Università degli studi di Pisa

Facoltà di Scienze Matematiche Fisiche e Naturali

Laurea Specialistica in Informatica

**Tesi di laurea**

# Analisi delle prestazioni dei protocolli tipo Aloha nei sistemi RFID

12 Dicembre 2008

*Candidato:*

**Nicola Lombardi**

*Relatore:*

**Prof. Maurizio A. Bonuccelli**

*Controrelatore:*

**Prof. Fabrizio Baiardi**

Anno Accademico 2007/08

*Alla mi' mamma.*

# Indice

# Capitolo 1

# Introduzione

Utilizzati per la prima volta durante la Seconda Guerra Mondiale per l'identificazione a distanza del traffico aereo, i sistemi di identificazione a radiofrequenza (RFID) si stanno inserendo sempre di più nella vita quotidiana delle persone cambiandone anche alcune abitudini al punto che molti studiosi ritengono che nel giro di alcuni anni questi sistemi possano portare a cambiamenti nella vita sociale dell'umanità paragonabili a quelli portati da Internet [1]

Le ridotte dimensioni e i costi contentuti dei componenti stanno, infatti, spingendo l'introduzione di questi sistemi di identificazione automatica in numerosissimi ambiti; alcune sue applicazioni, ormai ben integrate nelle abitudini della gente, sono rappresentate dai sistemi di anti-taccheggio dei supermercati o dai Telepass per il pagamento dei pedaggi autostradali. Un prossimo utilizzo degli RFID riguarda il riconoscimento di oggetti di svariata natura in sostituzione dei bar-code con conseguenti miglioramenti, ad esempio, nell'efficienza nella gestione dei magazzini o delle casse dei supermercati. Gli RFID stanno inoltre trovando applicazione, da parte di molti stati (tra

cui USA e Cina), nei documenti d'identità. con conseguente malcontento dei sostenitori della privacy che avanzano molte perplessità sul fatto che i componenti di questo sistema possano comunicare tra loro in maniera del tutto trasparente ad un utente che possegga oggetti in cui sono inseriti [2].

La convivenza di molteplici reader e tag, che rappresentano i due componenti principali di un sistema RFID, ha però portato alla luce problemi riguardanti la collisione tra le comunicazioni degli stessi. A riguardo sono stati fatte molte ricerche e avanzati alcuni protocolli atti a migliorare l'efficienza dei sistemi RFID e a risolvere il problema delle collisioni. Tra questi, l'EPCglobal, associazione no-profit di gruppi di ricerca e industriali, ha varato un protocollo basato su Aloha per risolvere la collisione tra le comunicazioni di molteplici tag.

In questa tesi analizzeremo e confronteremo le prestazioni di un generico protocollo Slotted Aloha applicato su RFID con quelle dell'EPCglobal. Per questi scopi, utilizzeremo il simulatore di reti NS-2 estendendolo in modo da supportare le caratteristiche di una rete RFID, funzionalità attualmente non presente, e dei protocolli in questione.

Nel capitolo 1 verranno presentati i sistemi RFID con una descrizione dettagliata dei suoi componenti, delle diverse tipologie esistenti e dei molteplici campi applicativi in cui ha trovato e troverà sbocco questa teconologia.

Nel secondo capitolo cominceremo ad entrare nel problema della collisione tra tag descrivendo i generali protocolli Aloha e Slotted Aloha per poi dedicarci alla descrizione dell'EPCglobal nel capitolo terzo.

Nel quarto e quinto capitolo verrà fatta un'ampia rassegna su NS-2 partendo da una descrizione generale del simulatore per finire su dettagli tecnici

e implementativi necessari per i nostri scopi.

Il sesto capitolo, quindi, descriverà e analizzerà in maniera approfondita i risultati ottenuti attraverso le simulazioni dei diversi protocolli che sarnno anche oggetto di confronti e spunti di riflessione.

Nelle conclusioni, infine, rissumeremo quanto studiato in questa tesi provando anche a suggerire eventuali futuri sviluppi e approfondimenti sulla base dei risultati ottenuti.

# Capitolo 2

# I sistemi RFID

## 2.1 Definizione di un sistema RFID

RFID (acronimo di Radio Frequency Identifaction) è una tecnologia per l'identificazione automatica di persone, animali ed oggetti di qualisiasi tipo. Il sistema si basa sulla lettura a distanza di informazioni contenute in *tag* da parte di uno o più *reader* per mezzo di comunicazioni radio.

La carattersitica principale di questa tecnologia è quella di poter scambiare e immagazzinare in maniera relativamente rapida informazioni senza che ci sia un contatto diretto, sia fisico che visivo, tra il tag e il reader. Per questo motivo la tecnologia RFID può essere vista, tra le varie applicazioni possibili, come l'evoluzione del sistema di lettura dei bar-code.

