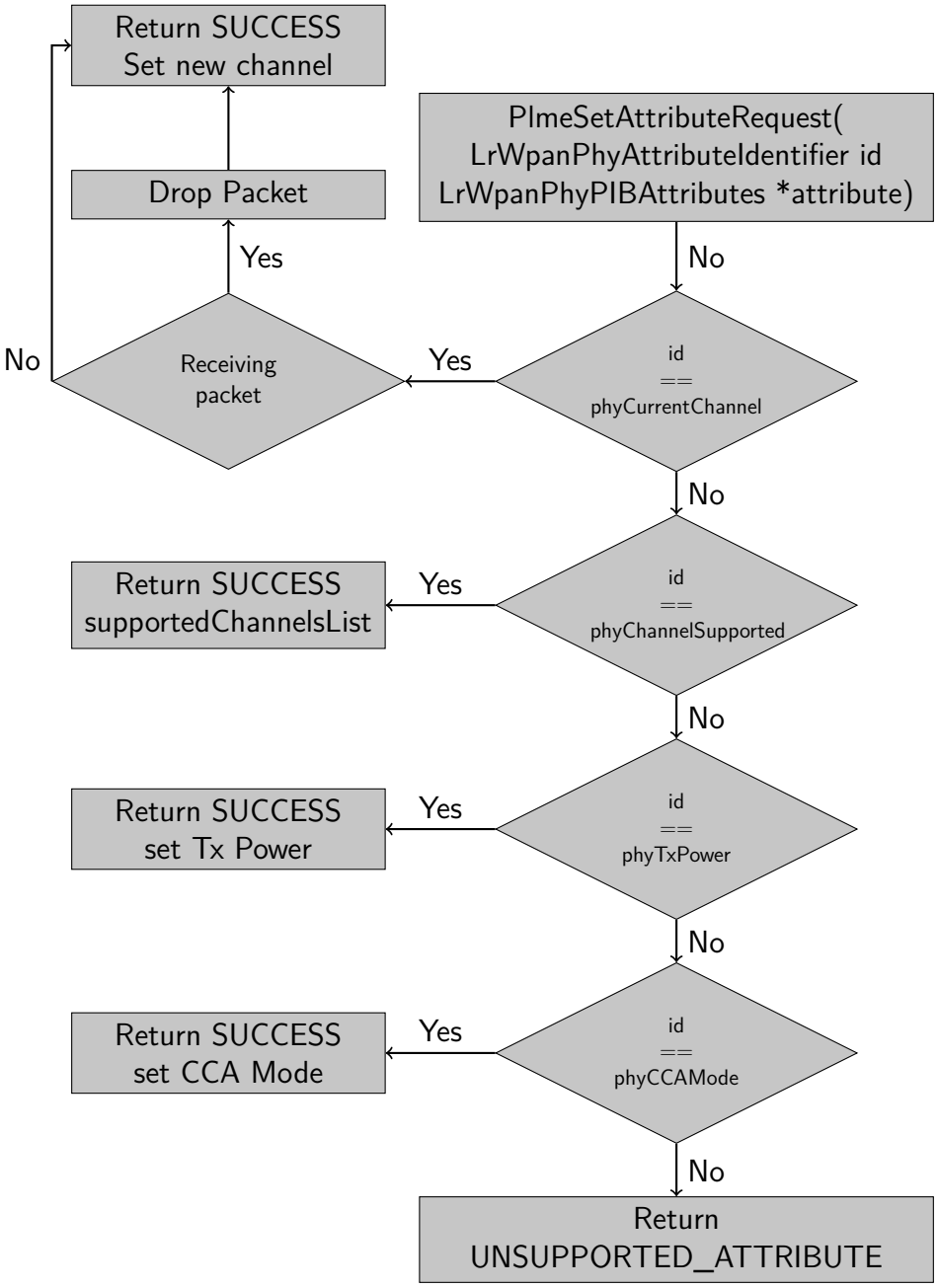


Figura 4.3: Diagramma di flusso per il metodo PlmeSetAttributeRequest

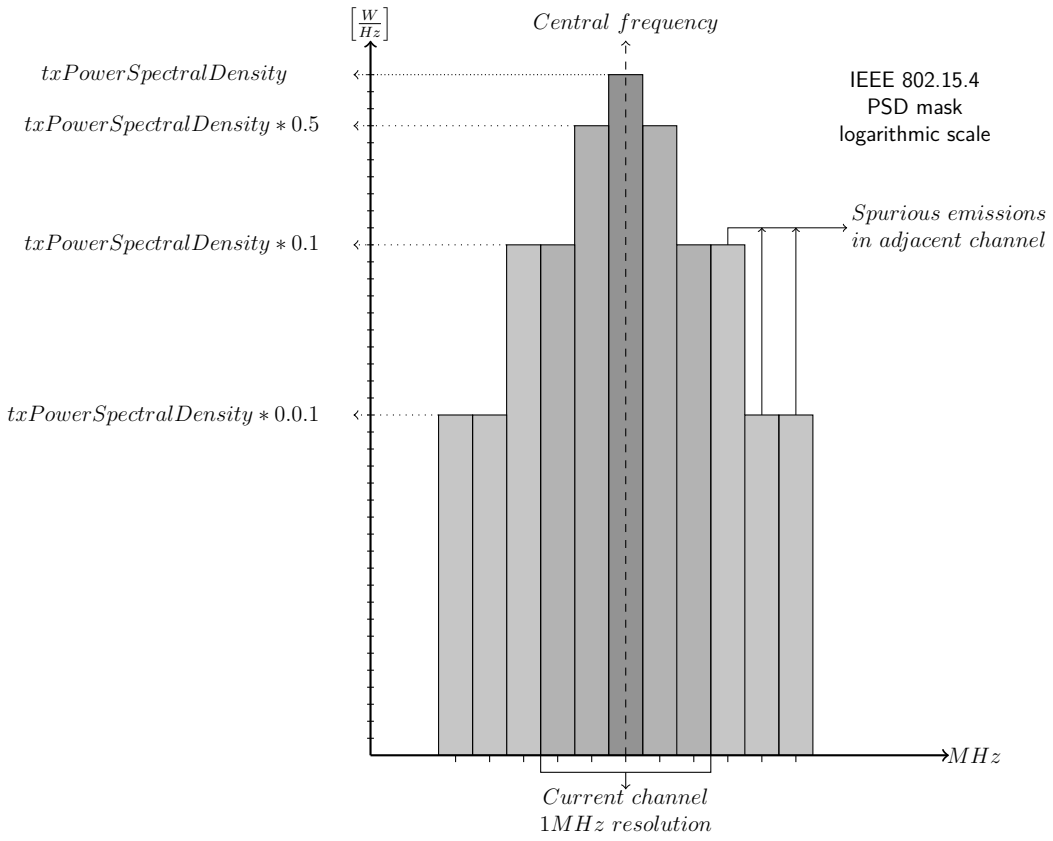
Data questa rappresentazione, l'energia che una trasmissione irradia nel mezzo trasmissivo è definita, seguendo la maschera di PSD definita nello standard, nel seguente modo:

```
(*txPsd)[2405+5*(channel-11)-2400-5]=txPowerDensity*0.01;
(*txPsd)[2405+5*(channel-11)-2400-4]=txPowerDensity*0.01;
(*txPsd)[2405+5*(channel-11)-2400-3]=txPowerDensity*0.1;
(*txPsd)[2405+5*(channel-11)-2400-2]=txPowerDensity*0.1;
(*txPsd)[2405+5*(channel-11)-2400-1]=txPowerDensity*0.5;
(*txPsd)[2405+5*(channel-11)-2400]=txPowerDensity;
(*txPsd)[2405+5*(channel-11)-2400+1]=txPowerDensity*0.5;
(*txPsd)[2405+5*(channel-11)-2400+2]=txPowerDensity*0.5;
(*txPsd)[2405+5*(channel-11)-2400+3]=txPowerDensity*0.5;
(*txPsd)[2405+5*(channel-11)-2400+4]=txPowerDensity*0.01;
(*txPsd)[2405+5*(channel-11)-2400+5]=txPowerDensity*0.01;
```

dove:

- `txPowerDensity` è la PSD espressa in $\left[\frac{W}{Hz}\right]$
- `channel` è l'indice del canale correntemente impostato per la trasmissione.

Questo tipo di rappresentazione è sufficiente nel caso in cui il canale IEEE 802.15.4 impostato per le comunicazioni sia costante, nel corso di questo progetto, tuttavia, è stato necessario affrontare il problema della rappresentazione del mezzo trasmissivo nel caso di comunicazioni parallele che sfruttino contemporaneamente i sedici canali previsti dallo standard.


$\left[\frac{W}{Hz}\right]$
Central frequency
$txPowerSpectralDensity$
$txPowerSpectralDensity * 0.5$
$txPowerSpectralDensity * 0.1$
$txPowerSpectralDensity * 0.0.1$
IEEE 802.15.4
PSD mask
logarithmic scale
Spurious emissions
in adjacent channel
MHz
Current channel
$1MHz$ resolution

Capitolo 5

McMac

Il tentativo di realizzare una politica di accesso al mezzo distribuita e a basso *overhead* in grado di sfruttare nella sua interezza la gamma di canali logici previsti dallo standard IEEE 802.15.4 è approdato al tema principale di questa tesi: il protocollo *Multi Channel Medium Access Control*: **McMac**.

Lo studio dello stato dell'arte nell'ambito dei protocolli di accesso al mezzo per *low rate wireless personal area network* che non prevedono la presenza di un coordinatore centrale, ha evidenziato come non esistano, per quanto è dato sapere all'autore, politiche in grado di sfruttare agilmente l'intera ampiezza della banda ISM 2.4GHz riservata alla tecnologia IEEE 802.15.4 senza l'ausilio di un canale di controllo che permetta, di volta in volta, di definire una frequenza di *rendez vous* per ciascuna coppia di nodi intenzionata ad avviare una comunicazione.
La presenza di un canale di controllo, sebbene sia di fatto la soluzione più intuitiva nel caso in oggetto, si rivela un notevole collo di bottiglia per il *throughput* massimo raggiungibile da una rete, infatti, anche supponendo che i nodi siano sempre attivi e pronti a ricevere (assunzione molto forte specialmente nel caso di reti di sensori dove è pratica diffusa utilizzare *duty cycle* anche molto bassi per preservare la durata di vita dei singoli nodi), all'aumentare del numero di trasmissioni che è necessario effettuare, il canale di controllo raggiungerebbe molto in fretta un punto di saturazione per cui molte trasmissioni non potrebbero essere avviate, non per mancanza di risorse in termini di frequenze disponibili, ma per l'impossibilità di concordare un canale di *rendez vous*.

La rinuncia all'utilizzo di un canale di controllo pone, tuttavia, un serio problema: come avviare una comunicazione con un nodo di cui non si conosce la frequenza operativa e senza avere la possibilità di sfruttare una frequenza comune?

La domanda è, ovviamente, retorica e una possibile risposta verrà fornita nelle sezioni seguenti.

5.1 Idea

L'idea alla base di McMac è di sopperire alla mancanza di un canale di controllo globale tramite l'assegnazione, a ciascun nodo della rete, di una frequenza di riferimento propria, calcolata sulla base dell'indirizzo fisico di ciascun dispositivo (indirizzo che all'interno di una rete è ovviamente univoco).

L'indice del canale IEEE 802.15.4 calcolato sulla base dell'indirizzo di un nodo n verrà chiamato *base channel* per n e verrà indicato, nel seguito, con ch_n. L'assegnazione di un *base channel* per ciascun dispositivo appartenente alla rete comporta principalmente due vantaggi:

- La possibilità di avere tante comunicazioni parallele quanti sono i canali disponibili nella rete (per un massimo di sedici come previsto dallo standard).
- La riduzione dell'*overhead* che si avrebbe nel caso della presenza di un canale di controllo, dal momento che volendo avviare una trasmissione, un nodo non ha necessità di contattare il destinatario per concordare una frequenza di *rendez vous*.

Il calcolo del *base channel* di un nodo può essere effettuato tramite una qualunque funzione di *hash* così definita: Siano dunque:

- I: lo spazio degli indirizzi possibili;
- i: l'indirizzo di un nodo generico n;
- h la funzione di *hash* selezionata;
- K l'insieme di canali disponibili per trasmettere.

$$h : I \to K$$

$$\forall i \in I,\ h(i) = k\ :\ k \in K \wedge P[h(i) = k] = \frac{1}{|K|}$$

L'uniformità della distribuzione degli output della funzione h è un requisito fondamentale allo scopo di poter ripartire equamente il carico trasmissivo tra i canali di sponibili.

A partire da questa idea il, lavoro di progettazione si è sviluppato nella direzione di un protocollo basato sulla tecnica del *preamble sampling* con politica di accesso a ciascun canale basata su CSMA-CA. A tal proposito si è deciso di utilizzare, come base di partenza per la definizione del comportamento da mantenere nel processo di trasmissione su ciascun canale, una variante del protocollo,

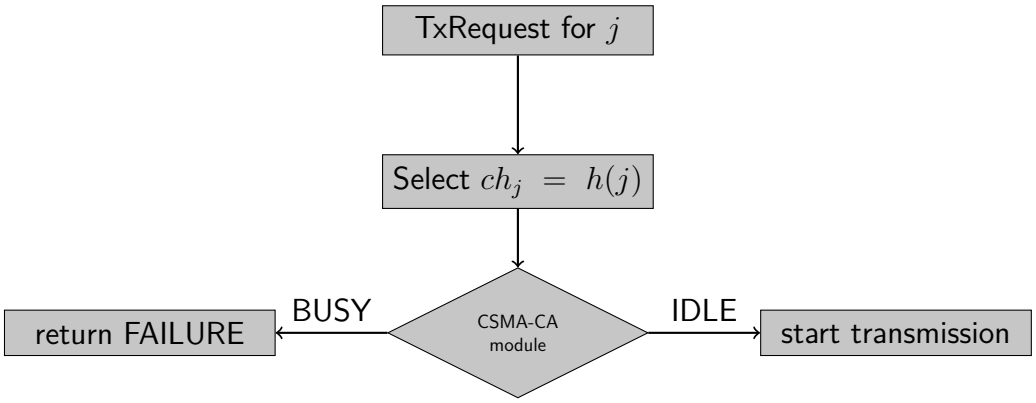


Figura 5.1: Schema generale dell'avvio di una trasmissione in McMac

già descritto in precedenza, XMAC, mentre, nell'ambito dell'accesso al canale selezionato, si è sviluppata una tecnica di *carrier sense multiple access with collision avoidance* tagliata ad hoc per l'applicazione in McMac. Il funzionamento generale del protocollo sarà dunque quello proprosto in figura 5.1. Si procederà ora ad una descrizione del funzionamento generale delle procedure di accesso al mezzo, trasmissione e ricezione.

5.2 Trasmissione

Al momento della ricezione di una richiesta di trasmissione proveniente dal livello superiore il livello MAC provvederà alle seguenti operazioni:

1. Stop del *duty cycle* e accensione del *transceiver* (qualora necessario);
2. Selezione del canale relativo al destinatario;
3. Avvio del modulo CSMA-CA;
4. Nel caso di canale libero avvio della trasmissione, altrimenti comunicazione del fallimento durante l'accesso al mezzo al layer chiamante;
5. Trasmissione di *strobed preamble* come previsto in XMAC per un tempo massimo pari alla durata del tempo di *sleep* previsto dal *duty cycle*;
6. Se durante la trasmissione di *strobed preamble* viene ricevuto il relativo *acknowledgment*, avvio della trasmissione dei dati;
7. Se, invece, nessun *ack* viene ricevuto, allo scadere del *timeout* verrà comunicato il fallimento al *layer chiamante*.

L'intero processo di trasmissione è descritto in figura 5.2

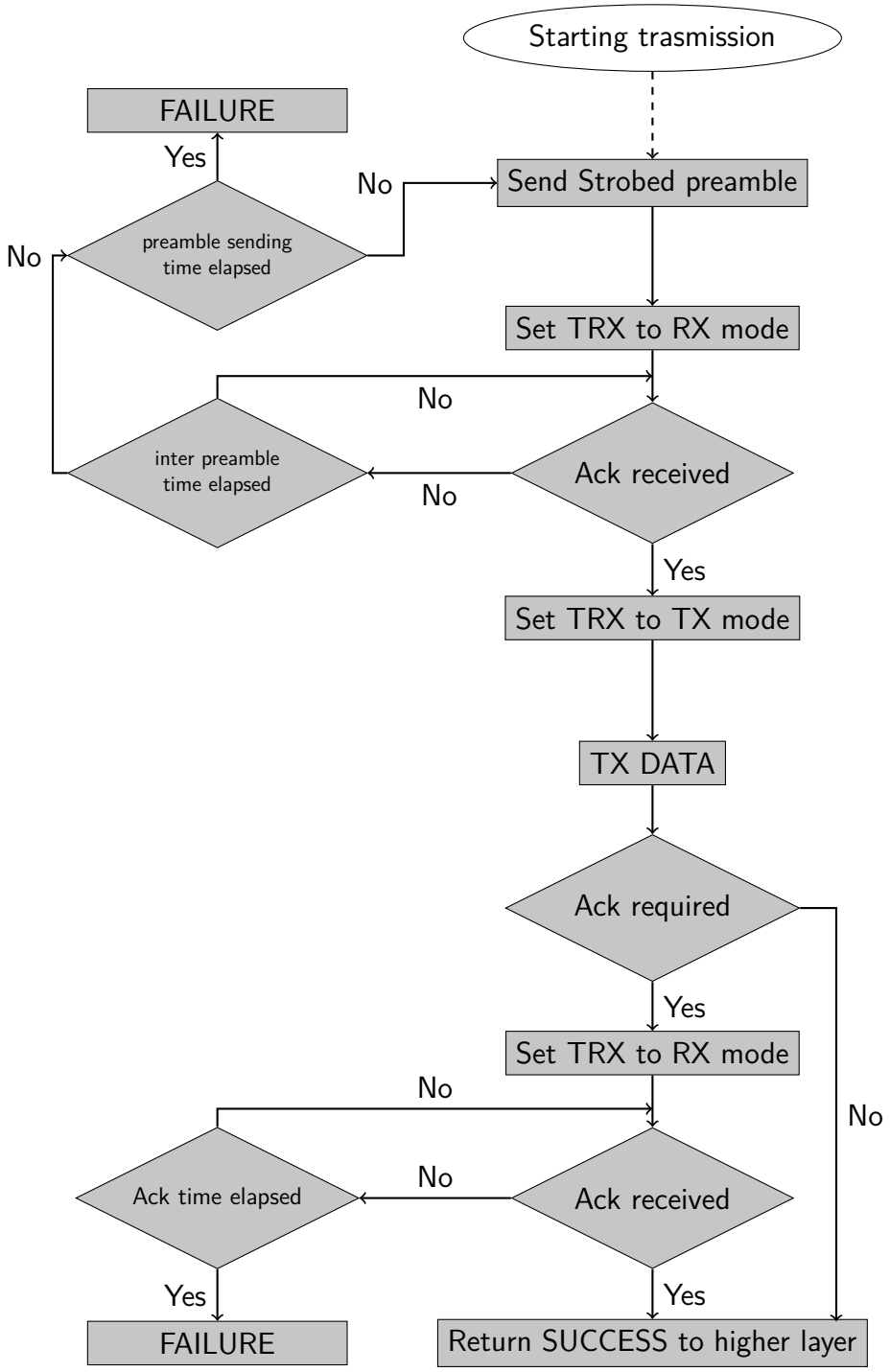


Figura 5.2: Diagramma di flusso di una trasmissione secondo il protocollo McMAC

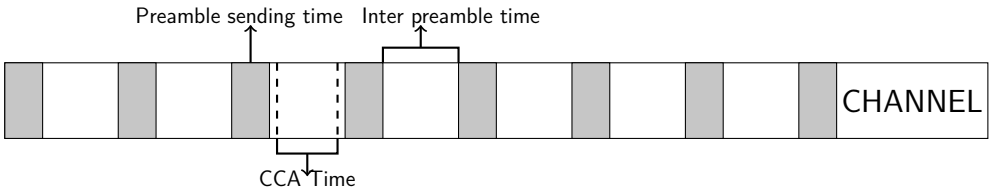


Figura 5.3: Esempio di fallimento di CSMA basato su un singolo periodo di CCA: in questo caso il canale sarebbe stato dichiarato IDLE

5.3 CSMA-CA

Il progetto del modulo CSMA-CA ha richiesto particolare attenzione data la particolare la natura di una trasmissione in McMac. in particolare, il principale problema che è stato affrontato nello sviluppo di questa componente del protocollo è stato:

- La modalità di identificazione dello stato del canale: la presenza di preamboli di breve durata, ma trasmessi in rapida successione potrebbe, nel caso di utilizzo di un protocollo CSMA-CA standard, portare all'errata conclusione di canale libero come descritto in figura 5.3.

Per quanto riguarda questo punto, la procedura che determina lo stato corrente del canale prevista dallo standard è stata modificata in modo da tenere in considerazione la presenza dei preamboli. Anziché utilizzare un unico periodo di CCA, nè vengono utilizzati tre opportunamente distanziati nel tempo e lo stato del canale corrisponde all'*AND* logico tra i risultati (si veda figura 5.4).
Sia dunque:

- T_p il tempo di invio di un preambolo;
- T_w il tempo di attesa tra un preambolo e il successivo;
- $T_{c_i},\ i = 1, 2$ il tempo che intercorre tra due operazioni di CCA.

allora:

$$T_{c_1} = T_w + U[1, |T_w - T_p|]$$
$$T_{c_2} = T_w$$

Nel caso in cui per tutte le rilevazioni il canale risulti IDLE si procede all'avvio della trasmissione, se invece per al termine di una rilevazione il mezzo trasmissivo risulta BUSY si interrompe il *sensing* e si procede all'avvio della procedura di *backoff*.

Sia ora:

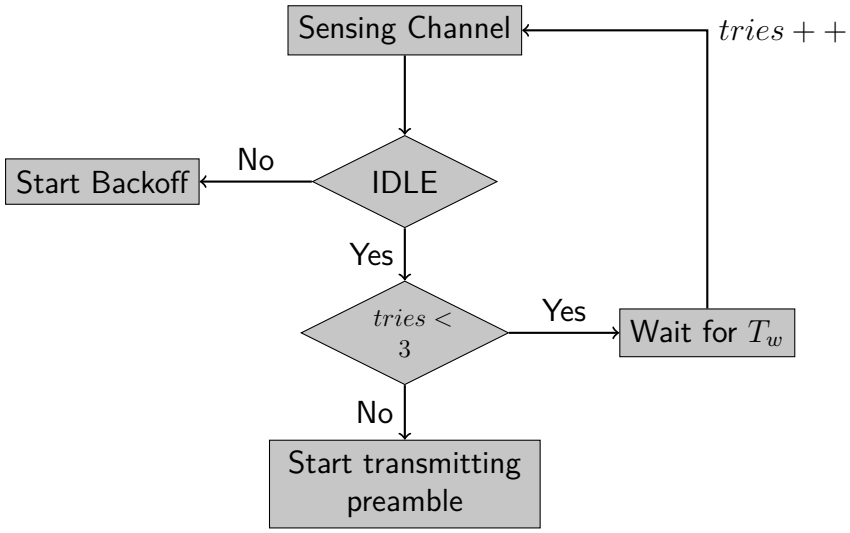


Figura 5.4: Procedura che determina lo stato del canale in McMac

- T_s il tempo di *sleep* di un dispositivo;
- T_d il tempo necessario a trasmettere un pacchetto dati (senza preamboli);
- T_{ack} il tempo necessario a ricevere un *acknowledgment*.

allora, supponendo che in ogni istante la probabilità che un nodo decida di trasmettere in un dato istante sia uniforme, la durata media di una trasmissione sarà:

$$T_{tx} = T_d + \left\lceil \frac{T_s}{2} \right\rceil + T_{ack}$$

All'aumentare, dunque, del tempo di sleep di un nodo, la durata delle trasmissioni aumenterà linearmente, di conseguenza quest'ultima potrebbe diventare coniderevolmente più lunga di una normale trasmissione basata su CSMA-CA, di conseguenza i parametri di *backoff* previsti dallo standard IEEE 802.15.4 sono stati rivisti in una chiave meno aggressiva:

- *Minimum backoff exponent*: da 3 a 7;
- *Maximum backoff exponent*: da 5 a 10;
- *Maximum backoffs number*: da 4 a 8.

La procedura completa di accesso al canale è schematizzata in figura 5.5.