Come già accennato, un sistema RFID è composto sostanzialmente da due componenti:

- il **reader** spesso interconnesso ad un computer o ad un sistema centrale per l'immagazzinamento dei dati raccolti attraverso l'interrogazione dei

tag

- il **tag** (o transponder) inserito nell'oggetto che vogliamo indentificare

### 2.1.1 Il reader

Il reader ha la capacità di interrogare individualmente i trasponder, inviare e ricevere dati ed interfacciarsi con i sistemi informativi esistenti.

Normalmente è costituito da due parti:

- *L'unità di controllo*, ovvero un microrilevatore con un sistema operativo in tempo reale che permette di gestire:
  - Interfacce con le antenne.
  - Interrogazione dei transponder che entrano nel campo d'azione di un antenna.
  - Gestione delle collisioni tra i messaggi di risposta dei transponder.
  - Interfaccia con i sistemi informativi aziendali.
- *Le antenne*, che sono le reali interfacce fisiche tra l'unità di controllo e i transponder. Questi, infatti, per poter comunicare con il reader devono entrare nel campo magnetico generato dall'antenna ad esso attaccato.

Dal punto di vista elettronico, il reader si può paragonare ad un sistema di trasmissione radio dotato di fonte di alimentazione propria.

L'antenna del reader è un elemento molto importante per ottenere le migliori prestazioni di velocità ed affidabilità di lettura; deve essere progettata e realizzata in funzione delle caratteristiche di distanza di lettura e

Figura 2.1: *Esempi di reader*

delle dimensioni dell'antenna dei transponder con cui il reader stesso dovrà comunicare [3].

### 2.1.2 Il transponder

Il transponder (o tag) è un componente elettronico composto dalle seguenti parti:

- il *chip* ovvero la componente elettronica del tag col compito di memorizzazione dei dati e di comunicazione con il reader. Rappresenta di fatto la parte intelligente del tag.

- l'*antenna* ovvero il componente fisico che permette la comunicazione con il reader. L'antenna permette anche l'alimentazione elettronica del tag nel caso in cui questo sia passivo.

I trasponder possono essere classificati o in base al tipo di alimentazione elettronica o in base alla tipologia di memoria di cui sono equipaggiati.

Secondo la prima classificazione i tag si dividono in tag *attivi*, tag *passivi* e tag *semipassivi* [3]


Figura 2.2: *Esempi di tag*

I tag **passivi** non sono provvisti di alimentazione elettrica propria. Questi vengono attivati dall'antenna del reader non appena entrino nel raggio di azione di questo. L'assenza di campo energetico prodotto da un'antenna li spegne immediatamente. Il vantaggio dei tag passivi è un basso costo di produzione a discapito di prestazioni limitate specialmente per quanto riguarda il raggio di comunicazione (ca. 10 metri).

I tag **attivi** sono alimentati da una piccola batteria interna che, oltre a renderli indipendenti dal punto di vista dell'alimentazione elettrica, ne aumentano le potenzilità. I tag attivi,oltre ad avere un raggio d'azione decisamente più ampio rispetto ai tag passivi, sono infatti in grado di poter comunicare autonomamente senza esser necessariamente interrogati dal reader.

Infine i tag **semipassivi** hanno una fonte di alimentazione propria, ma si attivano solo se interrogati da un reader. Coprono distanze di alcune decine di metri e sono più costosi da realizzare rispetto ai passivi.
Come è facile comprendere i tag passivi stanno emergendo maggiormente rispetto a quelli passivi specialmente per prodotti di tipo usa e getta in quanto i costi di realizzazione sono decisamente inferiori ed in continua diminuzione

rispetto agli altri [4]

Dal punto di vista della memoria utilizzata, invece, possiamo classificare i tag in *EAS*, *Read Only*, *OTP* e *Read/Write* [3].

I tag **EAS** (Electronic Article Surveillance) sono dotati di memoria ad unico bit che può essere ON o OFF. Tali tag sono generalmente utilizzati come antitacheggio nei negozi e nelle biblioteche.

I tag **Read Only** hanno una memoria con funzione di sola lettura che viene programmata al momento della creazione del tag e non può essere modificata successivamente. Generalmente contiene un codice unico identificativo (UId).

I tag **OTP** o One Time Programmabile utilizzano una memoria inalterabile che contiene l'UId ed una parte modificabile una sola volta dall'utente.

I tag **Read/Write** hanno una memoria con funzione di lettura e scrittura. La memoria deve essere di tipo statico e deve consentire la conservazione dei dati anche quando il transponder non si trova più nel campo d'azione del reader.

## 2.2 Applicazioni RFID

Sebbene molti di noi non abbiano mai sentito parlare diffusamente degli RFID, i Transponder non sono una tecnologia recente.

La prima loro applicazione degna di nota risale al 1940 da parte dell'aviazione britannica.

Per assistere gli operatori dei centri di controllo a terra, gli aeroplani britannici erano equipaggiati con un semplice tipo di Trasponder che rispondeva

in modo automatico alle interrogazioni dei sistemi di terra fornendo loro un codice identificativo.

Questo codice era noto come IFF o "identifcation Friend or Foe" e consentiva di identificare su uno schermo i velivoli amici. Era un dispositivo segreto dal nome in codice "pappagallo"; gli operatori di terra, per segnalare ai piloti che dovevano accendere il dispositivo, dicevano "Squakw your Parrot"; per questa ragione tutt'oggi i codici dei moderni Transponder utilizzati in aviazione vengono chiamati in gergo *Squaks*.Quel sistema di controllo è la fonte di molti termini tuttora utilizzati nel controllo del traffico aereo [1].

Molti anni sono passati dal primo utilizzo di questa tecnologia da parte dell' aviazione britannica e nel frattempo la tecnologia dei transponder e dei reader si è evoluta notevolmente oltre ad aver subito un notevole ribasso dei costi. Oggi l'utilizzo degli RFID comincia ad essere sempre più massiccio in ogni settore, portando enormi benefici e cambiamenti nei loro campi applicativi nonchè nelle abitudini delle persone al punto che molti studiosi ritengono che l'introduzione di tale tecnologia porti forti cambiamenti, paragonabili a quelli portati dall'introduzione di internet, all' umanità [1].

Tra le moltissime applicazioni, descriviamo di seguito le più importanti.

### 2.2.1 BarCode

La prima naturale applicazione dei sistemi RFID che salta subito alla mente è quella dei BarCode. Come tutti noi sappiamo, il codice a barre non è altro che un insieme di barrette nere di vario spessore presenti su un' etichetta apposta su una confezione o direttamente stampata sulla confezione stessa.

Questa sequenza di barrette rappresenta il codice identificativo del prodotto su cui è applicato. La cassiera fa passare l'etichetta di fronte ad un raggio luminoso in grado di leggere il codice e di identificare così il prodotto in questione. Per quest'ultima operazione una cassa è normalmente collegata ad un database centrale contenente tutte le informazioni associate al codice appena letto: prodotto, prezzo, categoria IVA, eventuali sconti, ecc...

L' inroduzione di un sistema RFID al posto di quello dei BarCode porterebbe notevoli vantaggi. In una pubblicità dell' IBM si vede un giovane che, passando attraverso i banconi con aria circospetta, si infila sotto il giubbotto un insieme di prodotti ed esce dal supermercato. Una guardia lo insegue, facendo ritenere allo spettatore che il ragazzo abbia commesso un furto, per poi, al contrario, consegnargli la ricevuta fiscale che il ragazzo aveva dimenticato di prelevare [1]

L'idea che la campagna pubblicitaria vuol sostenere è quella che, se tutti i prodotti fossero equipaggiati con un trasponder, sarebbe sufficiente passare attraverso un varco equipaggiato con un reader per poter calcolare il costo totale della spesa appena fatta e, magari, effettuare un pagamento in maniera automatica per mezzo di una tessera fedeltà (anch' essa ovviamente equipaggiata di un transponder).

Sebbene risulti ancora di difficile realizzazione per motivi tecnici, tra i quali i problemi di collisione nella comunicazione con i tags (argomento principale di questa tesi), la situazione appena descritta può rendere l'idea dei positivi cambiamenti e dell' efficienza che l' utilizzo degli RFID in sostituzione dei BarCode porterebbe in quanto questa tecnologia è in grado di garantire alcune funzionalità aggiuntive:

- il tag non deve essere a contatto, nè fisico nè visivo, con il reader per essere letto
- maggiore quantità d'informazione (100 byte dei BarCode contro gli 8 KByte dei transponder)
- modificabilità dell'informazione (solo per i tag Read/Write)
- maggiore sicurezza dei dati trasmessi grazie alla possibilità di cifratura degli stessi
- immunità alle condizioni ambientali quali sporcizia e cattiva illuminazione

### 2.2.2 Logistica Magazzini e catena di fornitura

L'utilizzo degli RFID in sostizione dei BarCode porta anche ad un'altra immediata applicazione per quanto riguarda la logistica e l'inventario all'interno dei magazzini anche degli stessi supermercati. Equipaggiando infatti i contenitori e gli scaffali dei magazzini con dei trasponder si otterebbe una più facile lettura dei prodotti da inventariare: non sarebbe infatti necessario, ad esempio, aprire gli imballaggi per verificare la tipologia e la quantità di prodotto in quanto basterebbe un'unica semplice interrogazione da parte del reader (anche in questo caso andremmo però incontro a problemi di collisione).