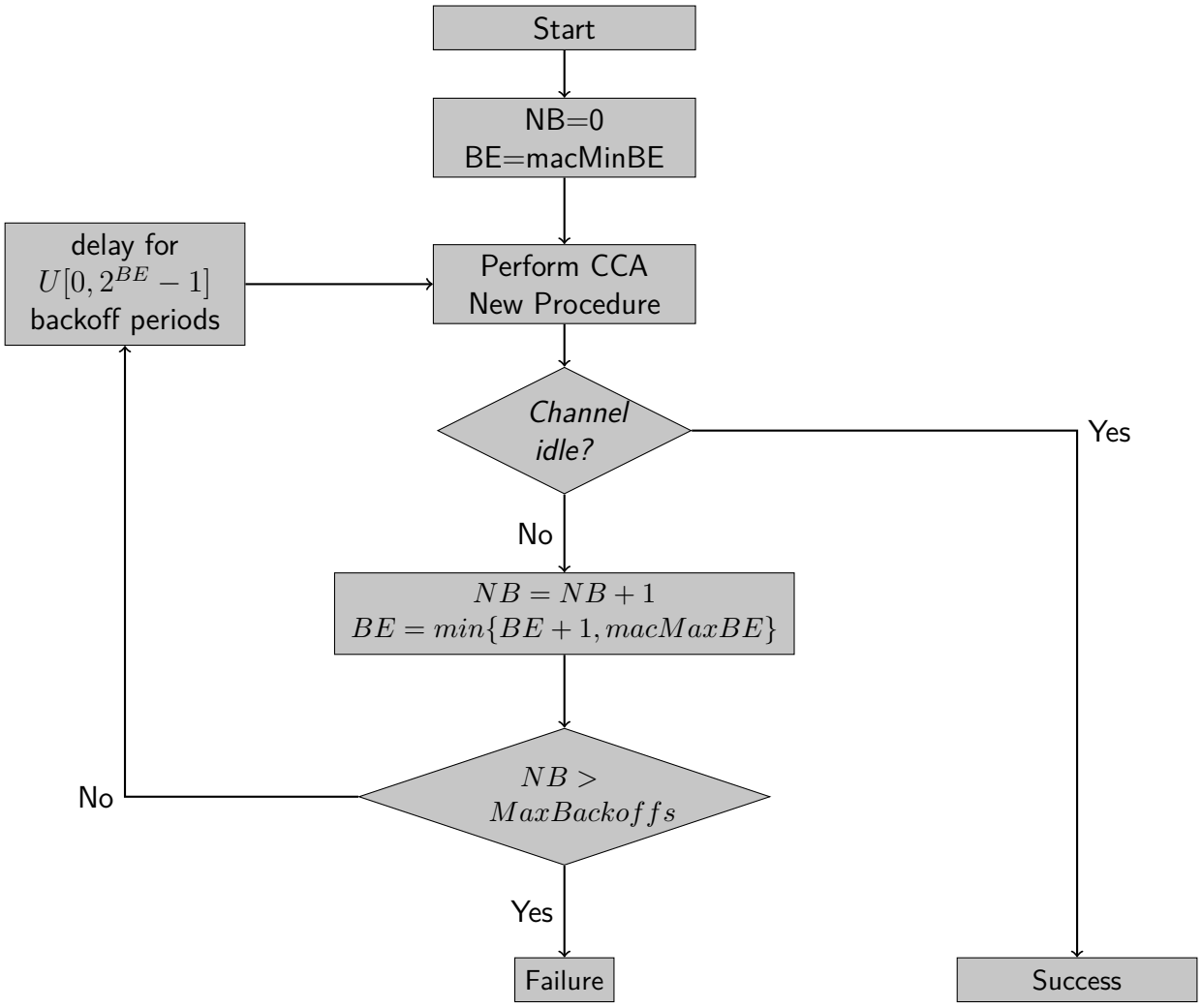


Figura 5.5: Procedura completa di accesso al mezzo come definita in McMac

5.4 Ricezione

La procedura di ricezione di pacchetti dati non necessita di particolari spiegazioni e pertanto viene solamente schematizzata in figura 5.6.

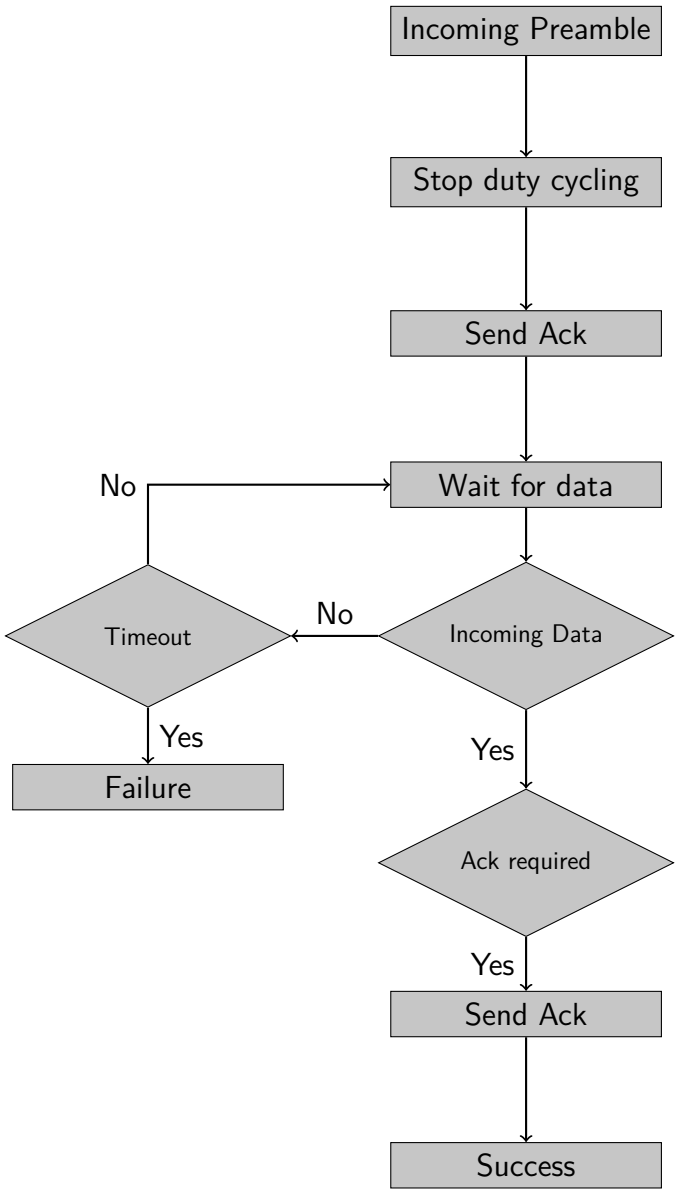


Figura 5.6: Ricezione in McMac

5.5 Implementazione

L'implementazione del protocollo McMac all'interno del *framework* NS-3 ha richiesto due fasi:

1. Adattamento della versione del livello fisico di un ideale *transceiver* IEEE 802.15.4 di cui si è discusso in precedenza alla necessità di gestire più trasmissioni in parallelo su canali diversi;

2. La realizzazione delle procedure sopra descritte e l'opportuna definizione dei parametri caratteristici del protocollo.

Come già accennato in precedenza, i livelli di potenza presenti nel mezzo trasmissivo sono rappresentati come valori a doppia precisione in un vettore rappresentante una discretizzazione dello spettro in sottobande di granularità predefinita. Allo scopo di ottenere la capacità di gestire più trasmissioni in contemporanea, ogni nodo appartenente alla rete è stato dotato di una propria rappresentazione dello spettro che permette di valutare i valori locali di potenza presenti su ciascuno dei canali IEEE 802.15.4.
La classe `SingleModelSpectrumChannel` è stata ridefinita modificando il metodo `StartTx` permettendogli di conoscere il *base channel* dei nodi che risultano collegati, in questo modo, al momento di avviare una trasmissione, per i nodi il cui base channel coincide con quello del destinatario verrà chiamata la funzione `LrWpanPhy::StartRx`, per tutti gli altri, invece, verrà aggiornata la rappresentazione dello stato locale del canale con un valore di interferenza pari alla potenza del segnale che dovrebbe essere ricevuto.

```
void SingleModelSpectrumChannel::StartTx (Ptr<PacketBurst> p,
        Ptr <SpectrumValue> txPsd,
        SpectrumType st,
        Time duration,
        Ptr<SpectrumPhy> txPhy)
{
  uint8_t logicalChannel;
  ...
  logicalChannel = txPhy->GetCurrentLogicalChannel();

  PhyList::const_iterator rxPhyIterator = m_phyList.begin ();

  ...

  while (rxPhyIterator != m_phyList.end ())
```

```
    {
      if ((*rxPhyIterator) != txPhy)
        {
          Ptr <SpectrumValue> rxPsd;
         ...
          else
            {
              if(logicalChannel!=0)
              {
                if(logicalChannel==(*rxPhyIterator)->
                        GetCurrentLogicalChannel())
                {
                  Simulator::Schedule (delay,
                         &SingleModelSpectrumChannel::StartRx,
                          this,
            pktBurstCopy,
            rxPsd, st,
            duration,
            *rxPhyIterator);
                }
                else
                {
                  (*rxPhyIterator)->AddNoisePsd(rxPsd);
                    Simulator::Schedule(delay+duration,
                          &SpectrumPhy::SubtractNoisePsd,
                          *rxPhyIterator, rxPsd);
                }
              }
            }
        }
      ++rxPhyIterator;
    }

}
```

Come si può notare da un rapida analisi del codice precedente, tutti i nodi il cui transceiver ha come *base channel* la stessa frequenza (quindi circa $\frac{number\ of\ nodes}{number\ of\ channels}$) riceveranno tutti i pacchetti trasmessi su quel canale, sarà compito di ogni singolo transceiver identificare le trasmissioni di cui non è destinatario e agire di conseguenza, nello specifico, tornando ad eseguire il proprio

duty cycle.
Ogni volta che un nodo avvia una trasmissione, la rappresentazione locale del canale di tutti gli altri nodi presenti nella simulazione verrà aggiornata di conseguenza, nel caso in cui uno di quest'ultimo gruppo di nodi sia impegnato nel ricevere un pacchetto, la fluttuazione nella potenza presente nel canale viene acquisita e mediata su tutta la durata della ricezione, in questo modo, quando verrà chiamato il relativo metodo `LrWpanPhy::EndRx` si avrà a disposizione una descrizione storica della variazione dei livelli di potenza presenti nel canale durante tutta la ricezione. Avere a disposizione questi dati è di fondamentale importanza per il calcolo del *Signal to Noise Plus Interference Ratio* (SINR) e, di conseguenza, per il calcolo della *Bit Error Rate* (BER) che permette, in seguito, il calcolo della *Packet Error Rate*(PER). Una volta ottenuta la probabilità di errore sul bit, la PER può essere calcolata attraverlo al formula:

$$1 - (1 - BER)^{packet\ \ size[bit]}$$

Il codice seguente dimostra quanto detto finora e come vengano effettuati i calcoli del SINR e della PER:

```
const double LrWpanPhy::BitErrorRateHigh[]={
0.344578258389676, 0.326766648389273, 0.307240937709636,
0.285965333118419, 0.262958976255790, 0.238318546375371,
0.212244669084648, 0.185070001481989, 0.157284441253224,
0.129549539662669, 0.102690386688045, 0.077650549521640,
0.055397228139874, 0.036774145770082, 0.022322386868133,
0.012121472499855, 0.005727760067982, 0.002274582098207,
0.000726498895173, 0.000176579840762, 0.000030458505715,
0.000003414516423, 0.000000222663414, 0.000000007344950,
0.000000000102778, 0.000000000000489, 0.000000000000001,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
};
```

```
void LrWpanPhy::EndRx ()
{
  Ptr<SpectrumValue> rxPsd;
  ...

  if(m_phyPIBAttributes.phyCurrentChannel==11)
  {
    averageRxPsd =
```

```
    (*rxPsd)[5*(m_phyPIBAttributes.phyCurrentChannel-11)];
  }
  else if(m_phyPIBAttributes.phyCurrentChannel==26)
  {
    averageRxPsd =
        (*rxPsd)[5*(m_phyPIBAttributes.phyCurrentChannel-11)];
  }
  else
  {
    averageRxPsd =
        (*rxPsd)[5*(m_phyPIBAttributes.phyCurrentChannel-11)];
  }

  ...

  if(m_spectrumSamplesNumber!=0)
  {
    energy = m_averageSpectrumFluctuation /
             m_spectrumSamplesNumber;
    if(energy!=0)
    {
      sinr = averageRxPsd/energy;
      intSinr = (int)(10*log10(sinr));
      bitErrorProbability = BitErrorRateHigh[intSinr+20];
      packetErrorProbability = 1-pow((1-bitErrorProbability),
      (m_currentRxPacket.first)->GetSize()*8);

      if(m_randomErrorGenerator.GetValue()<packetErrorProbability)
      {
        error = true;
      }
      else
      {
        error = false;
      }
    }
  }
  else
  {
    error = false;
  }
```

```
  m_averageSpectrumFluctuation = 0;
  m_spectrumSamplesNumber = 0;
  if(!error)
  {
    if (!m_pdDataIndicationCallback.IsNull ())
    {
      Ptr<Packet> p = m_currentRxPacket.first;
      m_pdDataIndicationCallback (p->GetSize (), p, 0);
    }
  }
}
```

L'implementazione delle funzionalità sopra descritte ha permesso di costruire il protocollo McMac sfruttando in modo nativo (grazie a quanto fatto) il supporto alle comunicazioni parallele.
Una volta definite le *state machine* relative alle varie fasi del protocollo e apportati gli interventi descritti a livello fisico e canale, il lavoro si è indirizzato verso:

- Il *Testing* del protocollo;
- Il *Tuning* dei parametri che ne determinano il funzionamento;
- Il miglioramento delle prestazioni del protocollo originale.

Mentre i primi due punti saranno oggetto di capitoli successivi, si ritiene opportuno, in questa sede, dedicare qualche riga al processo di raffinamento del protocollo originale.

5.5.1 Sincronizzazione dei *Duty Cycle*

I primi risultati sperimentali hanno evidenziato come la quantità di preamboli inviati per l'avvio di ciascuna trasmissione rappresentasse una componente importante nel consumo complessivo di energia. Allo scopo di ridurre tale consumo è stata studiata una modalità di mutua sincronizzazione tra i *duty cycle* dei nodi appartenenti alla rete che non preveda lo scambio di messaggi di controllo.
Il primo passo verso l'ottenimento della sincronizzazione di cui sopra ha visto la modifica del comportamento dei nodi quando, finita un'operazione, si trovassero nella condizione di riprendere l'attività di *duty cycle*, tale modifica è rappresentata in figura 5.7. La consapevolezza dello stato in cui dovrebbe trovarsi il *transceiver* se avesse continuato ininterrotto il proprio *duty cycle* è stata aggiunta, in ciascun nodo, per mezzo di due oggetti `Timer` aggiuntivi impostati con gli stessi valori di *delay* dei corrispettivi timer che gestiscono il *duty cycle*:

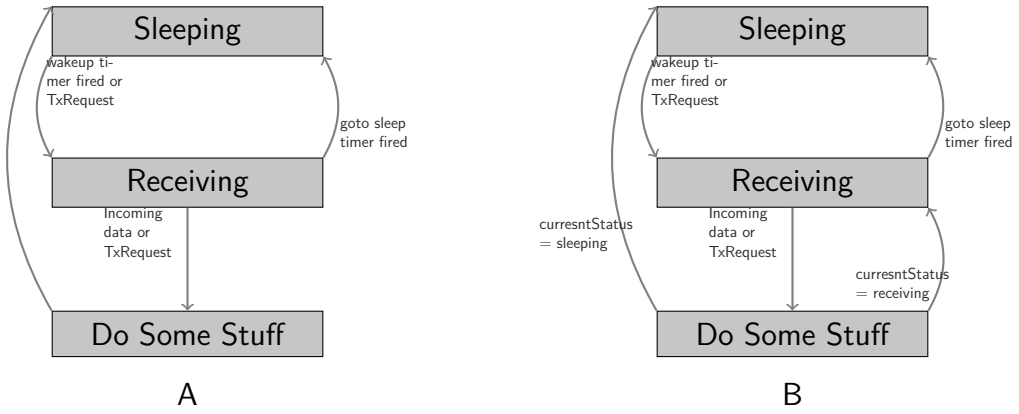


Figura 5.7: Comportamento del protocollo originale (A) e comportamento modificato per la sincronizzazione (B)

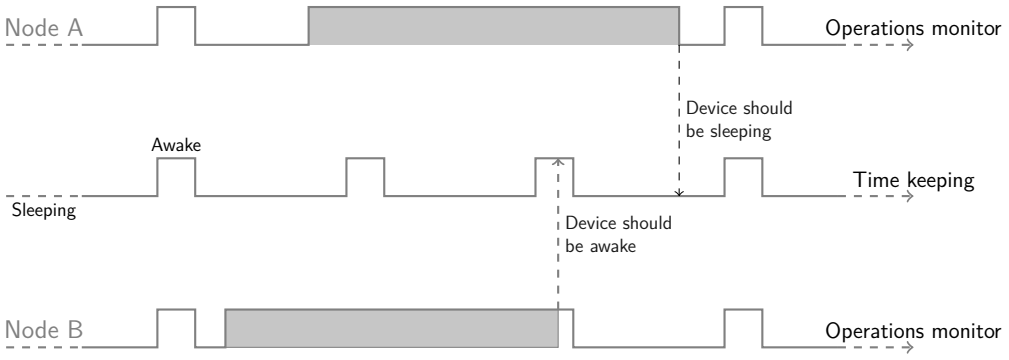


Figura 5.8: Mantenimento della periodicità dell'alternanza sonno veglia per mezzo di un sistema di *time keeping*

- `SleepTimeKeeper`;
- `WakeTimeKeeper`.

L'utilizzo dei temporizzatori aggiuntivi sopra citati permette di mantenere la periodicità locale dell'alternanza sonno/veglia di ciascun nodo, senza che essa sia perturbata da operazioni di invio e ricezione (si veda figura 5.8). Una volta ottenuto questo risultato si è potuto procedere all'implementazione del procedimento di auto sincronizzazione istantanea col *duty cycle* del destinatario.

Si supponga, allora, che il nodo i voglia trasmettere al nodo j, suo vicino, un pacchetto dati, i provvederà a:

1. Inviare il pacchetto come previsto nella definizione originale di McMac, nel caso in cui sia la prima volta che contatta j, al momento della ricezione

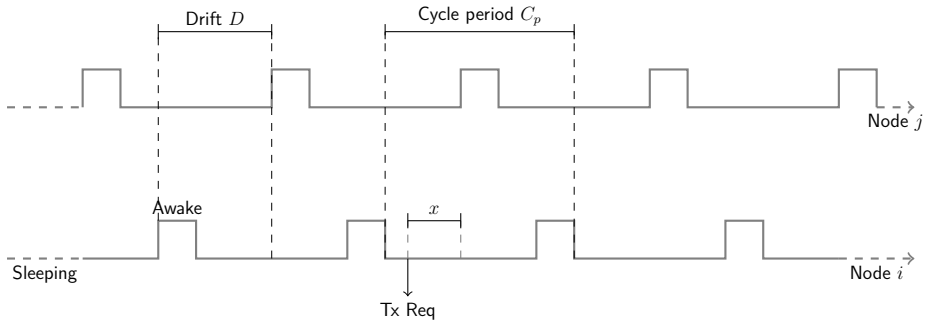


Figura 5.9: Quantità che permettono di stimare il ritardo da applicare a una trasmissione verso un nodo di cui si conosce lo sfasamento tra i rispettivi cicli di sonno/veglia

dell'*acknowledgment* relativo a uno dei preamboli, i stimerà la differenza temporale tra il proprio *duty cycle* rispetto a quello di j;

2. Stimare il tempo di ritardo da applicare alla trasmissione in modo da avvicinare l'inizio di quest'ultima il più possibile all'istante di *wakeup* di j, minimizzando in questo modo il numero di *strobed preamble* che sarà necessario trasmettere per notificare a j l'intenzione di avviare una comunicazione.

Il procedimento di stima del ritardo è rappresentato in figura 5.9. Come si può notare, il ritardo da applicare alla richiesta di trasmissione di i corrisponde al tempo x che può essere calcolato nel seguente modo:

$$x - D = SleepingTimeLeft - CyclePeriod \Rightarrow$$

$$x = D + SleepingTimeLeft - CyclePeriod$$

Nel caso in cui, invece, la richiesta di trasmissione arrivi durante il periodo di veglia di i, il calcolo del ritardo sarà:

$$x - D = AwakeTimeLeft - CyclePeriod \Rightarrow$$

$$x = D + AwakeTimeLeft - CyclePeriod$$

5.5.2 Bufferizzazione delle richieste di trasmissione

Un ulteriore abbellimento apportato alla versione originale del protocollo in oggetto è stato l'implementazione della capacità di memorizzare richieste di trasmissione, che non possono essere gestite immediatamente, tramite l'utilizzo di

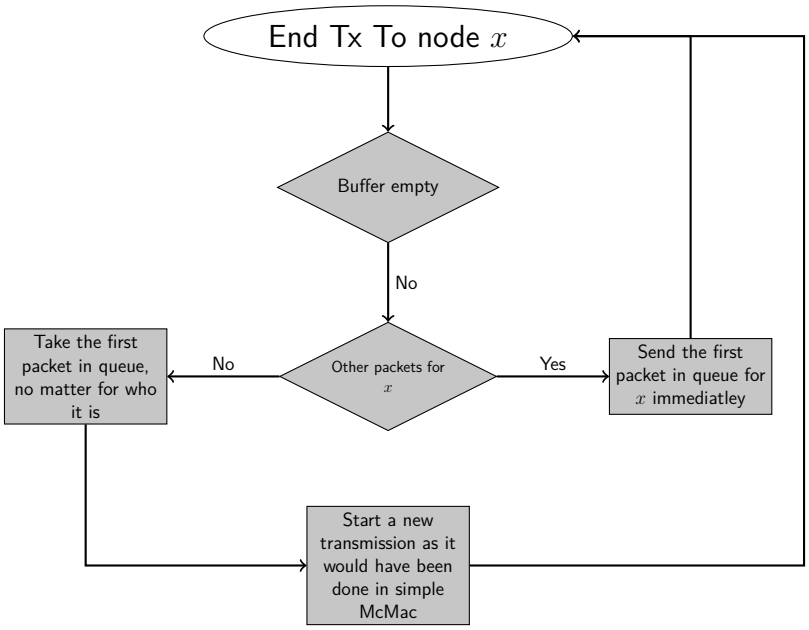


Figura 5.10: Schema generale di funzionamento del processo di trasmissione dati con supporto alla bufferizzazione

un *buffer* come coda di trasmissione.
Il buffer in cui vengono salvati i pacchetti e i relativi metadati comunicati al livello MAC dal *layer* superiore è stato implementato per mezzo di una *doubly linked list*. All'interno di questa struttura dati i pacchetti raggruppati utilizzando come elemento comune il destinatario, ciascuno di questi gruppi è ordinato sulla base dell'istante di inserimento realizzando in questo modo una struttura simile ad una *queue*.
Al momento di inserire un nuovo pacchetto nella lista, vengono eseguite le seguenti operazioni:

1. Verifica se sono presenti altri pacchetti col medesimo destinatario, se sì allora salva il pacchetto corrente in coda a questi ultimi;

2. Se, invece, non sono presenti pacchetti col medesimo destinatario, salva il pacchetto in coda a tutti gli altri.

Per utilizzare in modo efficiente questa nuova struttura dati, è stata reimplementato il metodo `McMac::McpsDataRequest` realizzando una politica di utilizzo aggressivo del canale, il comportamento del metodo di cui sopra è descritto in figura 5.10.

La politica di trasmissione implementata nel caso in oggetto risulta molto aggressiva, nel caso vi siano più pacchetti con medesimo destinatario presenti

nel buffer, dal momento che una volta acquisito l'accesso al canale il nodo in trasmissione non lo rilascia fino al completo svuotamento della porzione di coda relativa al destinatario di cui sopra. Va pertanto usata cautela nell'attivare questa funzionalità che potrebbe portare ad un comportamento della rete in alcuni casi *unfair*.

Capitolo 6

Validazione

Il processo di *testing* del protocollo descritto in precedenza ha richiesto la definizione dei parametri fondamentali sulla base dei quali studiarne il comportamento e l'implementazione di un generatore di traffico adatto allo studio delle grandezze di interesse.
In particolare l'attenzione si è concentrata sull'ottenimento, per ciascuna delle tre varianti del protocollo, di dati relativi a:

- *Throughput* massimo a regime raggiungibile;
- Consumo energetico;
- *Overhead* medio per pacchetto dati inviato;
- Percentuale di trasmissioni fallite rispetto alle richieste ricevute;
- Tempo trascorso nei vari stati di funzionamento del *transceiver*:
  - Sleeping;
  - Transmitting;
  - Receiving.

Le grandezze sopra elencate sono state studiate al variare dei seguenti parametri:

- Numero di canali disponibili;
- Numero di trasmettitori presenti nella rete;
- Tempo medio che intercorre tra una richiesta di trasmissione e la successiva per ciascun trasmettitore della rete.