Non solo. Equipaggiando i contenitori con tags del tipo read/write è possibile memorizzare informazioni sul chip. Il tag diventa quindi un sistema di identificazione che può tener traccia del prodotto in tutta la sua catena di fornitura: dalla fase di lavorazione a quella di vendita.

### 2.2.3 Logistica Trasporti

Anche il settore dei trasporti, sia pubblico che merceologico, risulta un potenziale campo applicativo degli RFID. Per quanto riguarda quello merceologico il legame è molto stretto a quello descritto nel paragrafo precedente riguardante la catena di fornitura; si pensi ad esempio all'utilizzo degli RFID all'interno di container trasportati da navi e treni. Allo stesso modo, tutto la gestione riguardante lo smistamento e il trasporto dei bagagli in un aeroporto potrebbe avvalersi delle potenzilità offerte da questo sistema.

Per quanto riguarda il trasporto pubblico, gli RFID hanno già avuto una loro prima applicazione. Un classico esempio è rappresentato dai telepass che permettono un rapido accesso in autostrade a pagamento per mezzo di un semplicissimo scambio di informazioni tra un reader e un tag disposti rispettivamente sul casello e sulla vettura.

Sempre in ambito trasporto pubblico, gli abitanti di molte città tra cui Londra, Parigi, Milano stanno utilizzando smartcard equipaggiate da transponder per l'accesso ai mezzi di superficie e metropolitane.

### 2.2.4 Identificazione della persona

Negli Stati Uniti, il Department of Homeland Security (il dicastero che si occupa della sicurezza interna) ha promosso, per i cittadini delle aeree confinanti con Canada e Messico, una tipologia di patente leggibile a distanza con l'obiettivo di risparmiare tempo e semplificare i passaggi alle dogane. Queste nuove patenti, infatti, sono equipaggiate con etichette RFID che si leggono direttamente nel portafogli, in tasca o borsa anche a decine di distanza di metri. In questo modo, quando un titolare della patente si avvicina a un

posto di frontiera, le onde radio trasmesse da un reader sono captate dal tag, inserito nella patente stessa, che invierà il proprio numero identificativo. Mentre il guidatore si avvicina all'agente di frontiera, il numero è già stato comunicato alla banca dati del Department of Homeland Security, e la fotografia del viaggiatore e altri dettagli vengono visualizzati sullo schermo in dotazione all'agente.

In paesi, tra i quali Inghilterra e gli stessi Stati Uniti, si sta pensando di equipaggiare anche passaporti e carte d'identità con microchip in grado non soltanto di riconoscere la persona, ma anche di avere maggior informazioni su questa riguardo spostamenti, caratteristiche personali di ogni genere, nonchè eventuali restrizioni come l'accesso ad uno stadio per motivi di ordine pubblico.

### 2.2.5 Problemi legati alla privacy

Il grande utilizzo degli RFID ha suscitato non poche preoccupazioni riguardanti la privacy dell'individuo specialmente nel momento in cui questa tecnologia venisse utilizzata all'interno di documenti d'identità. La maggiore perplessità giunge dalla caratteristica dei tag, inseriti in un documento d'identità o in un qualsiasi altro oggetto, di potersi identificare in maniera automatica e del tutto trasparente all'individuo che la possiede; un malintenzionato dotato di un reader potrebbe, infatti, effettuare una scansione sulla persona ottenendone diverse informazioni che vanno dalla propria identità alle proprie abitudini.

Per questo motivo molti paesi in procinto di utilizzare gli RFID nei documenti d'identità si sono adoperati per utilizzare tecniche di cifratura nei dati

inseriti nei tag in modo che soltanto gli enti autorizzati possano accedere all'informazione. Resta comunque la paura, da parte di molti sostenitory della privacy, che tali documenti d'identità possano essere comunque usati in modo illecito da governi che vogliono tenere sotto controllo i propri cittadini. Nelle carte d'identità della Cina, prendendo ad esempio uno dei paesi in procinto di utilizzare gli RFID per questi scopi, è codificata una quantità d'informazioni personali che la maggior parte delle persone potrebbe considerare scioccante compresa la storia sanitaria e il numero di figli del titolare, la sua professione, la sua religione, l'etnia ecc... [2]

Nell'unione Europea i legislatori stanno esaminando la situazione. La Commissione Europea ha riconosciuto il rischio di seri problemi di privacy con la teconologia RFID e all'inizio di quest'anno ha aperto un dibattito pubblico. In luglio si è stabilito di rendere pubbliche le raccomandazioni emerese, ma le aspettative per qualunque tipo di regolamentazione in favore della privacy del consumatore sono limitate [2].

Per quanto riguarda l'Italia, il Garante della Privacy ha emesso nel Marzo del 2005 un comunicato che sancisce i limiti e le modalità di impiego delle cosidette etichette intelligenti, riferendosi agli RFID. L'intero comunicato è consultabile in [5].

# Capitolo 3

# Tag Collision

Come descritto nel capitolo precedente un processo di comunicazione RFID consiste in uno scambio di informazioni da parte di uno o più tag verso il reader. Quest'ultimo inizia tale processo inviando in broadcast una richiesta di identificazione ai tag che, se rientreranno nel raggio di azione del reader, risponderanno inviando i propri dati.

La presenza di molteplici tag che vogliono comunicare con il reader crea però problematiche di interferenza; infatti, se due o più transponder rispondono simultaneamente i loro messaggi si sovrappongono sul canale radio creando un'informazione incomprensibile per il reader. Con **tag collision** intendiamo proprio questo evento di sovrapposizione tra i segnali di due o più tag appena descritto.

Dal momento che i tag hanno pochissime capacità funzionali e bassa energia, è impensabile ipotizzare che possano comunicare e accordarsi direttamente tra loro risolvendo così la collisione. Sarà quindi compito del reader rilevare la presenza di collisioni e stabilire eventuali regole di comunicazione basandosi su protocolli prestabiliti.

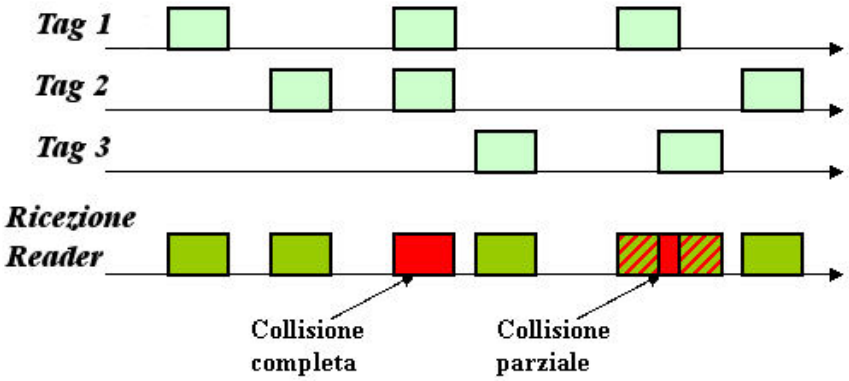


Figura 3.1: *aloha*

Come tutti i sistemi *multiple access*, il meccanismo per la limitazione delle collisioni può appartenere a due categorie distinte [3]

- protocolli probabilistici o *Aloha based*
- protocolli deterministici o *tree based*

## 3.1 Protocolli Aloha Based

Il protocollo Aloha, nella sua versione più generale, non prevede grossi vincoli riguardo l'istante in cui inviare i dati: un tag, una volta ricevuta la richiesta da parte del reader, genera un numero casuale, compreso in un certo intervallo, che rappresenta l'istante in cui inviare la risposta. Poichè ogni tag agisce indipendentemente dagli altri, il successo è determinato unicamente dalla mancata collisione con altre trasmissioni da parte di altre stazioni.

Come mostra la figura 3.1 si possono avere due tipi di collisione: totale e parziale. La prima si ha nel caso in cui due tag comincino a parlare con-

temporaneamente ottenendo, nell'ipotesi che i pacchetti inviati siano della stessa dimensione, un'offuscamento totale dei dati comunicati. La collisione parziale avviene, invece, nel momento in cui l'intervallo di tempo tra due comunicazioni di due distinti tag non è sufficientemente ampio per poter inviare un pacchetto nella sua totalità; in pratica avviene che il secondo tag incomincia a parlare nel momento in cui il primo non ha ancora finito di farlo.

Nel nostro contesto i tag non hanno capacità nè di ascoltare il canale prima di inviare i dati nè di stabilire autonomamente se i dati inviati siano giunti a destinazione in maniera corretta: sarà il reader ad inviare un'ack al tag in questione nel caso in cui la comunicazione è andata a buon fine.

I transponder, quindi, resteranno in attesa di un ack per un certo intervallo di tempo terminato il quale, non avendo ricevuto alcun messaggio dal reader, comprenderanno che il loro messaggio è stato vittima di una collisione. Nel caso in cui si dovesse verificare lo scenario descritto, i tag in questione genereranno un nuovo numero casuale che rappresenta il nuovo istante di invio. Al contrario, se un tag riceve un ack, ritenendo andata a buon fine la comunicazione, resterà in silenzio fino ad un nuovo eventuale ciclo di interrogazioni. Spesso all'ack inviato dal reader è associato un numero per identificare in maniera univoca il tag a cui l'ack stesso è indirizzato. Può infatti accadere che due tag tentino di inviare contemporaneamente un messaggio al reader ma che uno dei due messaggi subisca dei disturbi e non arrivi a destinazione. In questo scenario il reader riceverà in maniera corretta solamente uno dei due messaggi e provvederà a ringraziare il tag mittente. Il numero identificativo associato serve, appunto, per ringraziare in maniera

esclusiva il tag il cui messaggio è stato ricevuto in maniera integra da parte del reader; gli altri, non ricevendo un ack tenteranno un nuova comunicazione più avanti.