6.1 Impostazione delle simulazioni

L'impostazione generale di una simulazione prevede una topologia casuale, ma vincolata al fatto che tutti i nodi presenti (per un totale costante di sessantaquattro) siano nel medesimo raggio di trasmissione. Si è inoltre ritenuto opportuno, in questa fase dello studio del protocollo, assumere le trasmissioni istantanee e il canale affetto solamente da *Additive White Gaussian Noise* (AWGN) senza invece considerare fenomeni quali il *path loss*, il *multipath fading* e la presenza di agenti esterni forieri di interferenze (come per esempio dispositivi IEEE 802.11 b/g/n o forni a microonde -che irradiano potenza nella banda ISM a 2.4GHz -) Per ciascuna simulazione, i parametri che la caratterizzano sono:

1. Il numero di nodi (tra i sessantaquattro totali) abilitati a trasmettere;
2. Il numero di canali disponibili (tra i sedici totali);
3. Il tempo medio che intercorre tra una richiesta di trasmissione e la successiva.

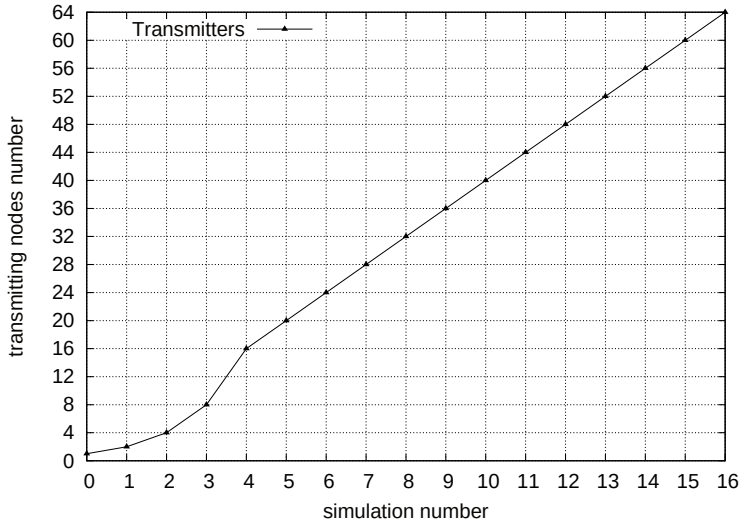


Figura 6.1: Variazione del numero di nodi abilitati alla trasmissione al variare dell'indice di simulazione

Per quanto riguarda il primo punto, si è ritenuto opportuno far variare il numero di trasmettitori n_{tx} nel seguente modo (si veda figura 6.1):

$$\begin{cases} n_{tx} = 2^i & i = 0, \ldots, 4 \\ n_{tx} = 4i & i = 5, \ldots, 16 \end{cases}$$

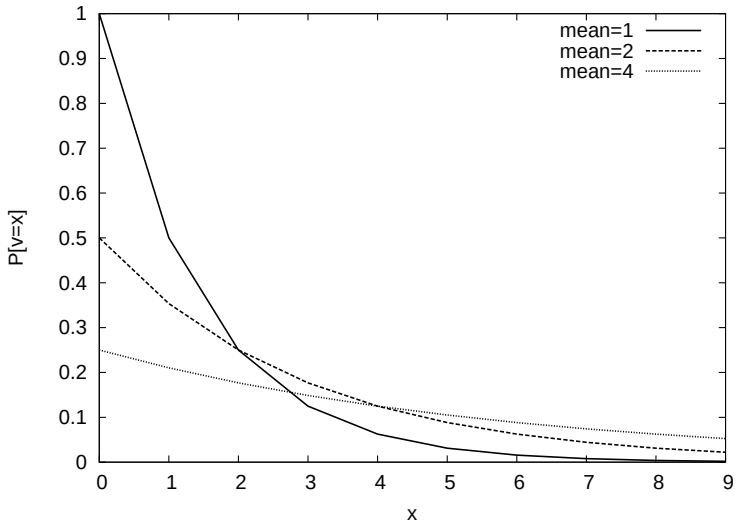


Figura 6.2: Distribuzione di una variabile esponenziale v con medie di 1,2,4 secondi

Per quanto riguarda, invece, la variazione del numero di canali utilizzati, dopo alcuni test, si è ritenuto opportuno limitare il numero di casi a 1, 8 e 16.

Il tempo medio che intercorre tra una trasmissione e la successiva, infine, è regolato, per ciascuno dei nodi abilitati a trasmettere, da una variabile aleatoria con distribuzione esponenziale e media $\mu = 1, 2, 4$ secondi (si veda figura 6.2). A ciascun nodo abilitato a trasmettere si richiede di elaborare correttamente cento richieste di trasmissione. Una richiesta risulta elaborata correttamente quando:

- il pacchetto viene correttamente inviato nel canale;
- la trasmissione fallisce perché il canale viene trovato occupato per un numero di volte superiore al numero massimo di *backoff* permessi;
- la trasmissione fallisce perché non viene ricevuto nessun *acknowledgment* durante la trasmissione dei preamboli.

Si è inoltre ritenuto opportuno mantenere costante l'entità del *duty cycle* impostato a 1% con durata del ciclo sonno/veglia pari a $1s$, il tempo trascorso nei due possibili stati per ciascun ciclo sarà:

- $10ms$ in stato di ricezione;
- $990ms$ in stato di *sleep*.

6.2 Implementazione del generatore di traffico

Il generatore di traffico (il cui codice è reperibile nella classe `LinkLayer3`) è stato implementato come livello applicativo agganciato direttamente al livello MAC. Le richieste di trasmissione vengono generate ad intervalli distribuiti come descritto in precedenza per mezzo della classe `ns3::ExponentialVariable` fornita da NS-3:

```
ExponentialVariable expVariable;
...
m_nextTxTime = MilliSeconds(expVariable.GetValue());
```

Una volta definito il tempo di attesa prima della successiva richiesta di trasmissione viene selezionato un ricevitore tra i sessantaquattro nodi possibili. Sia allora $i \in \{1, \ldots, 64\}$ l'indirizzo di un trasmettitore e sia $x \in \{1, \ldots, 64\}$ l'indirizzo del ricevitore selezionato dal nodo i:

$$\begin{cases} P[x = i] = 0 \\ P[x = j : j \in \{1, 64\}, j \neq x] = \frac{1}{63} \end{cases}$$

```
UniformVariable uVariable;
MACAddress m_nextDstAddress;
...
m_nextDstAddress.shortAddress =
        uVariable.GetInteger(0x01,m_neighboursNumber);

while(m_nextDstAddress.shortAddress==
        m_localMac->GetLocalAddress().shortAddress)
{
        m_nextDstAddress.shortAddress=
          uVariable.GetInteger(0x01, m_neighboursNumber);
}
```

La *callback* di livello MAC `McpsDataConfirmCallback` è fatta puntare al metodo

```
void LinkLayer3::McpsDataConfirm(
        uint32_t msduHandle,
        IEEE_802_15_4_Mac__Enumeration status,
        Time timestamp,
        int64_t numPreambleSent);
```

per mezzo delle istruzioni:

```
Ptr<McMac3> m_localMac;
...

m_localMac->SetMcpsDataConfirmCallback(
        MakeCallback(&LinkLayer3::McpsDataConfirm, this));
```

All'interno del metodo `McpsDataConfirm` vengono gestiti i possibili esiti di ciascuna trasmissione e viene tenuta traccia del conteggio delle richieste elaborate, una volta che tale conteggio raggiunge, per un dato nodo, le cento unità, viene disabilitata la trasmissione e il nodo in questione diventa un ricevitore:

```
if(m_packetsSent==100)
{
        m_canTx = false;
        m_jobDoneCallback(m_localMac->GetLocalAddress(),
                Simulator::Now());
        return;
}
```

Quando tutti i nodi hanno correttamente elaborato cento richieste di trasmissione la simulazione termina e i dati vengono aggregati per la successiva analisi:

```
void JobDone(MACAddress addr, Time endTime)
{
  end++;
  if(end==txNodesNumber)
  {
    for(int i = 0; i<nodesNumber; i++)
    {
      macList[i]->StopMac();
    }
        std::cout
                << "Simulation ended at: "
                << Simulator::Now() << "\n";
    Simulator::Stop(Simulator::Now());
  }
}
```

6.3 Raccolta dei risultati

Al fine di ottenere risultati statisticamente rilevanti, per ciascuna simulazione vengono eseguiti venti *run* facendo avanzare l'indice dei sottoflussi generati dal PRNG come descritto in 4.2. Questa operazione viene effettuata per mezzo dell'impostazione del valore `RngRun` relativo alla variabile d'ambiente `NS_GLOBAL_VARIABLE` al momento del lancio di ciascuna simulazione, in particolare, volendo implementare quanto detto in `bash`:

```
for i in {1..20}
do
        NS_GLOBAL_VALUE="RngRun=$i" ./waf --run Simulation
done
```

I risultati ottenuti durante la campagna di raccolta dati sono stati in seguito elaborati per mezzo di routine in linguaggio Perl ed analizzati nel capitolo successivo.

Capitolo 7

Risultati

Verranno ora presentati e commentati i risultati ottenuti tramite l'elaborazione dei dati ottenuti per mezzo delle simulazioni descritte nel capitolo 6.

Il comportamento di McMac nel caso vi sia un unico canale disponibile per le trasmissioni è assimilabile a quello descritto per XMAC nel capitolo 3, di conseguenza i risultati ottenuti nel caso di molteplici canali disponibili verranno di volta in volta confrontati col caso di un singolo canale, permettendo in questo modo di stimare il comportamento di McMac rispetto a XMAC.

7.1 Percentuale di pacchetti correttamente inviati

Come si può notare dai grafici nelle figure 7.1, 7.2, 7.3 la capacità della rete di assorbire il traffico generato aumenta consistentemente all'aumentare del numero dei canali disponibili. Risulta inoltre evidente come l'aumento del tempo medio che intercorre tra una richiesta di trasmissione e la successiva concorra ad aumentare la capacità di assorbimento della rete.

Si può dunque evincere, da questi primi risultati, che il protocollo McMac ottiene ampi miglioramenti prestazionali, dal punto di vista della capacità di portare a termine le richieste da parte del livello superiore, nel caso in cui la frequenza delle richieste di trasmissione sia inferiore alla durata del periodo di *duty cycling* dei nodi che implementano il protocollo.

Nelle figure dalla 7.4 alla 7.12 vengono messe a confronto le capacità delle tre varianti implementate di McMac al variare del numero di canali disponibili e del tempo medio di interarrivo delle richieste di trasmissione.
L'analisi dei risultati evidenzia come, dal punto di vista del numero di trasmissioni portate a termine, la versione originale di McMac risulti migliore delle due versioni successive nel caso in cui non sia disponibile sfruttare tutti i canali offerti da IEEE

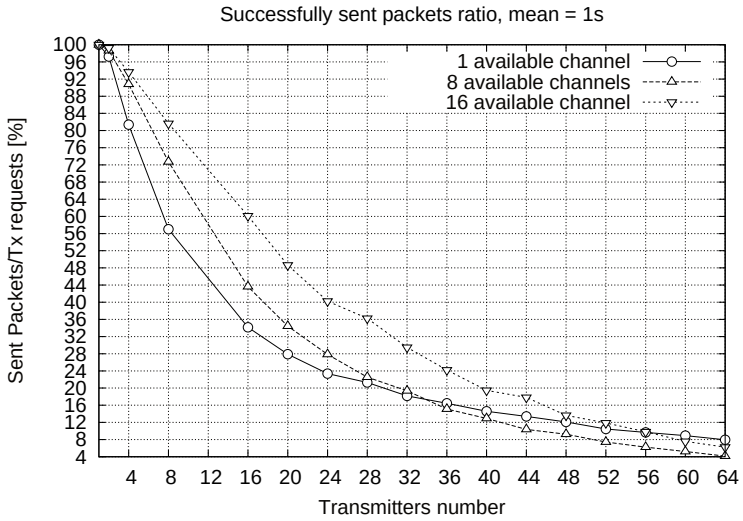


Figura 7.1: Percentuale di pacchetti correttamente inviati al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 1s

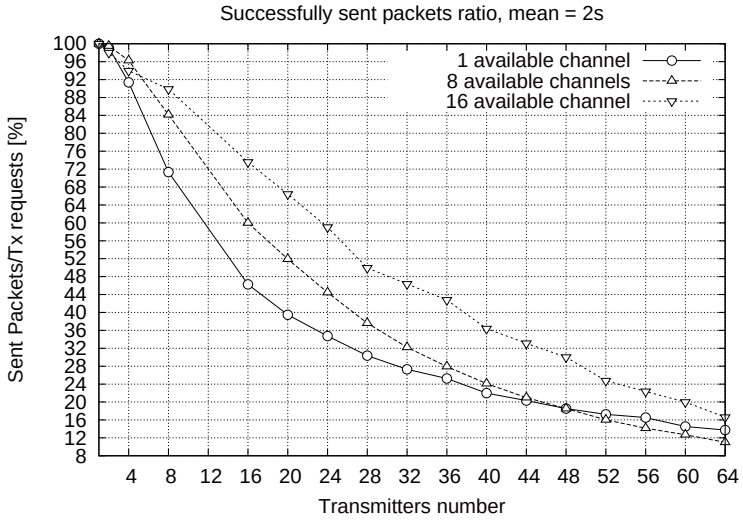


Figura 7.2: Percentuale di pacchetti correttamente inviati al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 2s

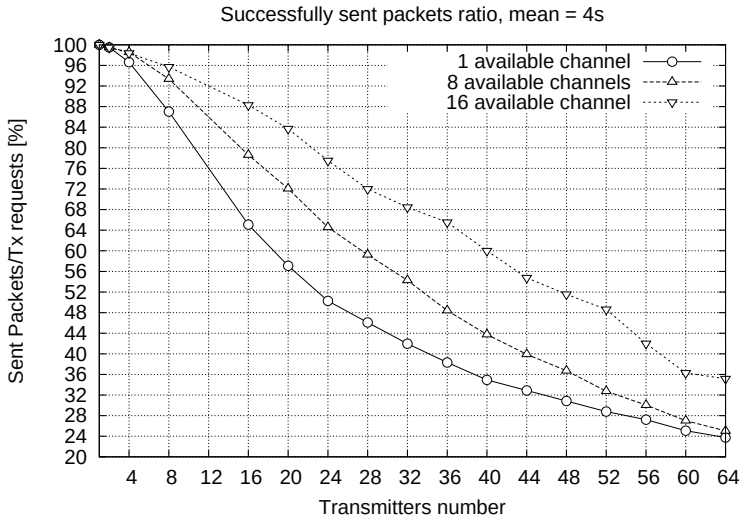


Figura 7.3: Percentuale di pacchetti correttamente inviati al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 4s

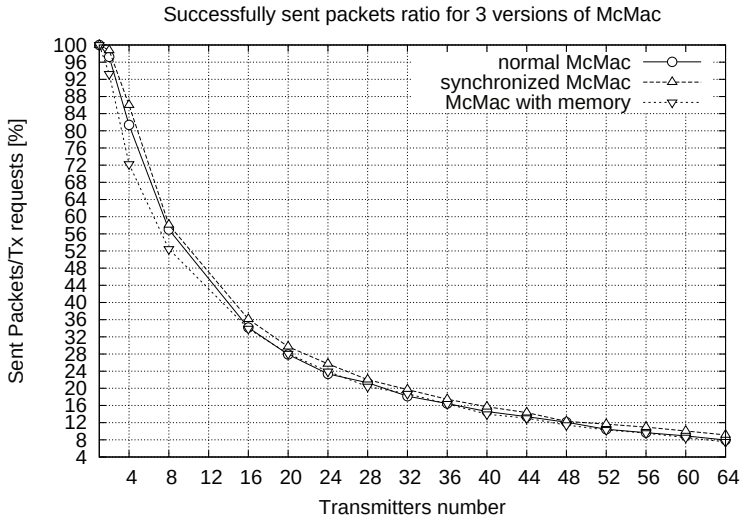


Figura 7.4: Percentuale di pacchetti correttamente inviati nelle tre varianti di McMac proposte, canali disponibili: 1, media: 1s

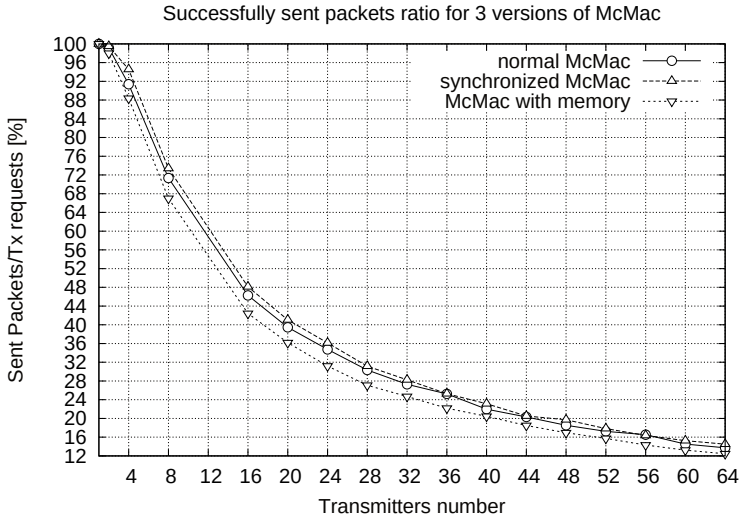


Figura 7.5: Percentuale di pacchetti correttamente inviati nelle tre varianti di McMac proposte, canali disponibili: 1, media: 2s

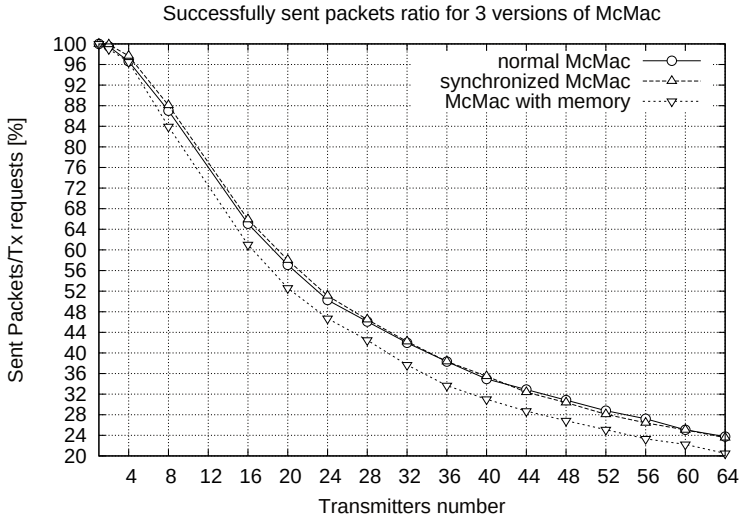


Figura 7.6: Percentuale di pacchetti correttamente inviati nelle tre varianti di McMac proposte, canali disponibili: 1, media: 4s

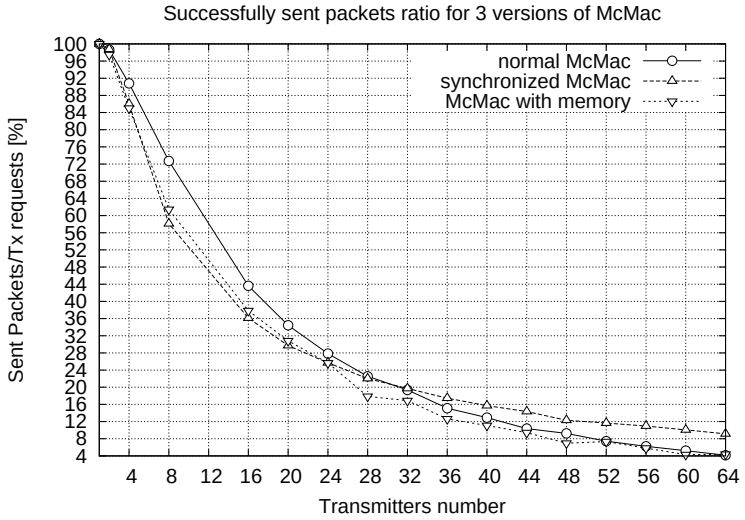


Figura 7.7: Percentuale di pacchetti correttamente inviati nelle tre varianti di McMac proposte, canali disponibili: 8, media: 1s

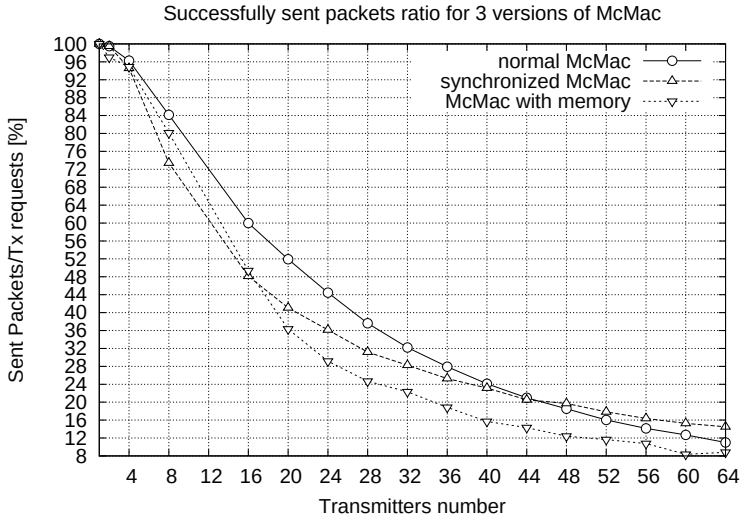


Figura 7.8: Percentuale di pacchetti correttamente inviati nelle tre varianti di McMac proposte, canali disponibili: 8, media: 2s

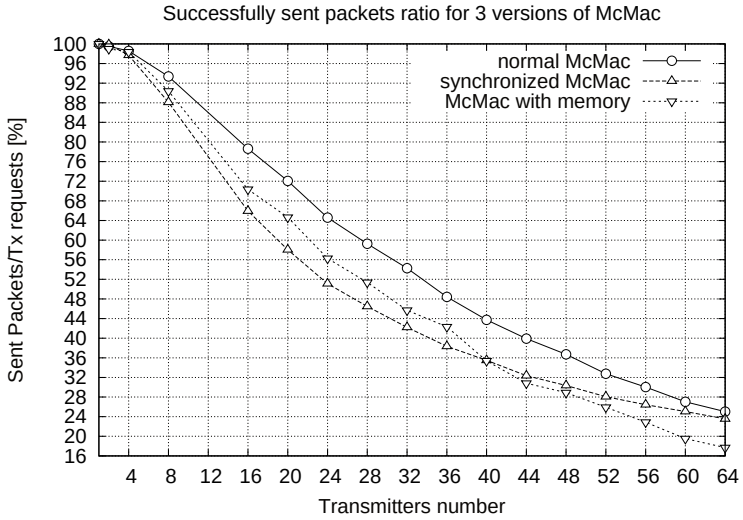


Figura 7.9: Percentuale di pacchetti correttamente inviati nelle tre varianti di McMac proposte, canali disponibili: 8, media: 4s

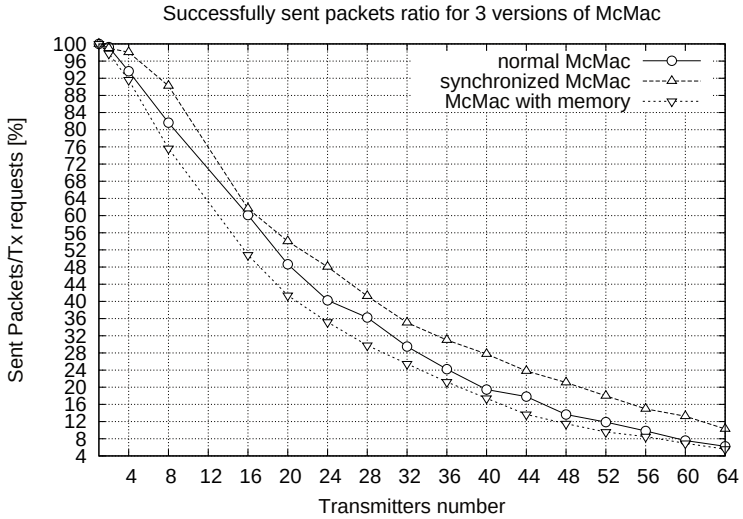


Figura 7.10: Percentuale di pacchetti correttamente inviati nelle tre varianti di McMac proposte, canali disponibili: 16, media: 1s