### 3.1.1 Slotted Aloha

Il protocollo Slotted Aloha aggiunge al protocollo Aloha un ulteriore caratteristica: quella che il tempo è suddiviso in intervalli discreti chiamati **slot**. L'insieme di più slot costituisce un **frame** che coincide con l'intervallo di tempo che passa tra una richiesta del reader e l'altra. Spesso indicheremo con **framesize** la dimensione, espressa in termini di tempo o in un numero di slot, di una frame.

Ogni tag è vincolato a cominciare la propria trasmissione all'inizio di uno slot che il tag stesso avrà scelto in maniera casuale tra quelli in cui è suddiviso il frame. E' compito del reader comunicare ai tag, insieme alla richiesta dati, la dimensione del frame nonchè il numero di slot in cui è suddiviso. Spesso, come faremo noi nelle nostre simulazioni, è stabilita a priori la dimensione di un singolo slot e il reader si limita a comunicare il numero di slot che costituiscono una framesize.

Una naturale conseguenza dello Slotted Aloha è quella che due trasmissioni o collidono completamente all'interno dello stesso slot oppure non collidono affatto; il problema delle collisioni parziali è dunque eliminato.

La gestione delle collisioni è praticamente analoga a quella dell'ALOHA tradizionale: il reader invierà un ack per ogni messaggio ricevuto correttamente e resterà invece in silenzio in caso di collisione. I tag che non ricevono il responso del reader realizzano, quindi, che il loro messaggio è andato in

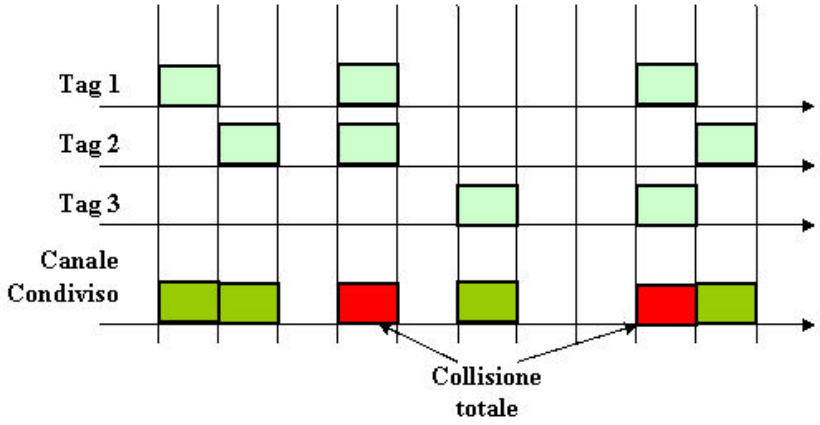


Figura 3.2: *Slotted Aloha*

collisione e si prepareranno, dunque, a ritrasmettere nel frame successivo.

Da notare che la dimensione di uno slot è tarata in modo da poter contenere il tempo necessario per inviare i dati di un tag e l'ack da parte del reader.

Al termine di ogni frame il reader conosce il numero di slot vuoti, di quelli con collisione e di quelli con communicazione corretta; ovviamente solo nel caso in cui il numero di slot con collisione sia maggiore di zero si provvederà ad un nuovo ciclo di interrogazioni per gli slot il cui messaggio non è stato compreso.

Nel caso in cui ci siano state collisioni si dovrà provvedere ad un nuovo ciclo di interrogazioni per identificare i tag che hanno colliso. In questa situazione, il reader ha due scelte possibili che danno luogo a due versioni differenti dello Slotted Aloha:

- lasciare la dimensione del frame (e dunque il numero di slot) intatta
- cambiare la dimensione del frame secondo calcoli e considerazioni prob-

abilistiche.

Entrambe le versioni dello Slotted Aloha sono state fonte di implementazione, di analisi e di confronto con l'EPC - Global.

### BFSA (Basic Frame Slotted Aloha)

L'algoritmo BFSA utilizza una dimensione prefissata del frame che non viene quindi mai modificata dal reader. A discapito di una maggiore semplicità dell'algoritmo abbiamo una minore ottimizzazione della framesize: se questa infatti contiene un numero relativamente piccolo di slot rispetto alla quantità di tag andremmo incontro a frequenti collisioni; in caso contrario avremmo molti slot inutilizzati e dunque un spreco.

### DFSA (Dinamic Frame Slotted Aloha)

Al contrario del BFSA, il protocollo Dynamic Frame Slotted Aloha cambia dinamicamente la propria framesize al termine di ogni ciclo di lettura in cui si siano verificate delle collisioni.

Nei nostri studi, per calcolare la nuova framesize (espressa in numero di slot), abbiamo utilizzato la **disuguaglianza di Chebyshev** secondo la quale il risultato di un'esperimento casuale che coinvolge una variabile aleatoria X, è molto vicino al valore atteso di X [4].

Siano $c_0$, $c_1$, $c_k$, rispettivamente il numero di slot vuoti, di slot in cui è avvenuta una sola comunicazione, e di slot con collisioni; siano inoltre $a_0$, $a_1$, $a_k$ i valori attesi rispettivamente per $c_0$, $c_1$, $c_k$. Allora, stando alla disuguaglianza di Chebyschev, minimizzando la differenza tra tali valori otteniamo una stima sul numero di tag che hanno trasmesso nell'ultimo frame [4].

In maniera più formale. Dati $l_i$ = lunghezza del frame, $c_0$ = num. slot vuoti, $c_1$ = num. slot con un tag, $c_k$ = num. slot con molteplici tag del frame appena analizzato, dobbiamo trovare quel valore di $n$ per cui

$$\epsilon[n] = \left| \begin{pmatrix} a_0^{l_i,n} \\ a_1^{l_i,n} \\ a_{\geq 2}^{l_i,n} \end{pmatrix} - \begin{pmatrix} c_0 \\ c_1 \\ c_k \end{pmatrix} \right| \tag{3.1}$$

è minima. Dove $c_1 + 2c_k \leq n \leq 2(c_1 + 2c_k)$ e ogni singolo valore atteso è dato dalla seguente formula:

$$a_r^{l_i,n} = \binom{n}{r} \left(\frac{1}{l_i}\right)^r \left(1 - \frac{1}{l_i}\right)^{n-r} \tag{3.2}$$

Il valore di $n$ determinato dalla (2.1), in quanto stima dei tag presenti nell'area del reader, sarà utilizzato come dimensione del framesize del nuovo ciclo di lettura.

La scelta del limite $(c_1 + 2c_k)$ inferiore in cui andare a cercare $n$ è dovuta alla semplice deduzione che questo rappresenta il minimo numero di tag presenti nel frame; per ogni slot con collisione ci sono infatti almeno due tag che hanno risposto al reader a cui devono essere sommati quelli che hanno risposto in maniera singola. Il limite superiore è stato invece scelto basandosi su precedenti lavori che hanno già affrontato questo problema [4].

### TSA (Tree Slotted Aloha)

L'idea di base del protocollo TSA [4] è quella di risovere le collisioni non appena queste si verifichino. Nei protocolli FSA, due tag che non collidono nello stesso frame potrebbero farlo nel frame successivo. In TSA questo non

può accadere in quanto se si verifica una collisione in uno slot, solamente i tag che hanno generato quella collisione in quello specifico slot saranno interrogati al successivo ciclo di lettura. Il nome di questo protocollo deriva dal fatto che questo è rappresentabile secondo una struttura ad albero in cui ogni nodo è una framesize.

I primi due passi del TSA sono analoghi a quelli degli altri algoritmi basati su Slotted Aloha. Nel primo passo il reader manda in broadcast una richiesta dati specificando la lunghezza del frame; i tag presenti nell'area di copertura del reader, quindi, selezionano in maniera casuale uno slot in cui trasmettere. La radice dell'albero con cui si può rappresentare questo algoritmo è costituita dalla frame iniziale.

Alla fine di ogni ciclo di lettura, se il reader riscontra un'avvenuta collisione in uno slot, inoltra una nuova richiesta destinata esclusivamente ai tag che hanno trasmesso nello slot in questione. Questo corrisponde ad aggiungere un nuovo nodo all'albero come figlio del precedente frame per ogni slot in cui si è verificata una collisione.

Ad ogni frame è associato quindi un valore *level* che rappresenta il livello in cui si trova in tale albero. La scelta riguardo la dimensione del nuovo frame ricade tra quelle già analizzate nei paragrafi precedenti: si può riutilizzare la solita size o ricalcolarla secondo le modalità descritte in DFSA

Ad ogni ciclo di lettura i tag memorizzano l'indice dello slot e il livello dell'albero in cui si trova il frame in cui hanno trasmesso in modo da sapere se siano coinvolti nelle future comunicazioni del reader.