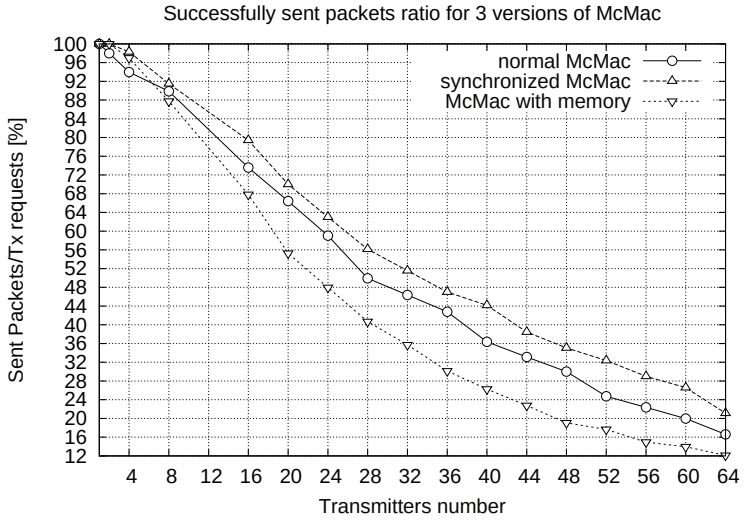


Figura 7.11: Percentuale di pacchetti correttamente inviati nelle tre varianti di McMac proposte, canali disponibili: 16, media: 2s

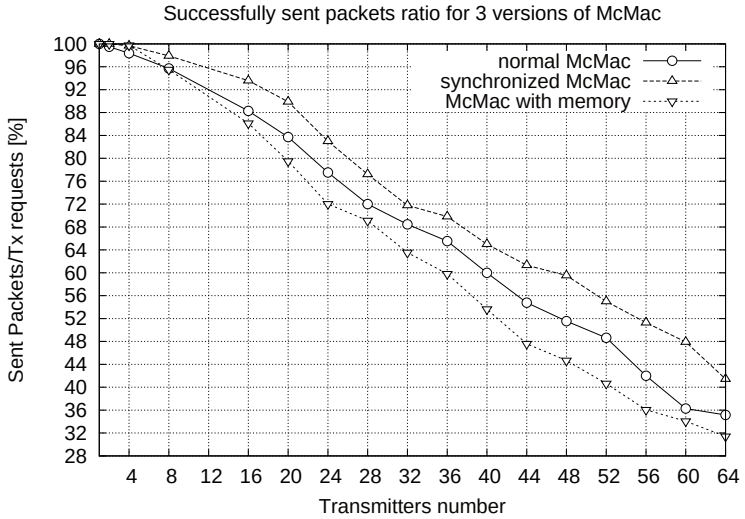


Figura 7.12: Percentuale di pacchetti correttamente inviati nelle tre varianti di McMac proposte, canali disponibili: 16, media: 4s

802.15.4 (si vedano figure 7.4, 7.5, 7.6, 7.7, 7.8, 7.9,), al contrario, nel caso in cui siano disponibili tutti i sedici canali (si vedano figure 7.10, 7.11, 7.12), la versione di McMac in cui è stata implementata la sincronizzazione dei *duty cycle* dimostra una maggiore capacità di servire le richieste di trasmissione. Questo fatto è dovuto principalmente alla minore occupazione del canale per ogni singola trasmissione, fattore questo che comporta principalmente due vantaggi:

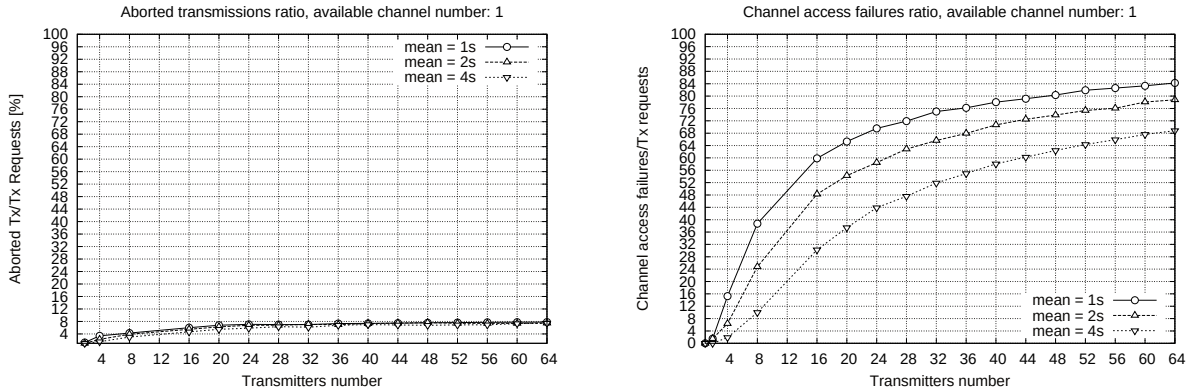
1. Un minor numero di fallimenti nell'accesso al canale;
2. Una maggiore probabilità di trovare il nodo a cui si desidera trasmettere sul relativo *base channel* grazie alla minore durata di ciascuna trasmissione.

Questi due vantaggi contribuiscono a diminuire il numero di eventi di *expired transaction* (dovuto al fatto di non trovare il nodo destinatario sul relativo canale base perché esso stesso è impegnato in una trasmissione su un altro canale ancora, questo evento comporta un notevole dispendio energetico dal momento che costringe il trasmettitore ad esaurire interamente la trasmissione degli *strobed preamble* prima di poter cancellare la trasmissione) e di *channel access failure* (causato dall'eccedere il numero massimo di periodi di *backoff* previsto dall'algoritmo CSMA-CA a causa di una forte occupazione del canale).
Per quanto riguarda, invece, la versione di McMac dotata di memoria, essa non migliora la capacità di assorbimento della rete a causa dell'aggressività della politica di trasmissione, un nodo, infatti, una volta trovato il canale libero, tende ad occuparlo per lungo tempo causando in tutti gli altri nodi che vogliano trasmettere sullo stesso canale un aumento dei fallimenti nell'accesso al canale.

7.2 Percentuale di trasmissioni fallite

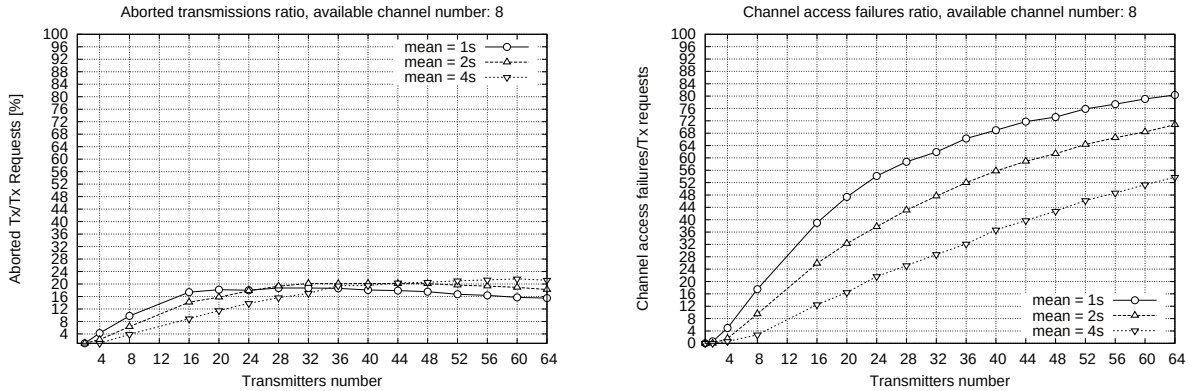
Una trasmissione può fallire per due motivi:

1. *Expired transaction*: questo tipo di fallimento avviene quando si cerca di trasmettere ad un nodo che è già impegnato in una trasmissione su un canale diverso rispetto al relativo *base channel*. Quando si verifica questa situazione il nodo mittente è costretto a trasmettere per intero gli *strobed preamble* prima di poter terminare la trasmissione e notificare il fallimento, questo caso dunque risulta in due principali inconvenienti:
   (a) Un notevole dispendio energetico per il nodo mittente causato dall'inutile invio di preamboli;
   (b) L'occupazione del canale per un tempo pari al tempo di *sleep* previsto dal periodo di *duty cycle* preimpostato.



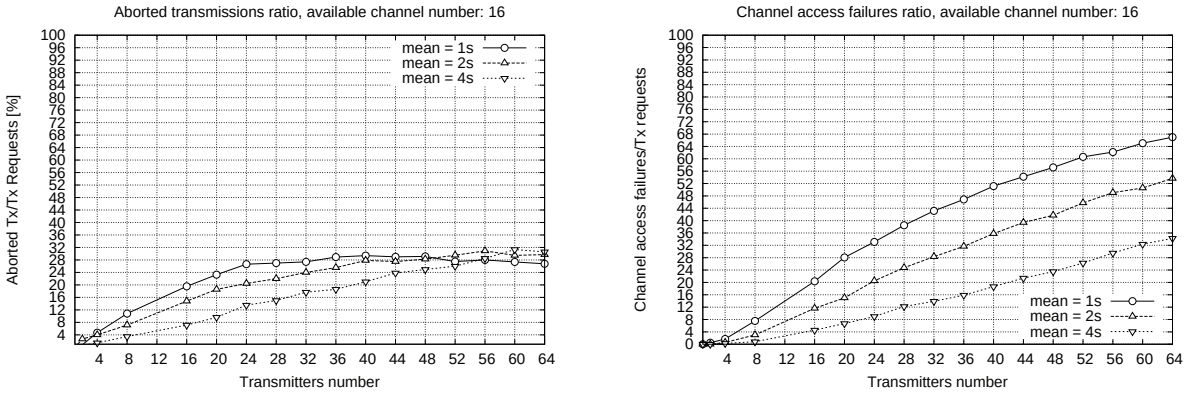
(a) aborted (b) failure

Figura 7.13: Percentuale di *expired transaction* (a) e di *channel access failure* (b) al variare del tempo medio di interarrivo delle richieste di trasmissione con 1 canale disponibile, McMac originale



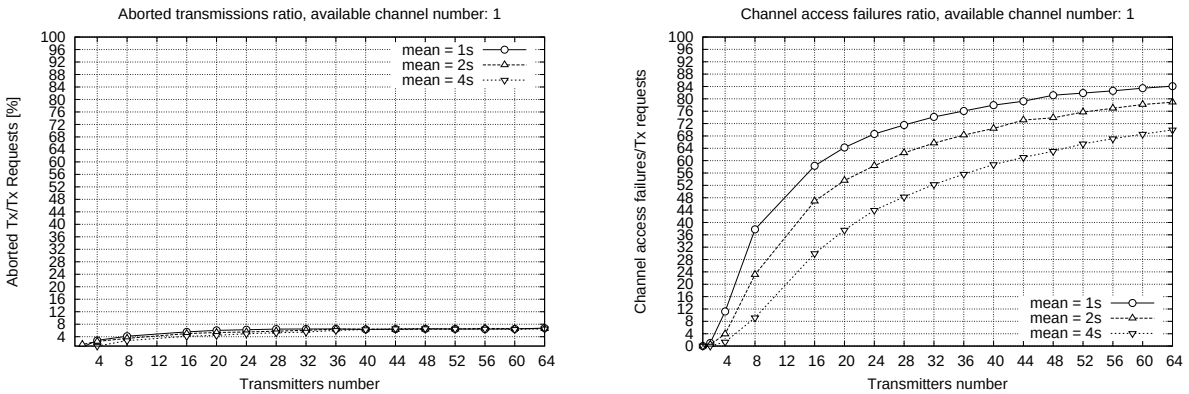
(a) aborted (b) failure

Figura 7.14: Percentuale di *expired transaction* (a) e di *channel access failure* (b) al variare del tempo medio di interarrivo delle richieste di trasmissione con 8 canali disponibili, McMac originale



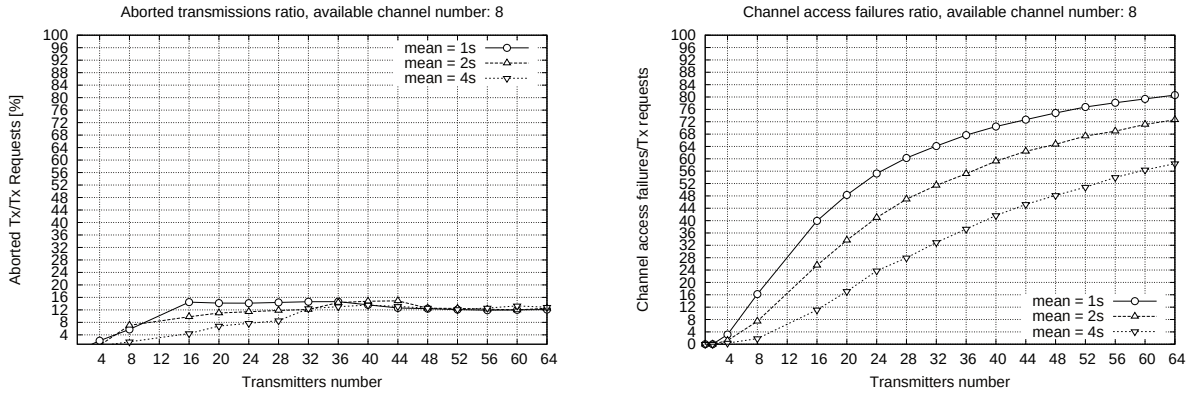
(a) aborted (b) failure

Figura 7.15: Percentuale di *expired transaction* (a) e di *channel access failure* (b) al variare del tempo medio di interarrivo delle richieste di trasmissione con 16 canali disponibili, McMac originale



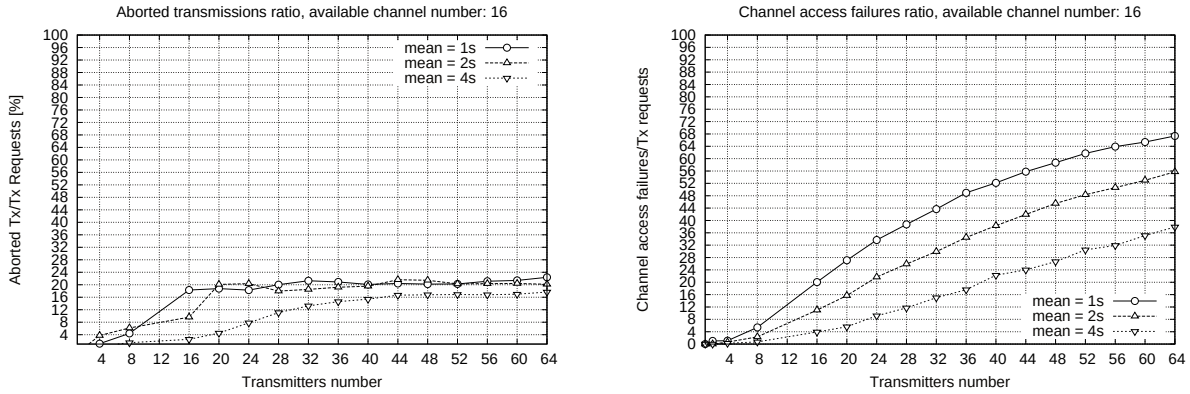
(a) aborted (b) failure

Figura 7.16: Percentuale di *expired transaction* (a) e di *channel access failure* (b) al variare del tempo medio di interarrivo delle richieste di trasmissione con 1 canale disponibile, McMac con sincronizzazione rispetto al *duty cycle* del destinatario



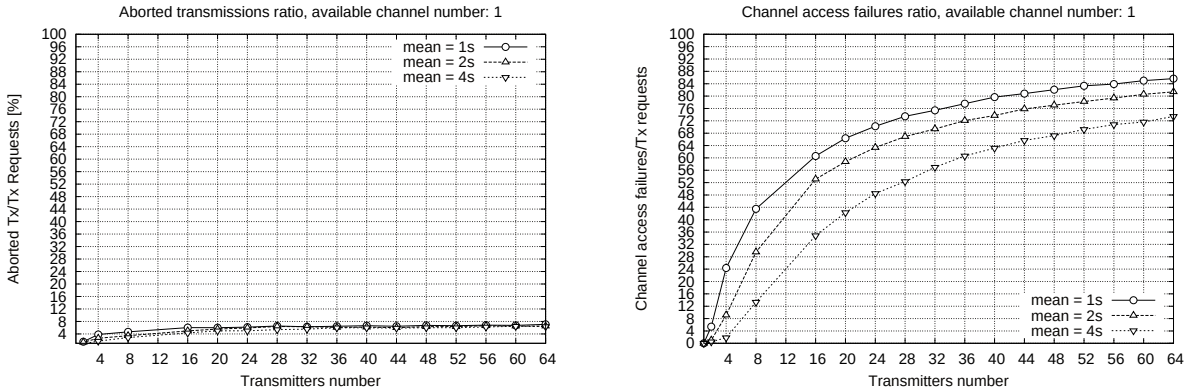
(a) aborted (b) failure

Figura 7.17: Percentuale di *expired transaction* (a) e di *channel access failure* (b) al variare del tempo medio di interarrivo delle richieste di trasmissione con 8 canali disponibili, McMac con sincronizzazione rispetto al *duty cycle* del destinatario



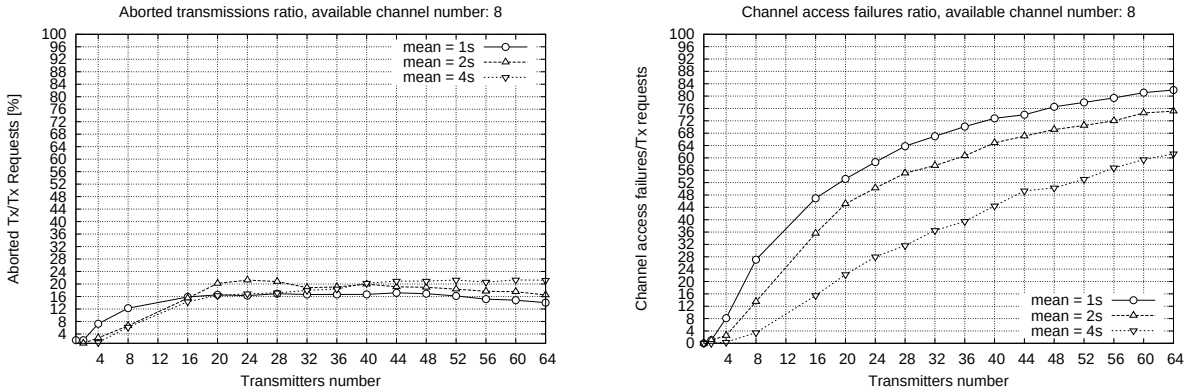
(a) aborted (b) failure

Figura 7.18: Percentuale di *expired transaction* (a) e di *channel access failure* (b) al variare del tempo medio di interarrivo delle richieste di trasmissione con 16 canali disponibili, McMac con sincronizzazione rispetto al *duty cycle* del destinatario



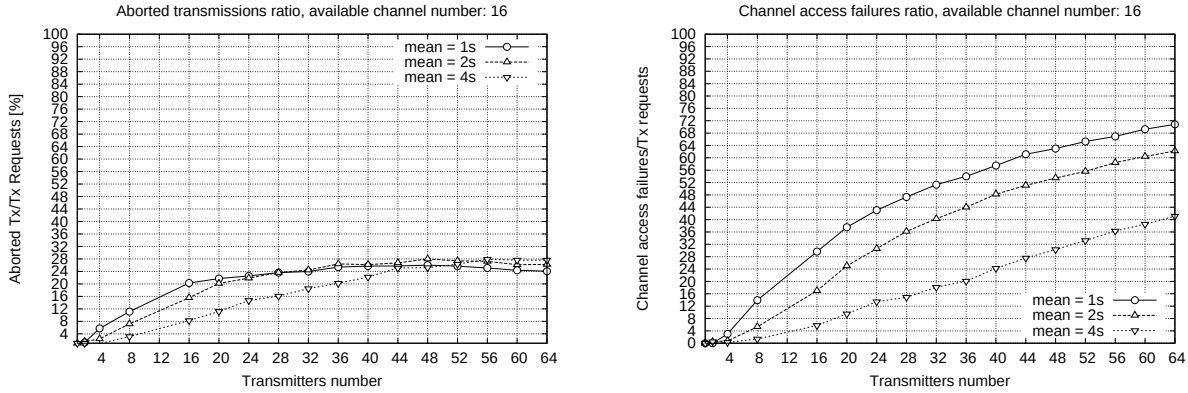
(a) aborted (b) failure

Figura 7.19: Percentuale di *expired transaction* (a) e di *channel access failure* (b) al variare del tempo medio di interarrivo delle richieste di trasmissione con 1 canale disponibile, McMac con memoria



(a) aborted (b) failure

Figura 7.20: Percentuale di *expired transaction* (a) e di *channel access failure* (b) al variare del tempo medio di interarrivo delle richieste di trasmissione con 8 canali disponibili, McMac con memoria



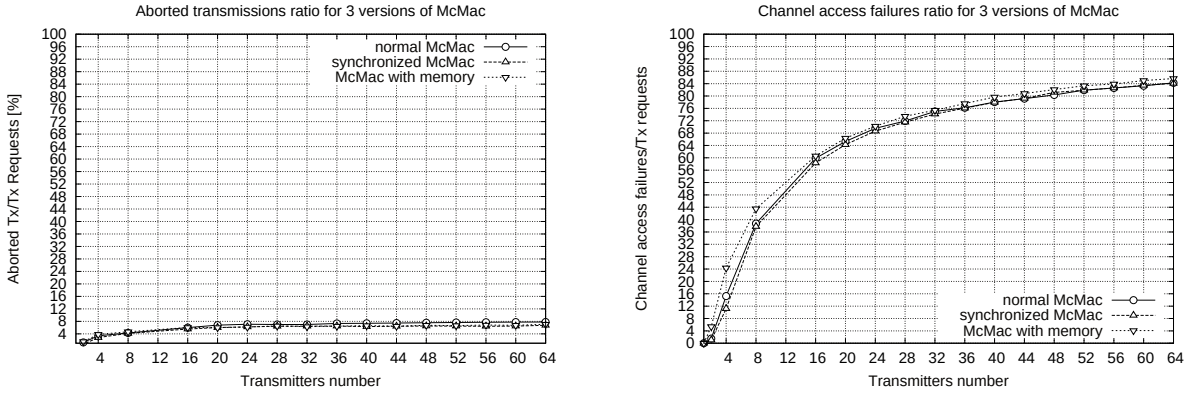
(a) aborted (b) failure

Figura 7.21: Percentuale di *expired transaction* (a) e di *channel access failure* (b) al variare del tempo medio di interarrivo delle richieste di trasmissione con 16 canali disponibili, McMac con memoria

2. *Channel access failure*: in questo caso viene superato il numero massimo di periodi di *backoff* previsto dall'algoritmo CSMA-CA senza mai trovare il canale libero. Questo evento ha meno impatto sulle prestazioni globali della rete rispetto al caso precedente, dal momento che affligge unicamente il nodo che desidera trasmettere facendo aumentare il tempo trascorso in ricezione senza, però, causare ulteriore occupazione del canale.

Le figure 7.13, 7.14, 7.15, mettono a confronto la percentuale di trasmissioni fallite per *transaction expired* con quella di trasmissioni fallite per *channel access failure* rispetto alla versione originale di McMac; un risultato desiderabile, per quanto detto sopra, è (oltre, ovviamente a limitare l'occorrenza di entrambi questi eventi) che le trasmissioni falliscano prevalentemente a causa di un errore nell'accesso al canale.

Osservando i grafici, si può notare come, mentre la percentuale di *channel access failure* diminuisca sensibilmente all'aumentare del numero di canali disponibili (si confrontino ad esempio le figure refab-fail-normal-1(b) e 7.15(b)), la percentuale di *expired transaction* aumenti all'aumentare del numero di canali. Questo effetto è congenito all'idea di sfruttare parallelamente più canali di trasmissione ed è la principale contropartita della possibilità di avviare parallelamente molteplici trasmissioni. La sordità di un nodo nei confronti di ciò che avviene in canali diversi da quello su cui è correntemente posizionato induce, all'aumentare del numero di nodi abilitati a trasmettere, una maggiore probabilità di trovare il nodo destinatario posizionato sul proprio *base channel* e di conseguenza un aumento del numero di *expired transaction* rispetto ad una con-



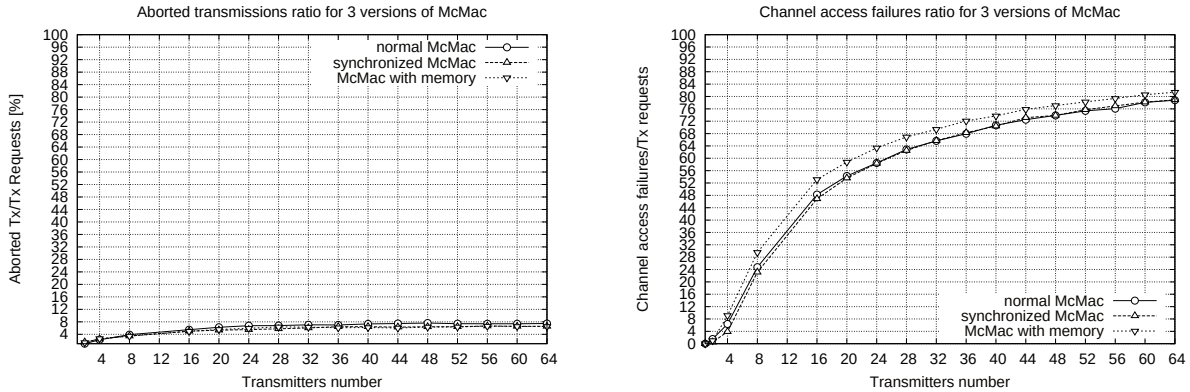
(a) aborted (b) failure

Figura 7.22: Percentuale di *expired transaction* (a) e di *channel access failure* (b) nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s

testuale diminuzione del numero di *channel access failure* dovuta alla ripartizione del carico trasmissivo su di uno spettro più ampio.

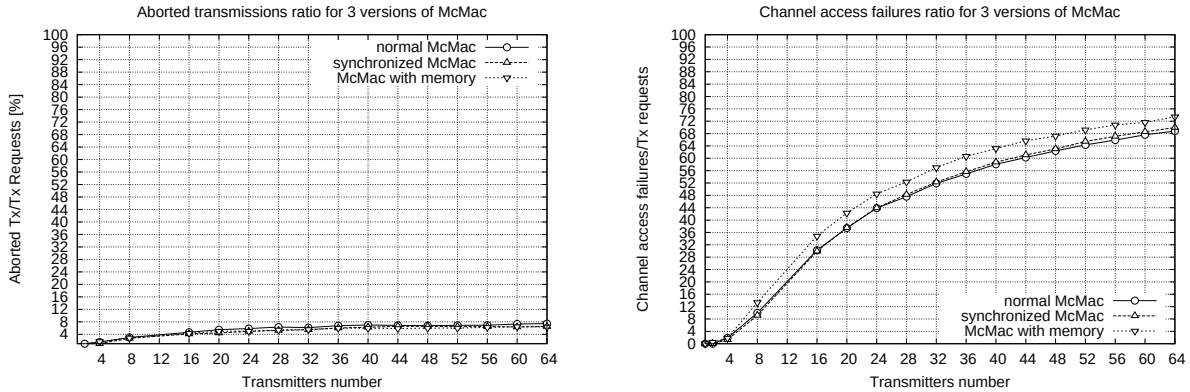
Le figure da 7.22 a 7.28, mettendo a confronto le prestazioni delle tre varianti di McMac presentate in questo lavoro al variare del numero di canali disponibili e del tempo medio di interarrivo delle richieste di trasmissione, mostrano come l'introduzione del processo di sincronizzazione permetta di abbassare, in alcuni casi consistentemente, la percentuale di trasmissioni fallite a causa di *transaction expired* mantenendo al contempo pressoché costante la percentuale di fallimenti nell'accesso al canale.
La versione dotata di memoria, invece si dimostra meno performante anche su questo versante dal momento che, pur mantenendo sostanzialmente inalterata la percentuale di *transaction expired* rispetto alla versione originale del protocollo, contribuisce ad un sensibile aumento della percentuale di *channel access failure* dovuto all'aggressività nell'occupazione del canale.
Le figure 7.28, 7.29, 7.30 evidenziano come, avendo a disposizione lo spettro IEEE 802.15.4 nella sua interezza, la sincronizzazione sul *duty cycle* del destinatario permetta di ridurre la percentuale di *expired transaction*, nel caso di forte congestione della rete, di valori che vanno dal 12% fino al 16%.

I risultati finora presentati portano a concludere che le prestazioni della rete possono essere migliorate progettando politiche che permettano di diminuire il numero di *expired transaction*. Muovendosi in questa direzione si sta sviluppando una nuova variante del protocollo McMac che permetta ai nodi, durante un eventuale periodo di *backoff*, di ritornare all'ascolto del proprio *base channel* in



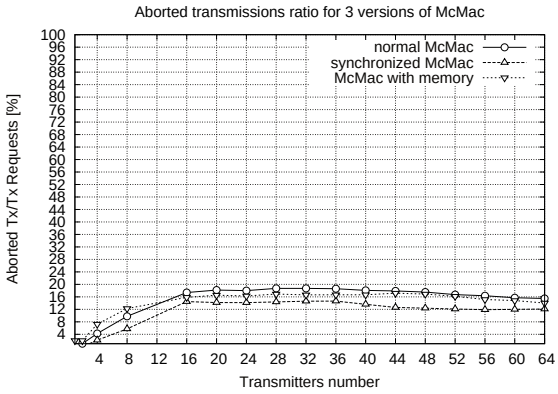
(a) aborted (b) failure

Figura 7.23: Percentuale di *expired transaction* (a) e di *channel access failure* (b) nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 2s

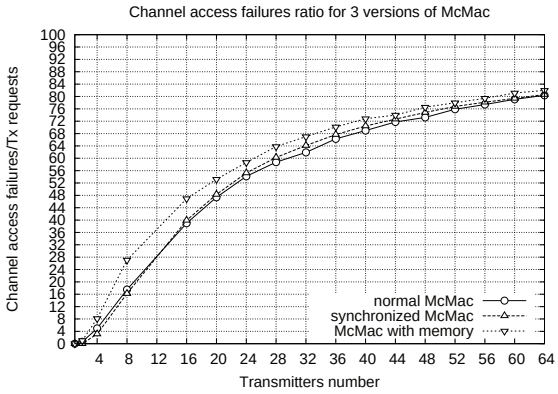


(a) aborted (b) failure

Figura 7.24: Percentuale di *expired transaction* (a) e di *channel access failure* (b) nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 4s

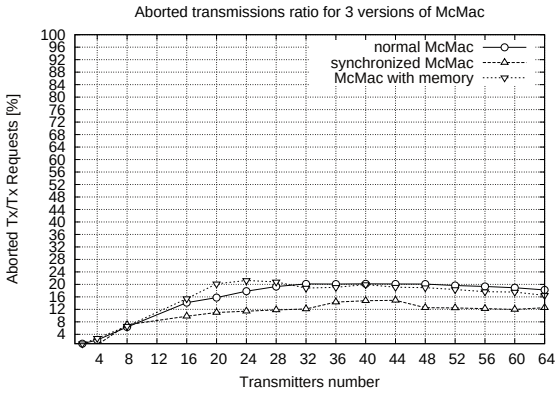


(a) aborted

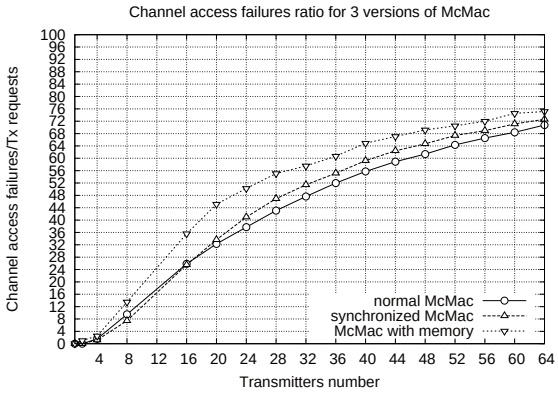


(b) failure

Figura 7.25: Percentuale di *expired transaction* (a) e di *channel access failure* (b) nelle tre varianti di McMac con numero di canali disponibili: 8 e media: 1s

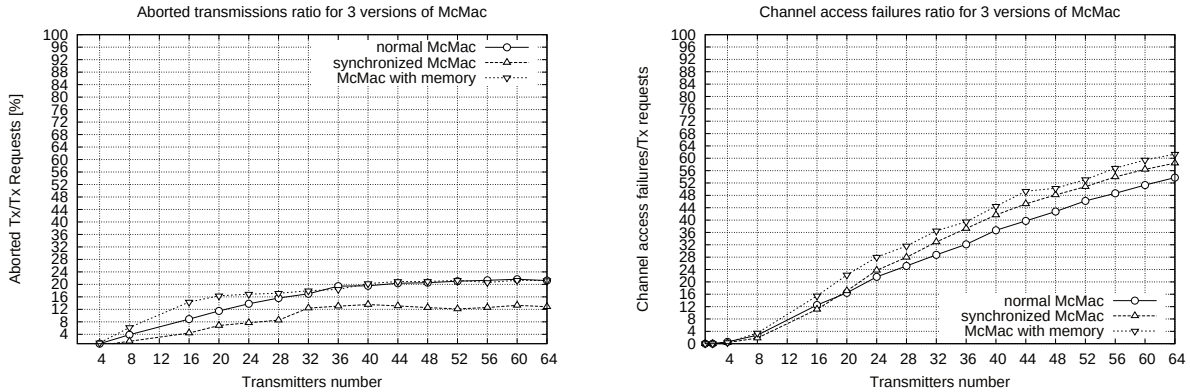


(a) aborted



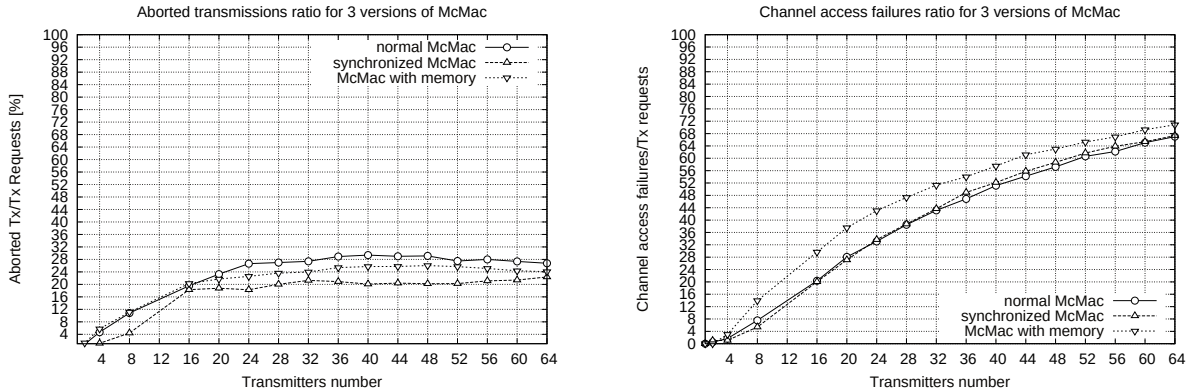
(b) failure

Figura 7.26: Percentuale di *expired transaction* (a) e di *channel access failure* (b) nelle tre varianti di McMac con numero di canali disponibili: 8 e media: 2s



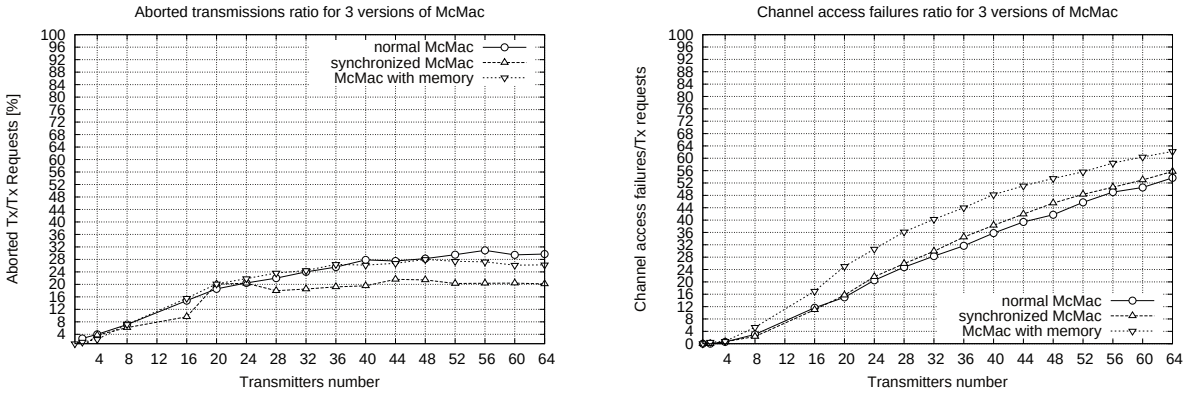
(a) aborted (b) failure

Figura 7.27: Percentuale di *expired transaction* (a) e di *channel access failure* (b) nelle tre varianti di McMac con numero di canali disponibili: 8 e media: 4s



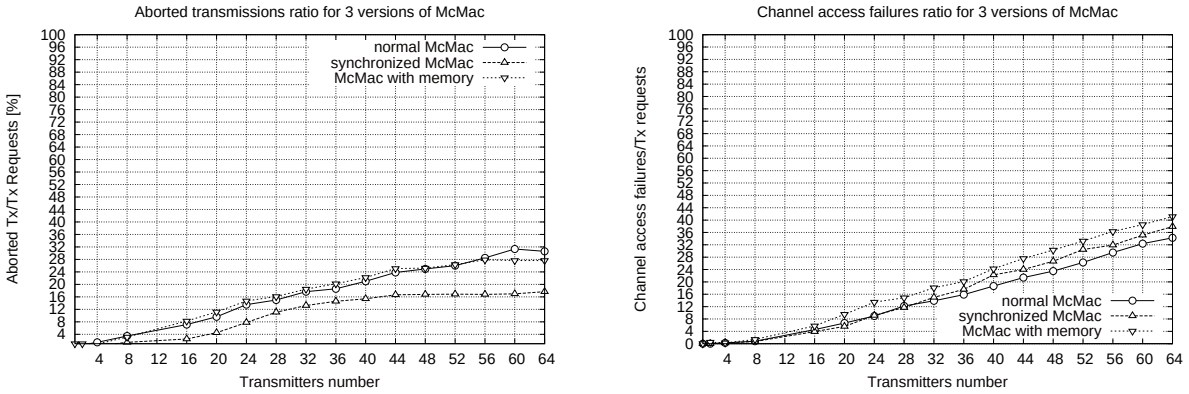
(a) aborted (b) failure

Figura 7.28: Percentuale di *expired transaction* (a) e di *channel access failure* (b) nelle tre varianti di McMac con numero di canali disponibili: 16 e media: 1s



(a) aborted (b) failure

Figura 7.29: Percentuale di *expired transaction* (a) e di *channel access failure* (b) nelle tre varianti di McMac con numero di canali disponibili: 16 e media: 2s



(a) aborted (b) failure

Figura 7.30: Percentuale di *expired transaction* (a) e di *channel access failure* (b) nelle tre varianti di McMac con numero di canali disponibili: 16 e media: 4s

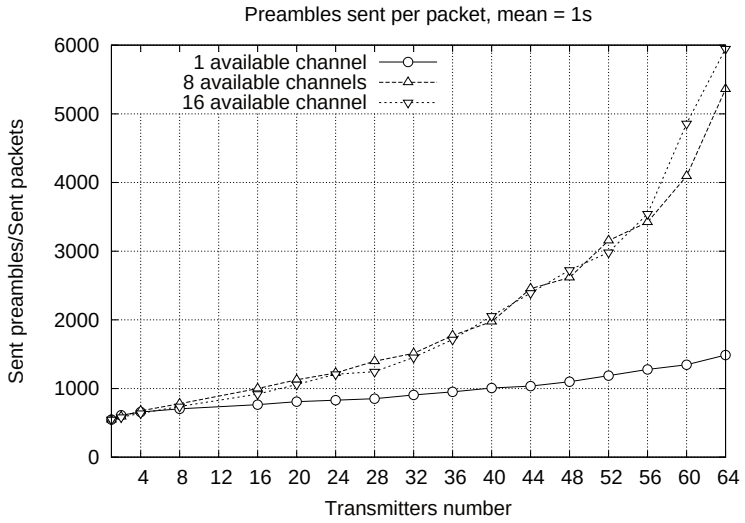


Figura 7.31: Numero di preamboli trasmessi per pacchetto al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 1s

modo da aumentare la capacità di gestire eventuali ricezioni anche se impegnati a servire una richiesta di trasmissione.

7.3 Overhead di comunicazione dovuto alla trasmissione degli *strobed preamble*

Si presentano ora i dati relativi all'*overhead* di comunicazione dovuto alla necessità di inviare la sequenza di *strobed preamble* al fine di interrompere il *duty cycle* del destinatario.

Nella versione originale di McMac, consistentemente con quanto descritto nelle sezioni precedenti, si nota (si vedano le figure 7.31, 7.32, 7.33) l'aumentare del numero di preamboli tramessi per pacchetto all'aumentare del numero di canali disponibili. Questo fatto accade a causa dell'aumentare del numero di eventi del tipo *transaction expired* che costringono a trasmettere, inutilmente, l'intera sequenza di preamboli. Le due varianti di McMac presentate in precedenza sono state sviluppate prevalentemente per ovviare al problema dell'*overhead* che, come si può notare, diventa sempre più consistente all'aumentare del numero di nodi abilitati a trasmettere (e di conseguenza all'aumentare dei nodi con la capacità di cambiare la propria frequenza portante per avviare una trasmissione verso un dispositivo con diverso *base channel*).

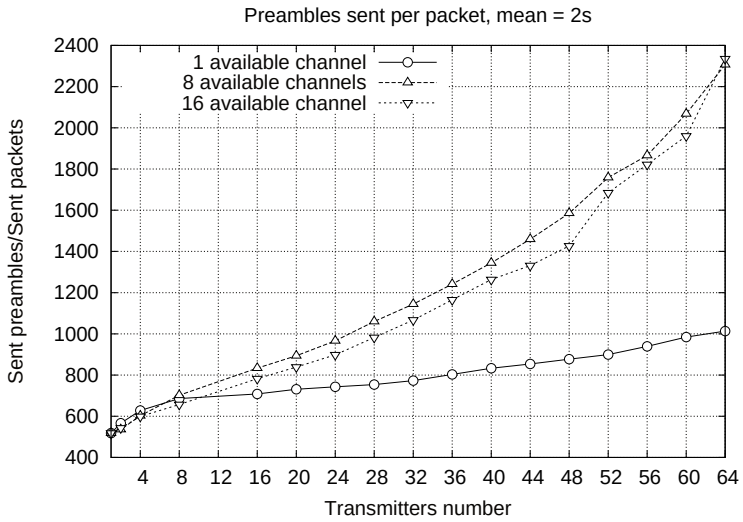


Figura 7.32: Numero di preamboli trasmessi per pacchetto al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 2s

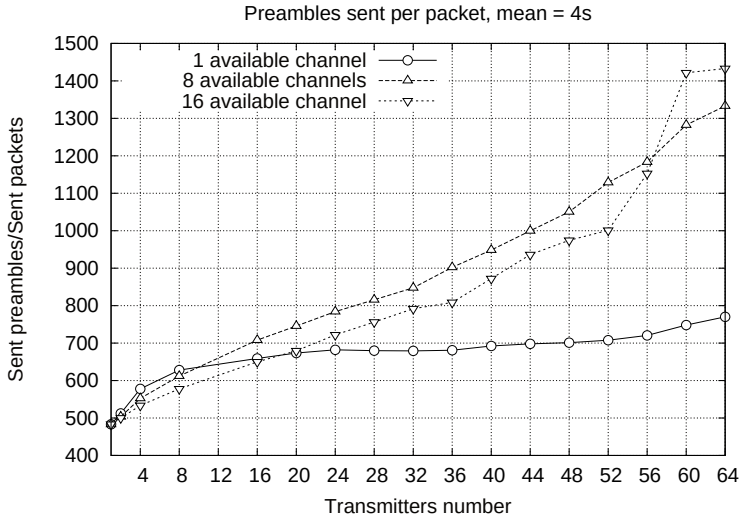


Figura 7.33: Numero di preamboli trasmessi per pacchetto al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 4s

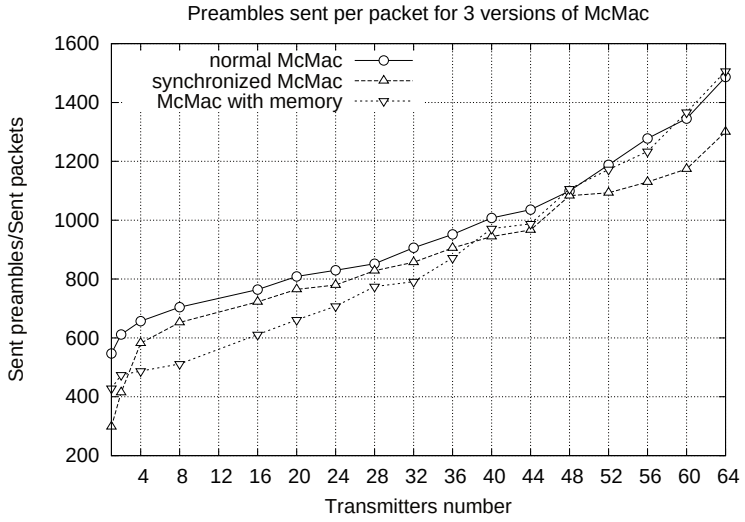


Figura 7.34: Numero di preamboli trasmessi per pacchetto nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s

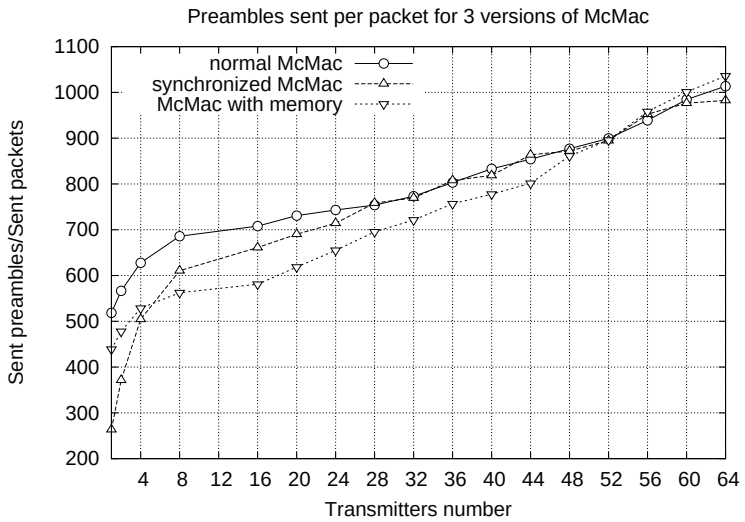


Figura 7.35: Numero di preamboli trasmessi per pacchetto nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 2s

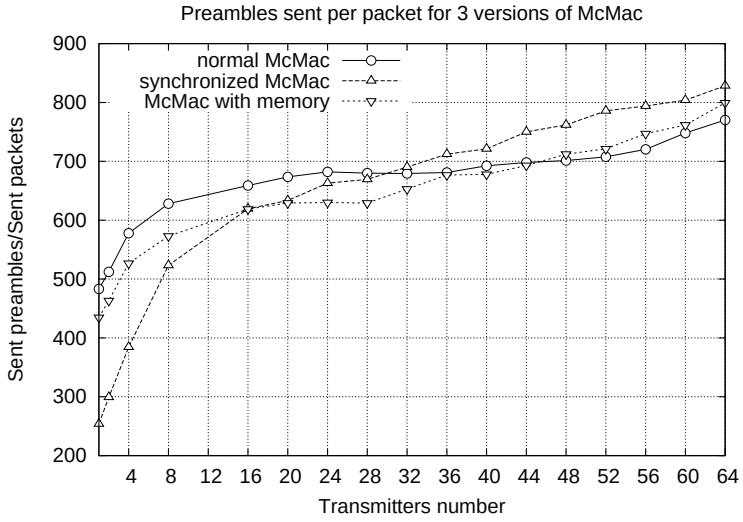


Figura 7.36: Numero di preamboli trasmessi per pacchetto nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 4s

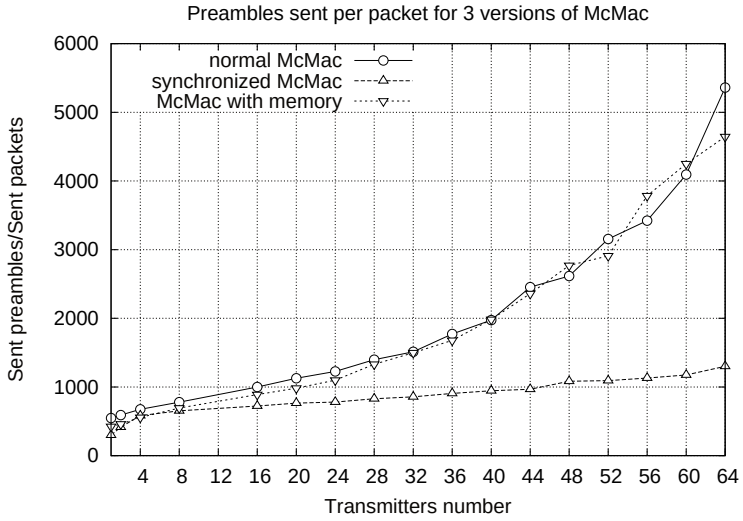


Figura 7.37: Numero di preamboli trasmessi per pacchetto nelle tre varianti di McMac con numero di canali disponibili: 8 e media: 1s

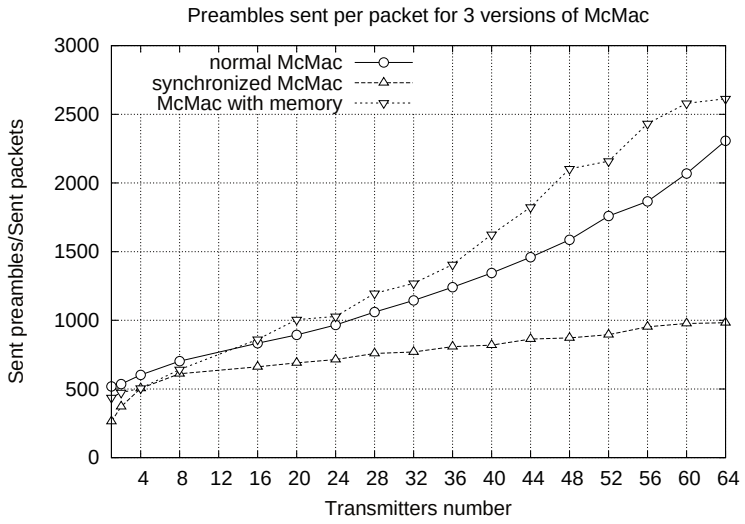


Figura 7.38: Numero di preamboli trasmessi per pacchetto nelle tre varianti di McMac con numero di canali disponibili: 8 e media: 2s

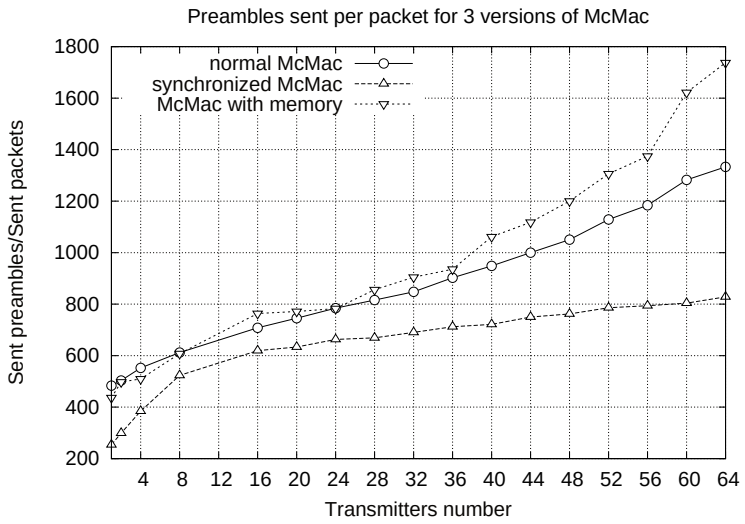


Figura 7.39: Numero di preamboli trasmessi per pacchetto nelle tre varianti di McMac con numero di canali disponibili: 8 e media: 4s

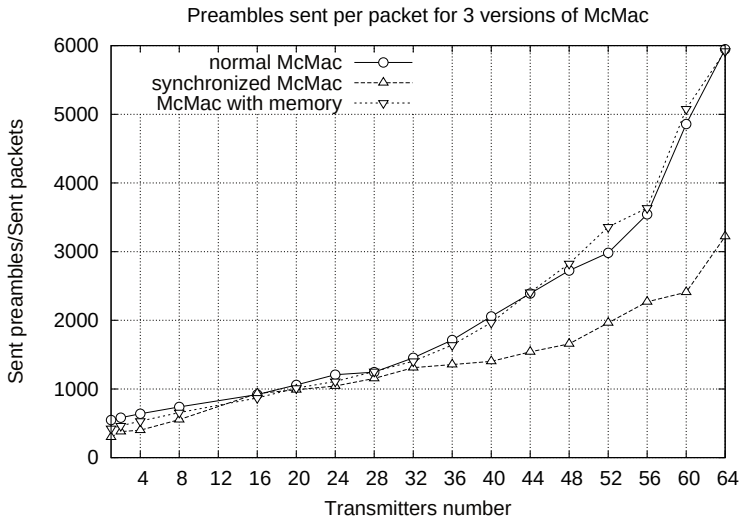


Figura 7.40: Numero di preamboli trasmessi per pacchetto nelle tre varianti di McMac con numero di canali disponibili: 16 e media: 1s

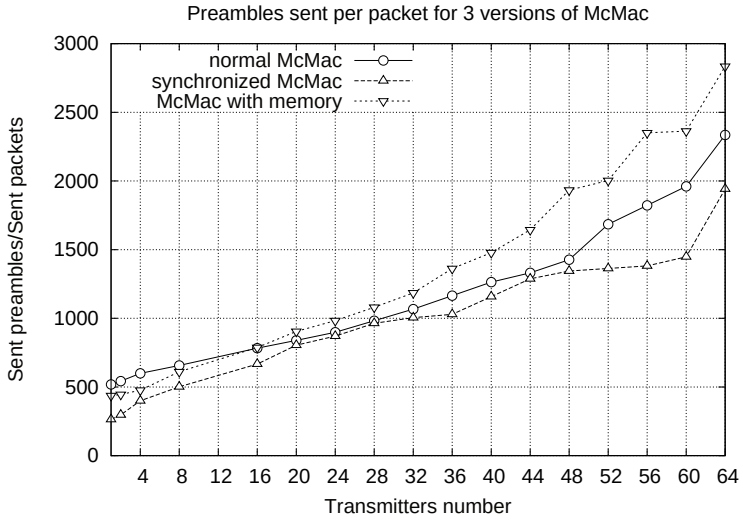


Figura 7.41: Numero di preamboli trasmessi per pacchetto nelle tre varianti di McMac con numero di canali disponibili: 16 e media: 2s

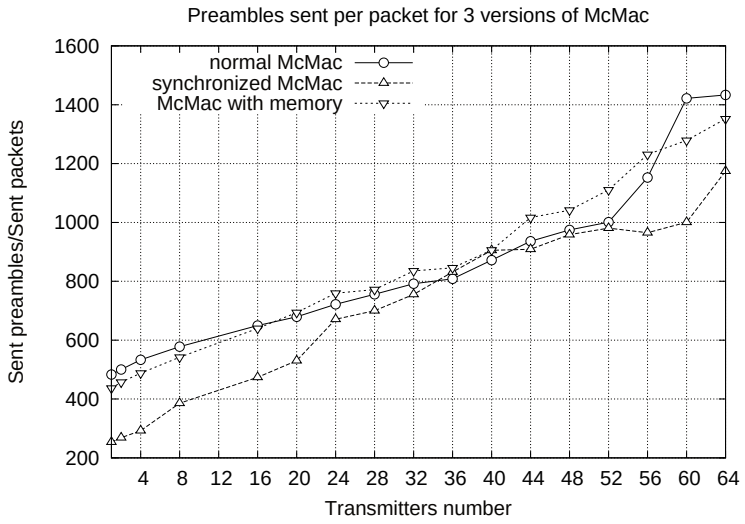


Figura 7.42: Numero di preamboli trasmessi per pacchetto nelle tre varianti di McMac con numero di canali disponibili: 16 e media: 4s

Il confronto tra le tre varianti di McMac implementate evidenzia come l'introduzione della sincronizzazione del trasmettitore con il *duty cycle* del destinatario permetta di ridurre consistentemente il numero di preamboli tramessi per pacchetto, contribuendo in questo modo a ridurre notevolmente il carico del canale dovuto alla trasmissione dei preamboli e, di conseguenza, permettendo di migliorare l'efficienza energetica ottenuta dai nodi tramite l'utilizzo di questa versione del protocollo.
Ancora una volta la versione di McMac dotata di memoria non evidenzia miglioramenti sensibili rispetto all'originale.

Altro parametro di interesse è il tempo medio necessario per la corretta trasmissione di un pacchetto, questi risultati verranno presentati nella sezione successiva.

7.4 Tempo medio necessario all'invio di un pacchetto

Le figure 7.43, 7.45, 7.45 presentano i dati relativi al tempo medio necessario al corretto invio di un pacchetto, coerentemente con le aspettative, quest'ultimo si attesta intorno al mezzo secondo.
Questi risultati sono da intendersi come tempo medio che intercorre tra il verificarsi di una richiesta di trasmissione e la relativa conferma di avvenuto invio del pacchetto. Il confronto tra le prestazioni in termini temporali tra la versione

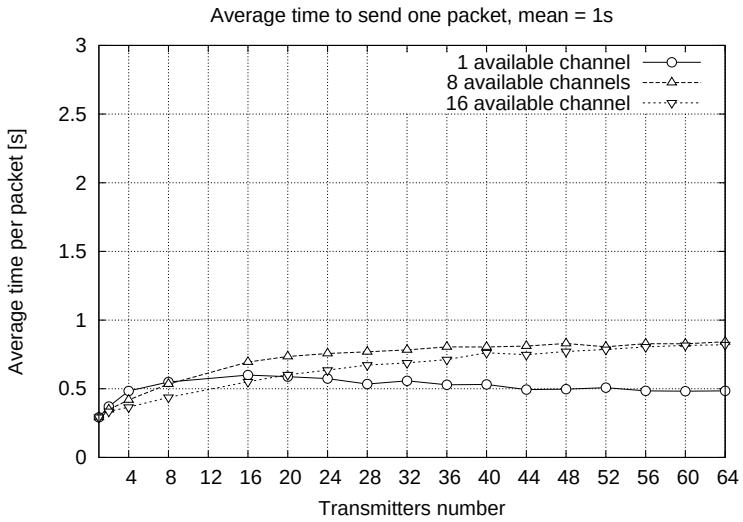


Figura 7.43: Tempo medio necessario alla trasmissione di un pacchetto al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 1s

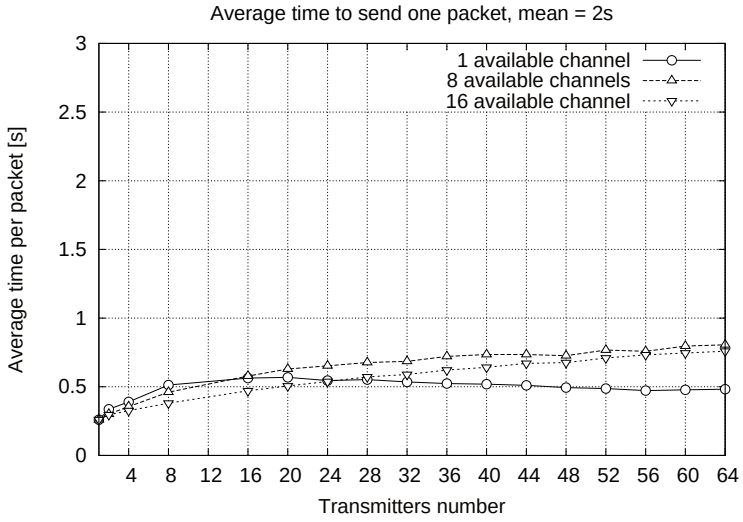


Figura 7.44: Tempo medio necessario alla trasmissione di un pacchetto al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 2s

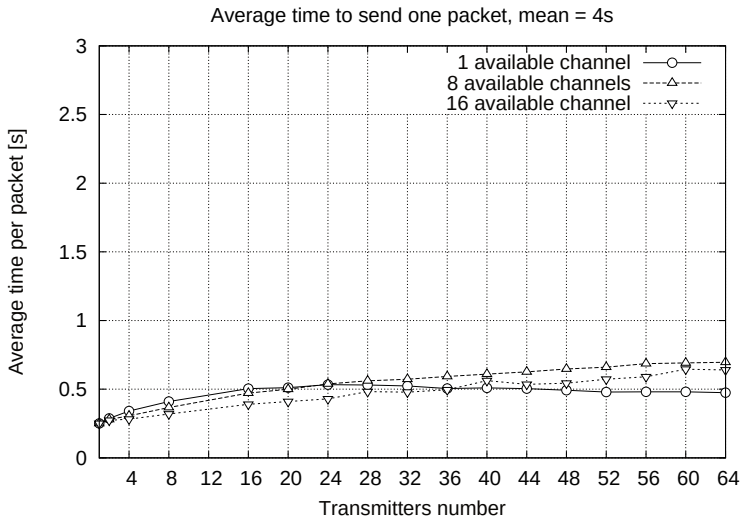


Figura 7.45: Tempo medio necessario alla trasmissione di un pacchetto al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 4s

originale di McMac e quella dotata della capacità di sincronizzazione con il *duty cycle* del destinatario evidenzia come, nel caso di quest'ultima, i tempi di attesa per la conferma di avvenuto invia siano leggermente più alti. Questo è dovuto al fatto che spesso le richieste di trasmissione vengono ritardate al fine di ottenere come risultato l'invio del minor numero possibile di preamboli. L'introduzione di questo ritardo, tuttavia, può, nel caso in cui la sincronizzazione venga perduta, risultare in un tempo di trasmissione più lungo di quanto strettamente necessario a causa del ritardo introdotto sulla base di una stima del *drift* tra il *duty cycle* del ricevitore e quello del destinatario non più valida.

7.5 Risparmio energetico

Il risparmio energetico relativo all'introduzione del *duty cycle* e della tecnica degli *strobed preamble* viene valutato come rapporto tra il tempo trascorso in uno stato di attività (trasmissione o ricezione) e il tempo totale di simulazione.
I risultati relativi alla quantità sopra descritta sono presentati, per quanto riguarda la versione originale del protocollo, nelle figure 7.55, 7.56, 7.57 e sono stati calcolati prendendo come riferimento le quantità relative ai nodi abilitati alla trasmissione. L'analisi dei risultati presentati nei grafici in figura 7.55, 7.56, 7.57, porge due conclusioni principali:

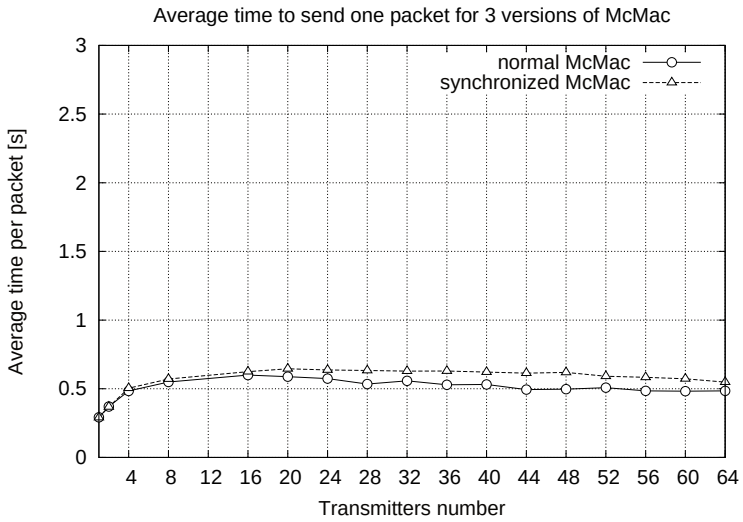


Figura 7.46: Tempo medio necessario alla trasmissione di un pacchetto nella versione originale e sincronizzata di McMac con numero di canali disponibili: 1 e media: 1s

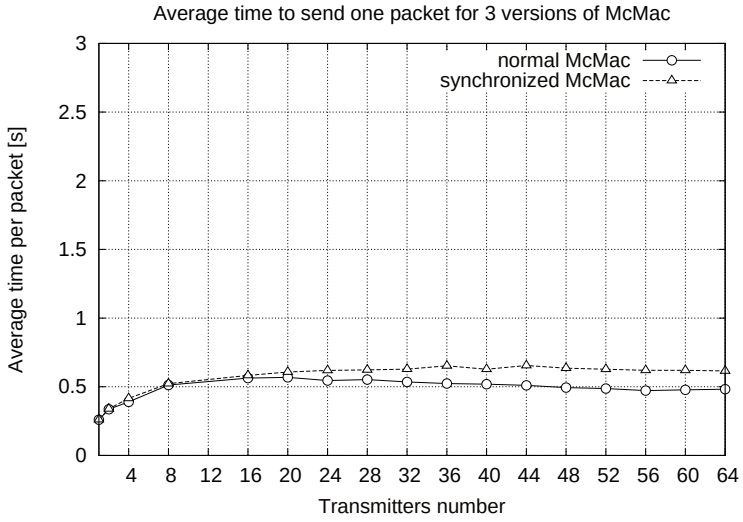


Figura 7.47: Tempo medio necessario alla trasmissione di un pacchetto nella versione originale e sincronizzata di McMac con numero di canali disponibili: 1 e media: 2s

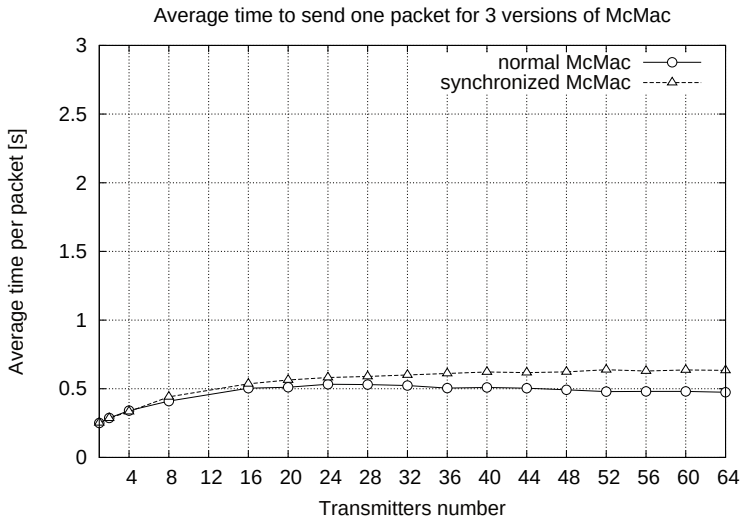


Figura 7.48: Tempo medio necessario alla trasmissione di un pacchetto nella versione originale e sincronizzata di McMac con numero di canali disponibili: 1 e media: 4s

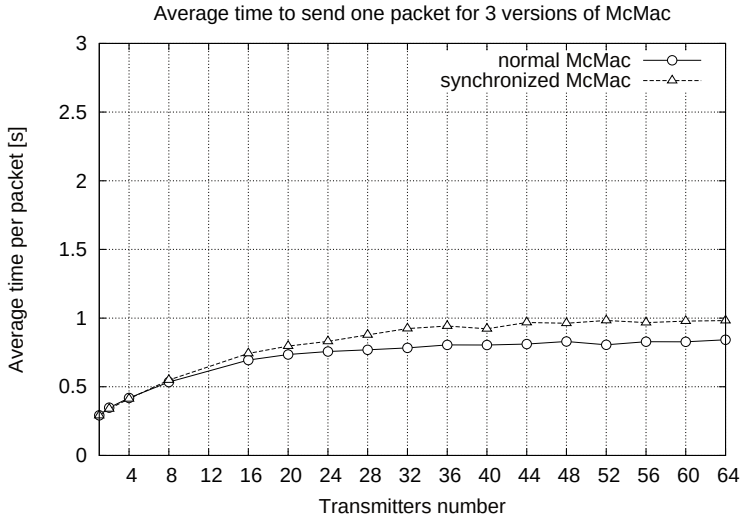


Figura 7.49: Tempo medio necessario alla trasmissione di un pacchetto nella versione originale e sincronizzata di McMac con numero di canali disponibili: 8 e media: 1s

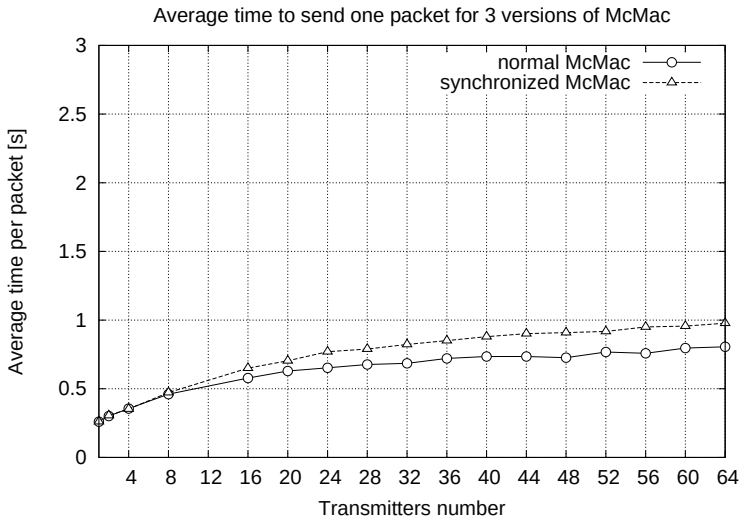


Figura 7.50: Tempo medio necessario alla trasmissione di un pacchetto nella versione originale e sincronizzata di McMac con numero di canali disponibili: 8 e media: 2s

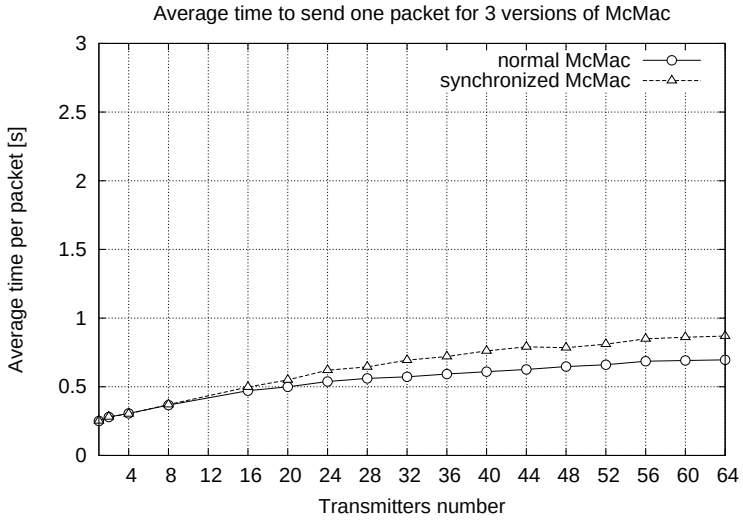


Figura 7.51: Tempo medio necessario alla trasmissione di un pacchetto nella versione originale e sincronizzata di McMac con numero di canali disponibili: 8 e media: 4s

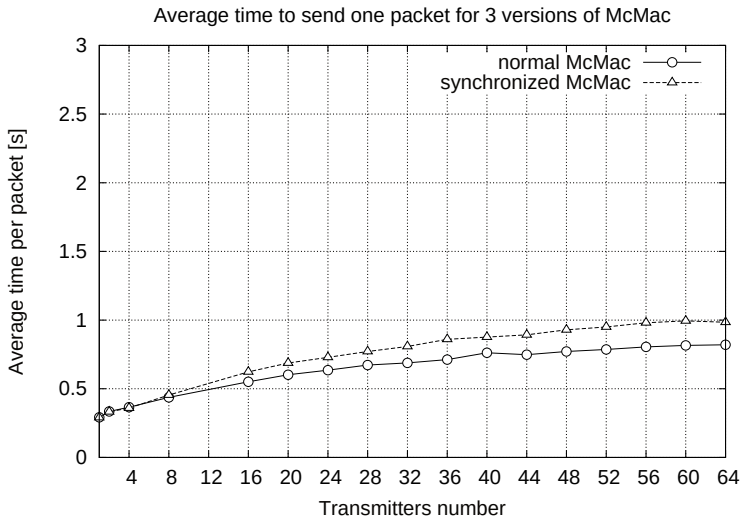


Figura 7.52: Tempo medio necessario alla trasmissione di un pacchetto nella versione originale e sincronizzata di McMac con numero di canali disponibili: 16 e media: 1s

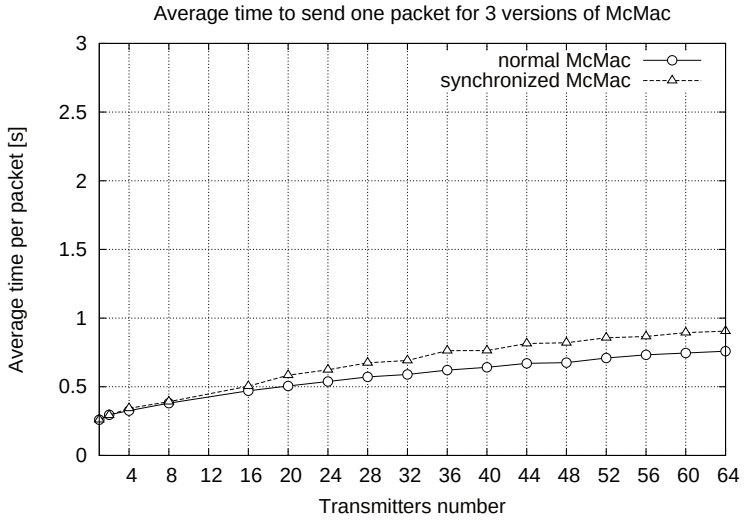


Figura 7.53: Tempo medio necessario alla trasmissione di un pacchetto nella versione originale e sincronizzata di McMac con numero di canali disponibili: 16 e media: 2s

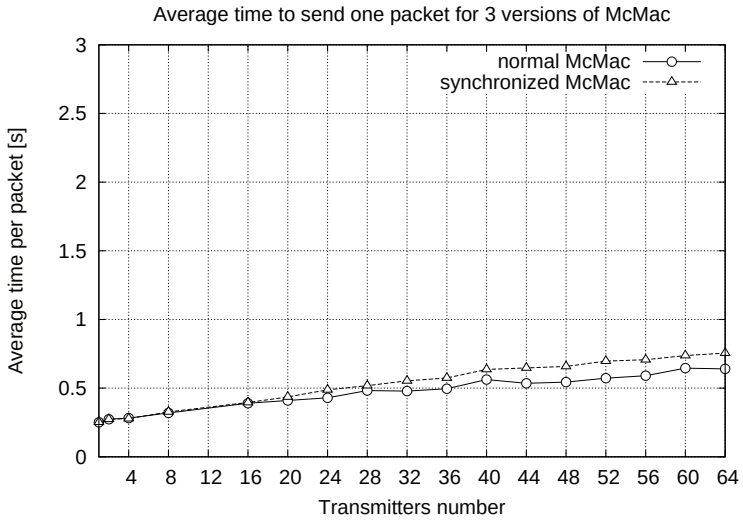


Figura 7.54: Tempo medio necessario alla trasmissione di un pacchetto nella versione originale e sincronizzata di McMac con numero di canali disponibili: 16 e media: 4s

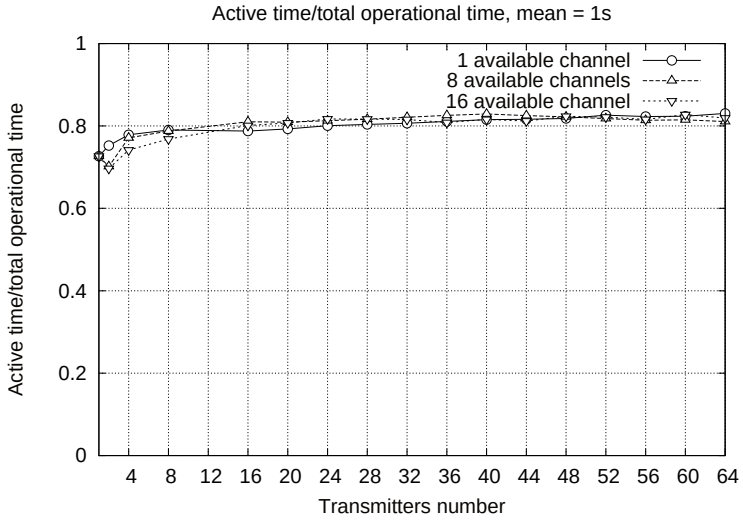


Figura 7.55: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 1s

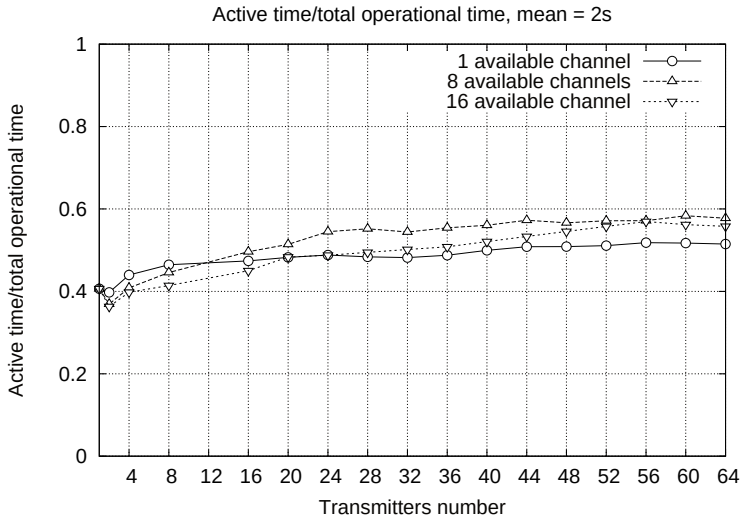


Figura 7.56: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 2s

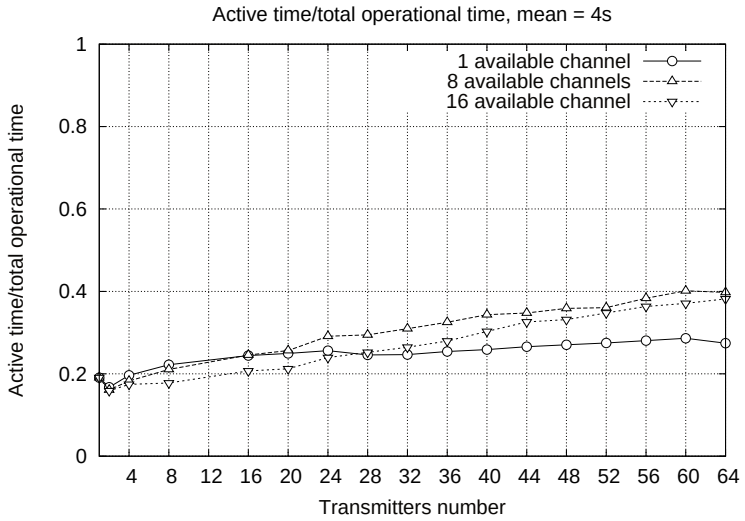


Figura 7.57: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 4s

1. L'introduzione del *duty cycle* e degli *strobed preamble* permette, nel caso in cui il tempo medio di arrivo delle richieste di trasmissione sia superiore alla durata del tempo di *sleep* prevista dal periodo di *duty cycle*, di risparmiare fino all'80% dell'energia che andrebbe consumata se i dispositivi non impegnati in trasmissione restassero sempre in uno stato di *idle listening*.

2. L'aumento del numero di canali, pur garantendo alla rete la capacità di assorbire una maggiore quantità di traffico (si vedano a tal proposito le figure da 7.1 a 7.12), comporta un maggior dispendio energetico causato dall'aumento delle trasmissioni fallite a causa di *expired transaction*, come già argomentato nelle sezioni precedenti (si vedano figure da 7.13 a 7.30).

Il confronto della versione originale del protocollo con le due varianti implementate (si vedano a tal proposito le figure da 7.58 a 7.66) evidenzia come l'introduzione della sincronizzazione consenta di ridurre consistentemente la frazione di tempo tempo trascorso in stato di attività. Questo risultato è dovuto principalmente al fatto che per ciascun nodo impegnato nell'invio di dati, il tempo che quest'ultimo trascorre negli stati attivi di trasmissione e ricezione è sensibilmente ridotto grazie al ritardo di sincronizzazione applicato a ciascuna trasmissione. Questo ritardo permette, nei casi più favorevoli, di ridurre il tempo di una trasmissione a quello necessario all'invio di un singolo preambolo e del relativo pacchetto dati, per un tammontare totale di (secondo i parametri definiti per le simulazioni) 131 byte, corrispondenti ad un tempo di trasmissione di $4.192ms$.

7.6 Percentuale di pacchetti correttamente ricevuti

In questa sezione si passa ad analizzare i risultati relativi alla quantità di pacchetti correttamente ricevuti. Questi dati permettono di saggiare la bontà di due componenti di questo protocollo:

1. Il modulo CSMA-CA implementato per McMac;

2. La resistenza all'autointerferenza dovuta alla contemporaneità di trasmissioni su canali adiacenti.

Dal momento che nello scenario di simulazione non erano previste fonti di interferenza esterna (come per esempio dispositivi IEEE 802.11 o forni a microonde), le perdite di pacchetti si possono verificare solo a causa dei due fattori sopracitati: se il modulo CSMA-CA non fosse progettato con la dovuta attenzione si dovrebbe assistere a una consistente diminuzione della percentuale di pacchetti ricevuti

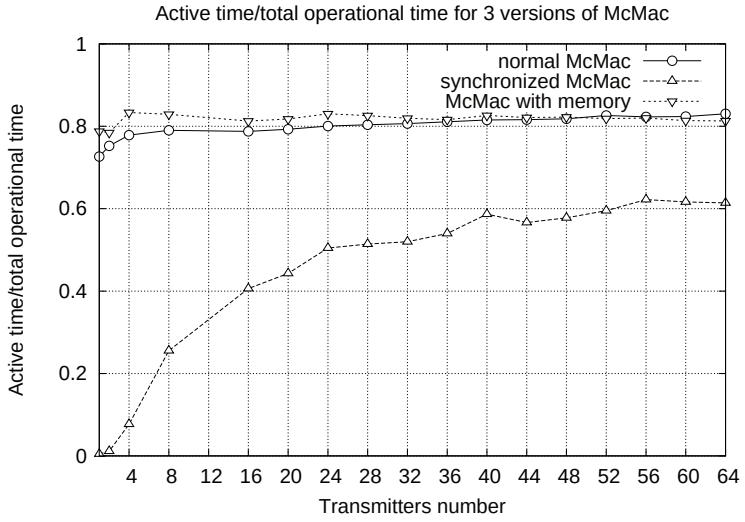


Figura 7.58: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s

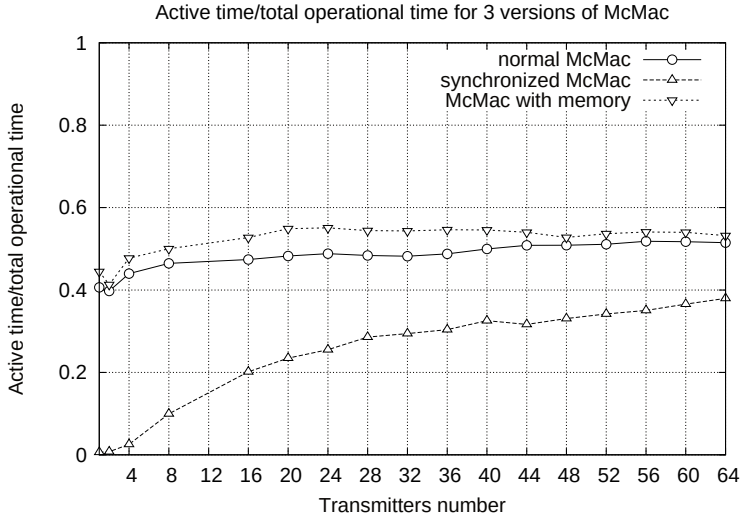


Figura 7.59: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 1 e media: 2s

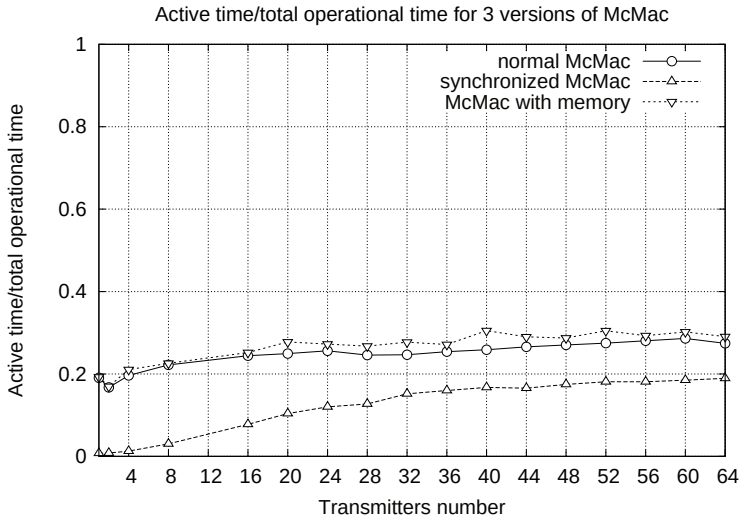


Figura 7.60: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 1 e media: 4s

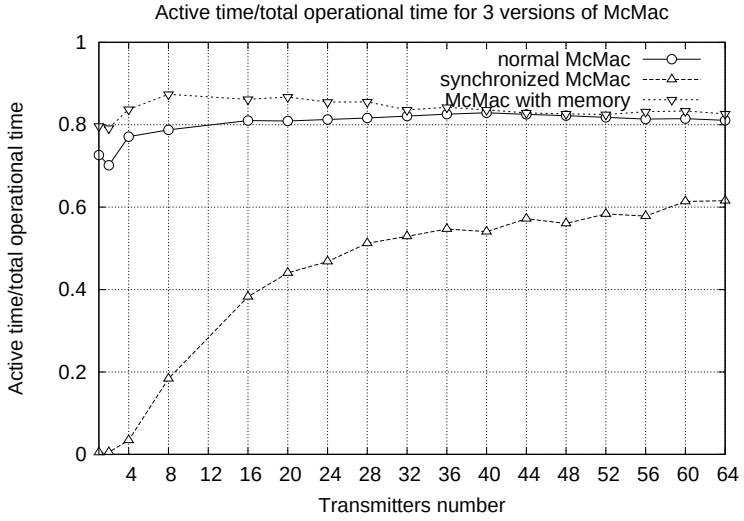


Figura 7.61: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 8 e media: 1s

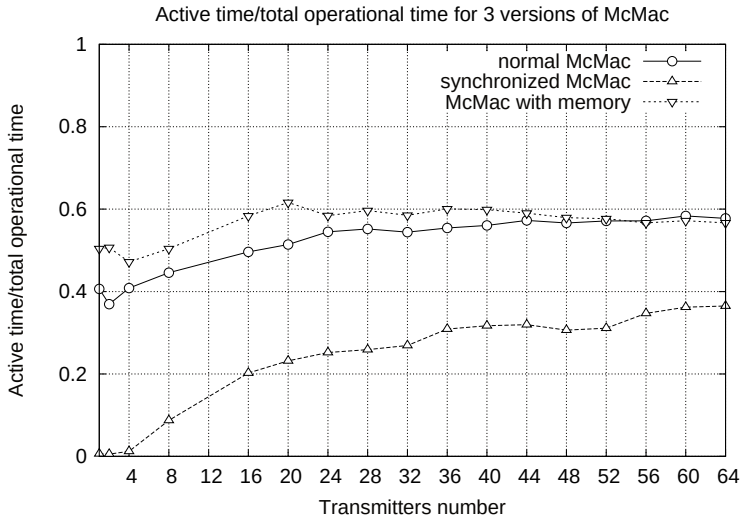


Figura 7.62: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 8 e media: 2s

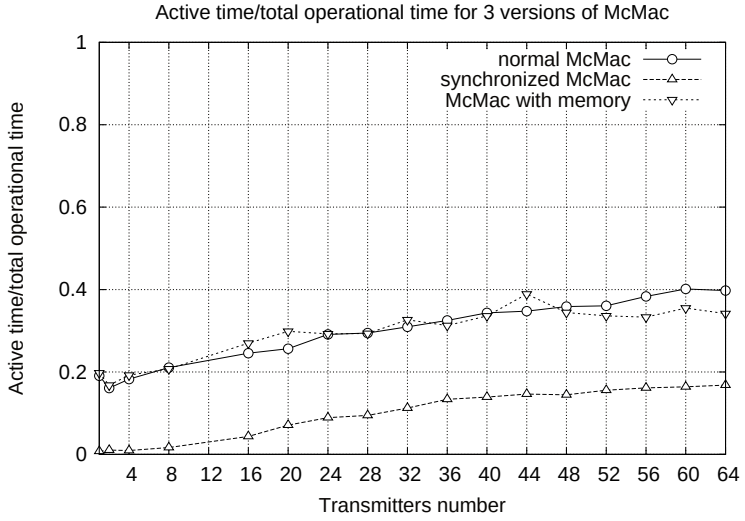


Figura 7.63: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 8 e media: 4s

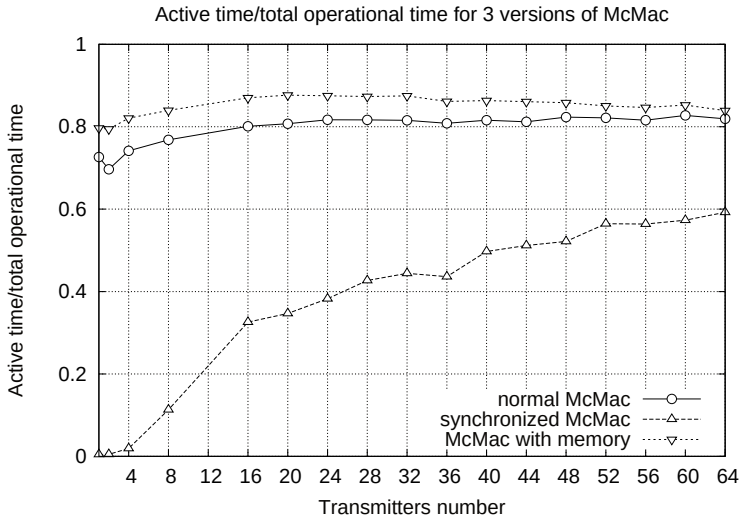


Figura 7.64: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 16 e media: 1s

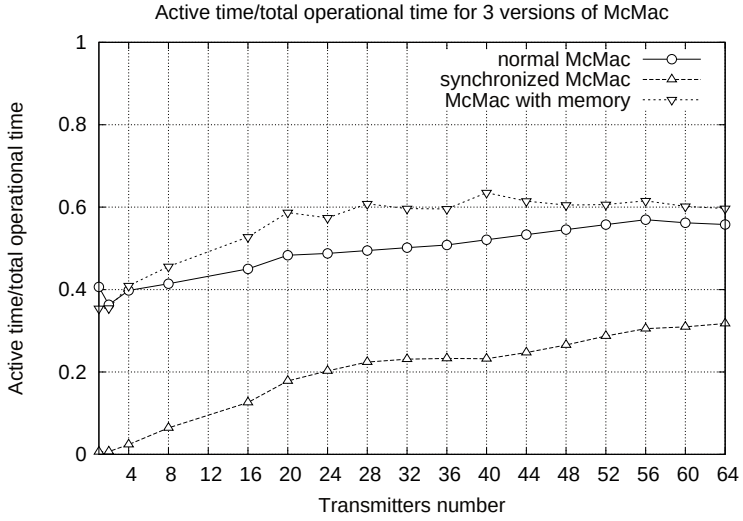


Figura 7.65: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 16 e media: 2s

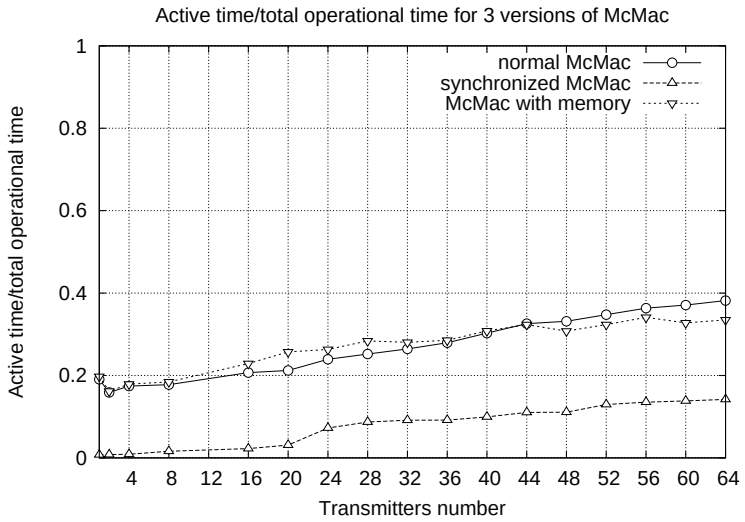


Figura 7.66: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 16 e media: 4s

rispetto a quelli inviati, specialmente nel caso in cui il numero di canali disponibili fosse esiguo; se, invece, il protocollo non fosse resistente all'autointerferenza, nei termini sopra descritti, si dovrebbe osservare un peggioramento delle prestazioni proporzionale all'aumentare del traffico presente nella rete.
Dall'analisi delle figure 7.67, 7.68, 7.69 si può evincere come nessuna delle due eventualità sopra descritte si verifichi, si può infatti osservare come la percentuale di pacchetti correttamente ricevuti si mantenga sostanzialmente costante al variare del numero di canali disponibili e del numero di dispositivi abilitati a trasmettere presenti in ciascuna simulazione.

Le figure da 7.70 a 7.78 mettono a confronto i risultati ottenuti dalle tre varianti di McMac al variare del numero di canali disponibili e del tempo medio di interarrivo delle richieste di trasmissione.
Come si può notare, mentre la versione di McMac dotata di sincronizzazione garantisce prestazioni equivalenti alla versione originale, la versione in cui è stata implementata la gestione della memorizzazione delle richieste di trasmissione descritta nel capitolo 5 risulta, in alcuni casi, consistentemente peggiore delle prime due versioni. Questo risultato è dovuto al fatto che, nel caso vi siano più pacchetti nel buffer con medesimo destinatario, se quest'ultimo riceve correttamente il primo pacchetto, ma uno dei successivi viene perso nel canale (a causa di collisioni o interferenze distruttive), tutti i pacchetti successivi a quello perduto vengono anch'essi perduti dal momento che il destinatario, non rilevando

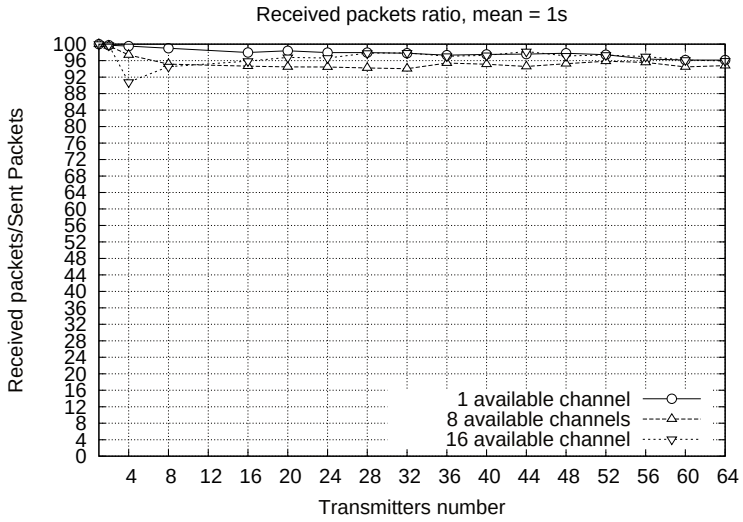


Figura 7.67: Percentuale di pacchetti correttamente ricevuti rispetto al numero di pacchetti correttamente inviati al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 1s

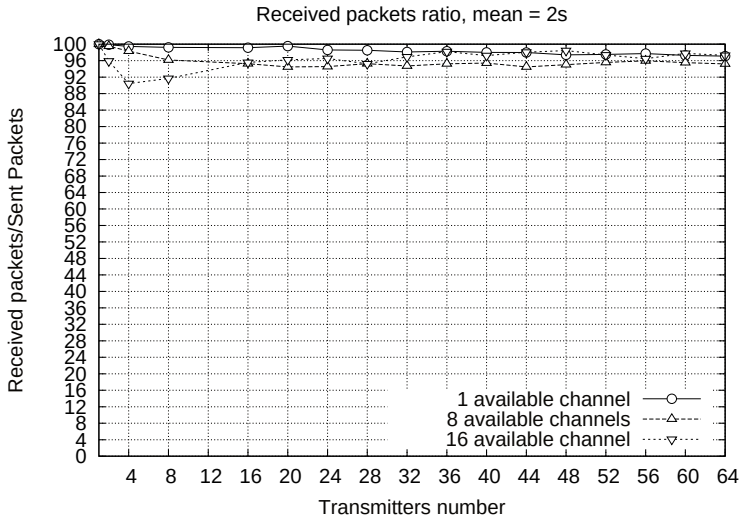


Figura 7.68: Percentuale di pacchetti correttamente ricevuti rispetto al numero di pacchetti correttamente inviati al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 2s

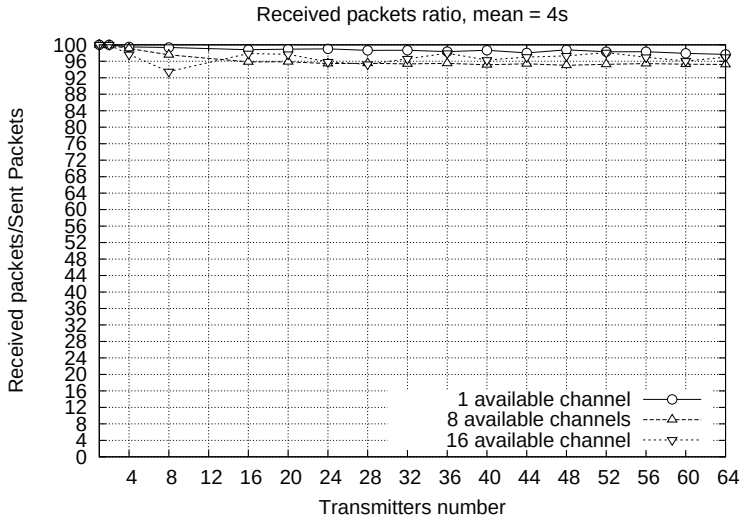


Figura 7.69: Percentuale di pacchetti correttamente ricevuti rispetto al numero di pacchetti correttamente inviati al variare del numero di canali disponibili nel caso di distribuzione esponenziale del tempo che intercorre tra una richiesta di trasmissione e la successiva con media 4s

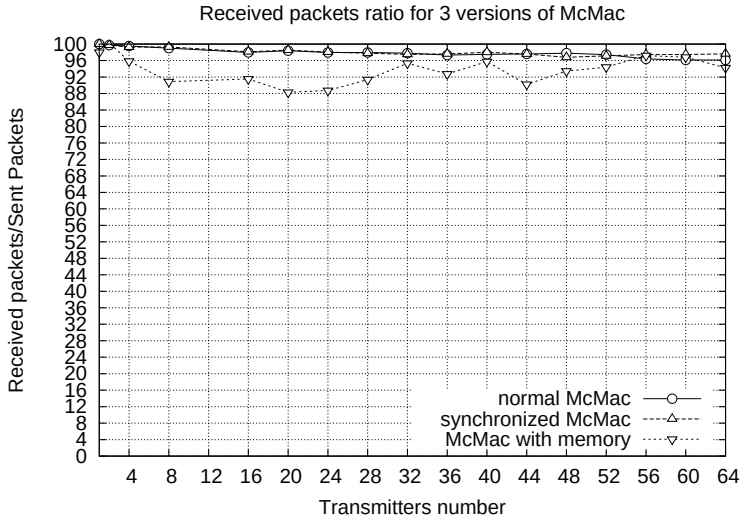


Figura 7.70: Frazione di tempo in cui il *transceiver* è attivo rispetto al tempo totale di simulazione nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s

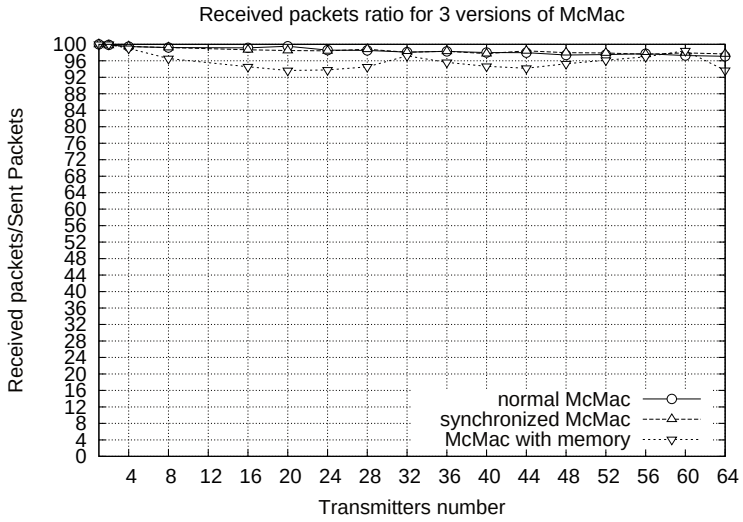


Figura 7.71: Percentuale di pacchetti correttamente ricevuti rispetto al numero di pacchetti correttamente inviati nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 1 e media: 2s

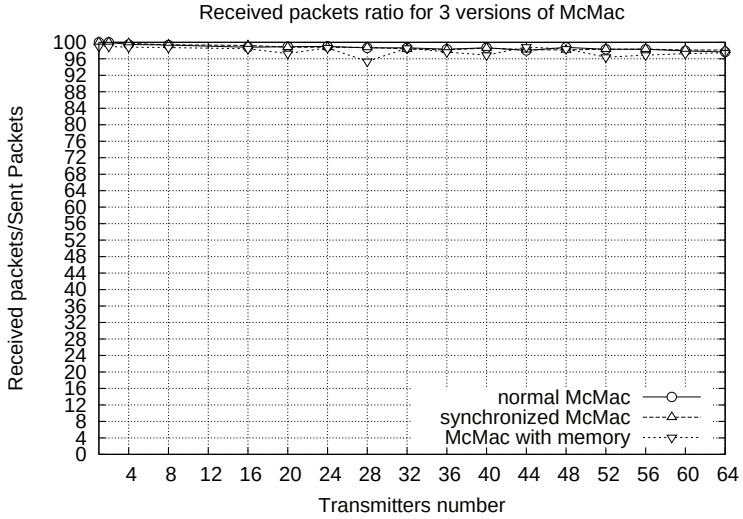


Figura 7.72: Percentuale di pacchetti correttamente ricevuti rispetto al numero di pacchetti correttamente inviati nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 1 e media: 4s

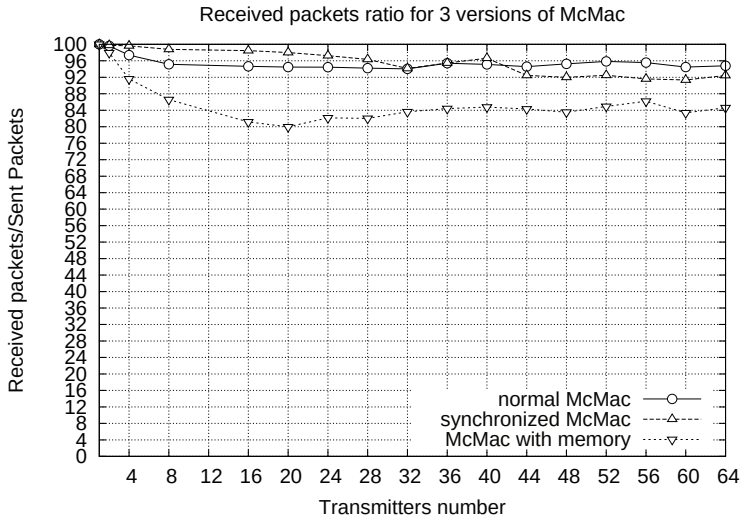


Figura 7.73: Percentuale di pacchetti correttamente ricevuti rispetto al numero di pacchetti correttamente inviati nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 8 e media: 1s

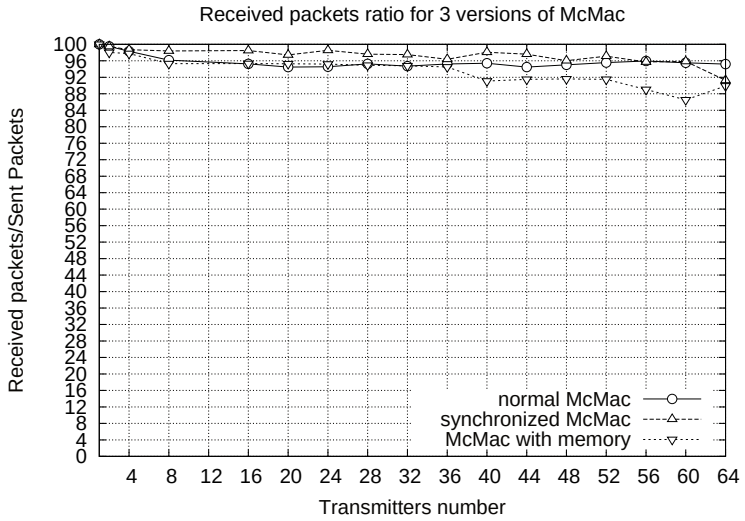


Figura 7.74: Percentuale di pacchetti correttamente ricevuti rispetto al numero di pacchetti correttamente inviati nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 8 e media: 2s

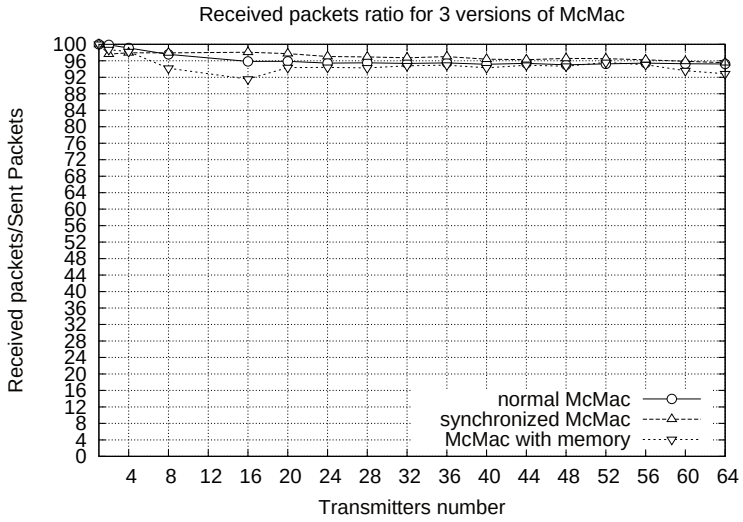


Figura 7.75: Percentuale di pacchetti correttamente ricevuti rispetto al numero di pacchetti correttamente inviati nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 8 e media: 4s

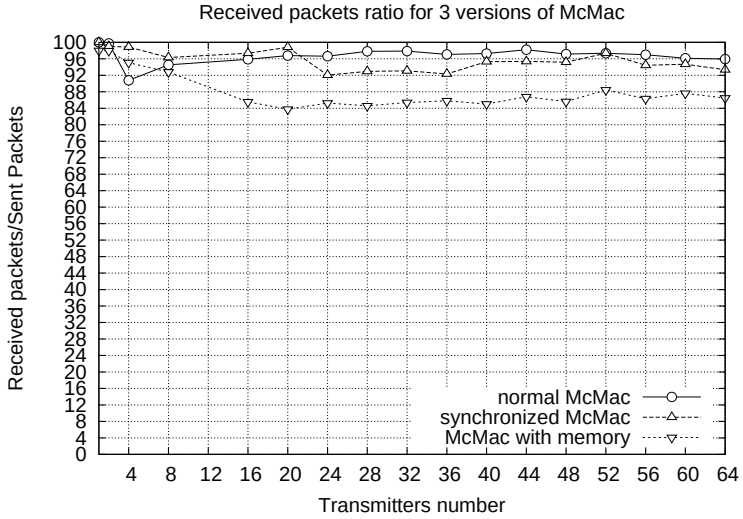


Figura 7.76: Percentuale di pacchetti correttamente ricevuti rispetto al numero di pacchetti correttamente inviati nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 16 e media: 1s

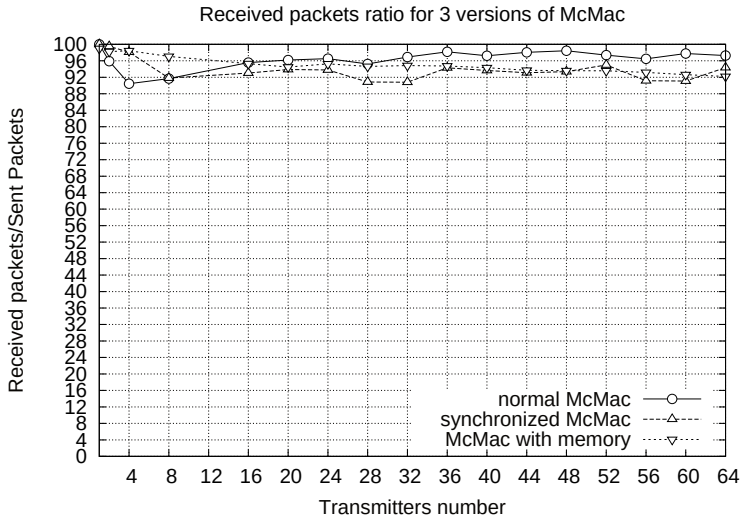


Figura 7.77: Percentuale di pacchetti correttamente ricevuti rispetto al numero di pacchetti correttamente inviati nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 16 e media: 2s

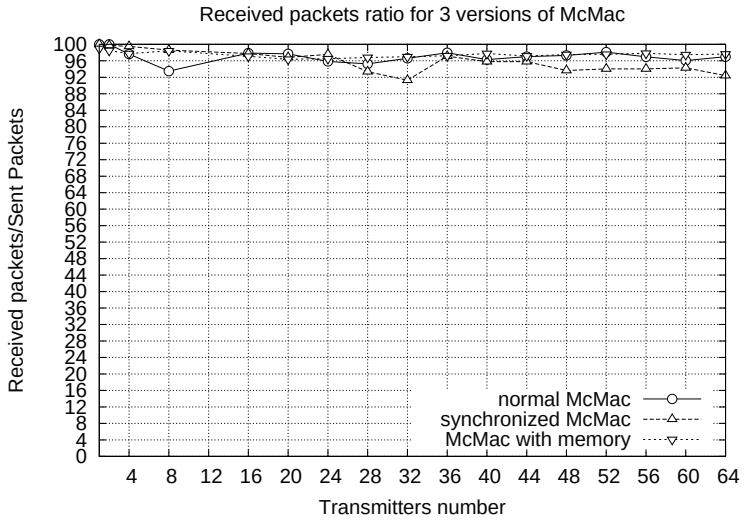


Figura 7.78: Percentuale di pacchetti correttamente ricevuti rispetto al numero di pacchetti correttamente inviati nelle tre varianti di McMac con numero di canali disponibili: 1 e media: 1s con numero di canali disponibili: 16 e media: 4s

ulteriori comunicazioni in entrata, ricomincia il processo di *duty cycling*.

Conclusioni

Il protocollo McMac, risultato del lavoro svolto per questa tesi, ha lasciato intendere, visti i risultati presentati nel capitolo precedente, come la strada della multiplazione dei canali di trasmissione, seppur poco battuta, sia promettente nel momento in cui si vogliano garantire in una rete buone prestazioni in termini di *throughput* e, nel contempo, prolungare il *time of life* atteso dei dispositivi che compongono la rete.
Le versioni di McMac ivi presentate sono il risultato del processo preliminare di progettazione e non costituiscono un prodotto finito, ma dimostrano come il percorso seguito e le idee alla base del protocollo siano nella maggior parte dei casi foriere di miglioramenti rispetto a politiche presenti in letteratura appartenenti alla classe dei protocolli basati su *preamble sampling*. Vi è dunque spazio per ulteriori miglioramenti, nonché per l'ampliamento delle funzionalità del protocollo McMac nelle direzione di un completo supporto allo standard IEEE 802.15.4 finalizzati alla presentazione di un prodotto completo utilizzabile in scenari reali e non, quindi, destinato a restare relegato all'esclusivo ambito accademico.

Alcuni miglioramenti che risulta opportuno apportare, visti i risultati precedentemente presentati sono:

1. Modifica della politica di gestione della lista di richieste di trasmissione memorizzate per una successiva elaborazione. Da questo punto di vista, le prestazioni evidenziate dall'attuale politica di gestione si sono dimostrate deludenti, a questo proposito è auspicabile una riprogettazione dell'algoritmo di gestione del buffer nell'ottica di una gestione meno aggressiva del possesso del canale. Interessante, inoltre, risulta la possibile integrazione della versione di McMac dotata della capacità, da parte del trasmettitore, di sincronizzare il momento della trasmissione dei dati al periodo di attività previsto dal *duty cycle* del destinatario, con una versione del protocollo dotata di memoria progettata tenendo conto di quanto è stato detto;

2. Un ulteriore miglioramento che sicuramente sarebbe opportuno introdurre è la possibilità per un nodo impegnato in una trasmissione di utilizzare i periodi *backoff* decretati dall'algoritmo CSMA-CA allo scopo di ascoltare

il proprio *base channel* in modo da avere la possibilità di comunicare a un eventuale mittente la volontà di rimandare la comunicazione fino al termine della trasmissione in corso;

3. Per quanto riguarda la versione di McMac dotata di sincronizzazione, oltre alla modifica descritta nel punto precedente, risulterebbe interessante, in vista di una possibile fase di tesing su dispositivi reali (i cui *clock*, progettati allo scopo di mantenere contenuti i costi di realizzazione, dimostrano consistenti imprecisioni rispetto ai *clock* dei calcolatori su cui sono state condotte le simulazioni), migliorare il processo di stima del *drift* tra il *duty cycle* del mittente e quello del destinatario in modo che lo sfasamento temporale tra i relativi periodo sia monitorato costantemente al fine di ottenere stime il più possibile accurate in ogni istante.

Per quanto riguarda l'estensione delle funzionalità, invece, oltre implementare un supporto completo alle primitive previste dallo standard IEEE 802.15.4, al fine della realizzazione di un prodotto finito, risulta indispensabile l'implementazione dei seguenti moduli

1. Implementazione di un algoritmo efficiente di *nighbour discovery* multicanale che permetta ai nodi di popolare la propria lista di vicini. Questa procedura risulta fondamentale principalmente in tre scenari:

   - Al momento del *deployment* della rete: in questo caso, infatti, nessuno dei dispositivi che andranno a formare la rete ha informazioni riguardo gli altri, deve quindi essere disponibile una procedura efficiente che consenta di creare da zero la rete in modo autonomo;
   - Nel caso in cui un nodo perda contatto con i propri vicini a causa di una qualche *failure* o nel caso in cui un nodo venga aggiunto alla rete in un momento successivo a quello di *deployment*. In entrambi questi casi, il nodo in questione dovà essere in grado di acquisire nel minor tempo possibile con un dispendio energetico comunque limitato tutte le informazioni disponibili sui nodi all'interno del proprio raggio di trasmissione;
   - Nel caso in cui sia necessario fornire supporto alla mobilità, in questo scenario, ancora più che nei precedenti, è necessario che la procedura di *neighbour discovery* sia efficiente sia in termini energetici che, soprattutto, temporali;

2. Una efficiente procedura bi *broadcasting*, possibilmente multicanale anch'essa.

3. La gestione, da parte del *network coordinator* delle richieste di associazione alla rete e deassociazione dalla stessa.

Un ulteriore miglioramento che risulta interessante progettare per il *core* di McMac è l'introduzione della capacità della rete di operare, secondo i livelli di traffico rilevati sia nella modalità basata su *preamble sampling* che, nel caso di forte congestione, in una modalità basata su *scheduling*. L'introduzione di questa capacità renderebbe il protocollo in oggetto estremamente versatile e, nel caso di buoni risultati sperimentali, applicabile alla maggior parte degli scenari di comunicazione in cui le reti di sensori si trovano ad operare.

In conclusione, si ritiene che il protocollo McMac rappresenti, per lo meno dal punto di vista dei risultati ottenuti per mezzo di simulazioni, un prodotto da tenere in considerazione per sviluppi futuri.

Bibliografia

[80206] Ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements - part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans). *IEEE Std 802.15.4-2006*, 2006.

[AZ10] A. Asterjadhi and M. Zorzi. Jenna: A jamming evasive network-coding neighbor-discovery algorithm for cognitive radio networks. In *Communications Workshops (ICC), 2010 IEEE International Conference on*, pages 1 –6, may 2010.

[Ben07] N. Benvenuto. *Communication systems: fundamentals and design methods*. John Wiley, 2007.

[BM09] Nicola Baldo and Marco Miozzo. Spectrum-aware channel and phy layer modeling for ns3. In *Proceedings of ICST NSTools 2009*, Pisa, Italy, October 2009.

[Bri88] Brian W. Kernighan; Dennis M. Ritchie. *The C Programming Language (2nd ed.)*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[BSL10] Joris Borms, Kris Steenhaut, and Bart Lemmens. Low-overhead dynamic multi-channel mac for wireless sensor networks. In Jorge Silva, Bhaskar Krishnamachari, and Fernando Boavida, editors, *Wireless Sensor Networks*, volume 5970 of *Lecture Notes in Computer Science*, pages 81–96. Springer Berlin / Heidelberg, 2010.

[BYAH06] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 307–320, New York, NY, USA, 2006. ACM.

[CD09] Kameswari Chebrolu and Ashutosh Dhekne. Esense: communication through energy sensing. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, MobiCom '09, pages 85–96, New York, NY, USA, 2009. ACM.

[Doh10] Mischa Dohler. *Embedded mac and routing protocols*. CTTC, Barcellona, 2010.

[Dur09] O. Durmaz Incel. *Multi-Channel Wireless Sensor Networks: Protocols, Design and Evaluation*. PhD thesis, University of Twente, Zutphen, March 2009.

[EFMSV08] Sinem Coleri Ergen, Carlo Fischione, Dimitri Marandin, and Alberto L. Sangiovanni-Vincentelli. Duty-cycle optimization in unslotted 802.15.4 wireless sensor networks. In *GLOBECOM*, pages 388–393. IEEE, 2008.

[EHD04] A. El-Hoiydi and J.-D. Decotignie. Wisemac: an ultra low power mac protocol for the downlink of infrastructure wireless sensor networks. In *Proceedings of the Ninth International Symposium on Computers and Communications 2004 Volume 2 (ISCC04) - Volume 02*, ISCC '04, pages 244–251, Washington, DC, USA, 2004. IEEE Computer Society.

[Gol05] Andrea Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[JD07] M.D. Jovanovic and G.L. Djordjevic. Tfmac: Multi-channel mac protocol for wireless sensor networks. In *Telecommunications in Modern Satellite, Cable and Broadcasting Services, 2007. TELSIKS 2007. 8th International Conference on*, pages 23 –26, sept. 2007.

[JJ09] Inwhee Joe and Sungmin Jung. Rp-mac: a receiver preamble mac protocol for wireless sensor networks. In *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, CCNC'09, pages 514–515, Piscataway, NJ, USA, 2009. IEEE Press.

[KHCL07] Kevin Klues, Gregory Hackmann, Octav Chipara, and Chenyang Lu. A component-based architecture for power-efficient media access control in wireless sensor networks. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[KSC08] Youngmin Kim, Hyojeong Shin, and Hojung Cha. Y-mac: An energy-efficient multi-channel mac protocol for dense wireless sensor networks. In *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pages 53 –63, april 2008.

[LC02] A. Lozano and D.C. Cox. Distributed dynamic channel assignment in tdma mobile communication systems. *Vehicular Technology, IEEE Transactions on*, 51(6):1397 – 1406, nov 2002.

[LSCK02] Pierre L'Ecuyer, Richard Simard, E. Jack Chen, and W. David Kelton. An object-oriented random-number package with many long streams and substreams. *Oper. Res.*, 50:1073–1075, November 2002.

[LY08] Hao Liu and Guoliang Yao. A variable packet length adaptive mac protocol for wireless sensor networks. In *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pages 1 –4, oct. 2008.

[LZG+10a] Jinbao Li, Desheng Zhang, Longjiang Guo, Shouling Ji, and Yingshu Li. Arm: An asynchronous receiver-initiated multichannel mac protocol with duty cycling for wsns. In *IPCCC*, pages 114–121, 2010.

[LZG+10b] Jinbao Li, Desheng Zhang, Longjiang Guo, Shouling Ji, and Yingshu Li. M-cube: A duty cycle based multi-channel mac protocol with multiple channel reservation for wsns. *Parallel and Distributed Systems, International Conference on*, 0:107–114, 2010.

[MKC+09] Neeraj Mittal, Srinivasan Krishnamurthy, R. Chandrasekaran, S. Venkatesan, and Yanyan Zeng. On neighbor discovery in cognitive radio networks. *J. Parallel Distrib. Comput.*, 69:623–637, July 2009.

[MKF07] M.F. Munir, A.A. Kherani, and F. Filali. On stability and sampling schemes for wireless sensor networks. In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks and Workshops, 2007. WiOpt 2007. 5th International Symposium on*, pages 1 –10, april 2007.

[np11a] ns 3 project. *ns-3 Manual, release ns-3.10*. ns-3 project, 2011.

[np11b] ns 3 project. *ns-3 Tutorial, Release ns-3.10*. ns-3 project, 2011.

[PHC04] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 95–107, New York, NY, USA, 2004. ACM.

[REF08] R.A. Rashid, W.M.A.E.W. Embong, and N. Fisal. Enhanced lightweight medium access protocol with adaptive multi-timeslot allocation for wireless sensor network. In *RF and Microwave Conference, 2008. RFM 2008. IEEE International*, pages 505 –510, dec. 2008.

[ROGLA03] Venkatesh Rajendran, Katia Obraczka, and J. J. Garcia-Luna-Aceves. Energy-efficient collision-free medium access control for wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 181–192, New York, NY, USA, 2003. ACM.

[SB10] C. Schlegel and E. Bouton. Multi-code wireless packet random access. In *MILITARY COMMUNICATIONS CONFERENCE, 2010 - MILCOM 2010*, pages 731 –736, 31 2010-nov. 3 2010.

[SHRD09] Paul Starzetz, Martin Heusse, Franck Rousseau, and Andrzej Duda. Hashing backoff: A collision-free wireless access method. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, pages 429–441, Berlin, Heidelberg, 2009. Springer-Verlag.

[Sou07] Juan Soulié. *C++ language tutorial*. cplusplus.com, 2007.

[Sta08] L. Stabellini. Energy optimal neighbor discovery for single-radio single-channel wireless sensor networks. In *Wireless Communication Systems. 2008. ISWCS '08. IEEE International Symposium on*, pages 583 –587, oct. 2008.

[Sta09] Luca Stabellini. *Managing interface in unlicensed Bands*. KTH Information and communication technology, 2009.

[VTGK09] Sudarshan Vasudevan, Donald Towsley, Dennis Goeckel, and Ramin Khalili. Neighbor discovery in wireless networks and the coupon collector's problem. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, MobiCom '09, pages 181–192, New York, NY, USA, 2009. ACM.

[WL09] Feng Wang and Jiangchuan Liu. Duty-cycle-aware broadcast in wireless sensor networks. In *INFOCOM 2009, IEEE*, pages 468 –476, april 2009.

[YVM09] Rajesh Yadav, Shirshu Varma, and N. Malaviya. A survey of mac protocols for wireless sensor networks. *UbiCC Journal*, 4(3):827–833, 2009. Survey.