Il procedimento di un nuovo ciclo di interrogazioni e di creazione di un nuovo frame descritto si ripete, chiaramente, in maniera ricorsiva fino al-

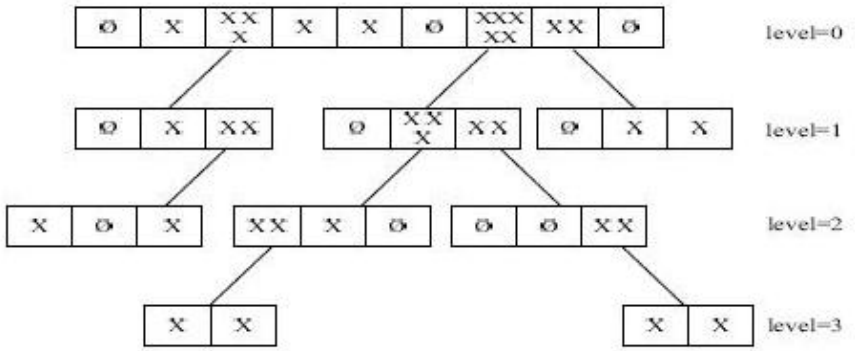


Figura 3.3: *Esempio di esecuzione del TSA*

la completa identificazione di tutti i tag nella zona ovvero fino alla totale mancanza di collisioni.

### 3.1.2 I protocolli tree based

I protocolli tree based non causano la *tag starvation*, ovvero l'impossibilità da parte di una tag pronta all'esecuzione di ottenere la risorsa di cui necessita per la trasmissione, poichè coinvolta in collisioni ripetute, però possono avere lunghi ritardi nell'identificazione.

Ci sono due famiglie di protocolli tree based:

- protocolli *binary tree based*
- protocolli *query tree based*

Nei protocolli che fanno uso dei binary tree [3] il reader chiede un bit dell'UId alla volta, cominciando dal bit che si trova nella prima posizione dell'UId della tag, avanzando nelle posizioni più basse.

Ogni volta che il reader riscontra una collisione per un bit, mette nello stato di quiete tutte i tag per i quali quel bit ha valore 1 e procede nell'identificazione delle altre interrogando circa il bit successivo. Ripete quindi il processo fino all'identificazione completa di un singolo tag, dopodichè resetta gli altri tag precedentemente messi nello stato di quiete per riattivarli e cominciare un nuovo ciclo di query.

Nel protocollo query tree based, il reader, piuttosto che inviare un solo *inquiring* bit, invia un prefisso dell'UId. I trasponder, il cui UId coincide con il prefisso, risponde al reader. Se più di un trasponder risponde, allora il reader deduce che ci sono almeno due tag con lo stesso prefisso: aggiunge 0 o 1 al prefisso e riformula la query.

Quest'ultimo è un protocollo cosidetto *memoryless*. In tali protocolli, la risposta data dal trasponder dipende esclusivamente dalla richiesta corrente del reader, senza alcuna informazione delle interrogazioni passate. Ciò comporta una grande semplificazione dei circuiti delle tag che non necessitano di memoria aggiuntiva oltre al proprio UId.

I nostri studi si focalizzeranno con maggiore attenzione sui protocolli Slotted Aloha BFSA e DFSA in quanto anche EPCglobal si basa su questa tipologia di comunicazione.

## 3.2 Reader Collision

In molte applicazioni emergenti lo spazio di interesse, ovvero l'aria di copretura coinvolta dall'applicazione, supera di gran lunga il range coperto dalla comunicazione tra un singolo reader e un transponder. E' necessario quindi ricorrere alla dislocazione di molteplici reader in comunicazione tra loro e

collegarli ad un server centrale in grado di raccogliere i dati provenienti dai reader stessi [3]

La presenza di molteplici reader in una stessa zona crea però evidenti problemi di sovrapposizione dei segnali radio dando origine al problema della reader collision. In particolare si possono avere due tipologie di collisioni dovute alla presenza di più reader:

**reader to reader collision** occorre quando un reader trasmette un segnale che interfersice con le operazioni di un altro reader impedendo a quest'ultimo di comunicare con il tag nella propria area di interrogazione.

**reader to tag collision** occorre quando un tag si trova contemporaneamente nell'area di lettura di due o più differenti reader e questi cerchino di comunicare con con il tag in questione nello stesso istante [6].

Qui ci limitiamo ad enunciare la problematica in quanto, pur appartenendo alle applicazioni e allo studio sui sistemi RFID, ha radici ben diverse da quella sulla tag-collision e una sua analisi prevede uno ricerca completamente differente da quella che affronteremo nei prossimi capitoli. Maggiori informazioni si possono comunque trovare consultando differenti articoli quali [3] [6] [7] [8]

# Capitolo 4

# EPCglobal

Nata nel 2003 dalla fusione di diversi gruppi di ricerca universitaria e industriale, l' EPCglobal si propone come un consorzio no-profit per la realizzazione di uno standard internazionale per EPC (Electronic Product Code), un sistema per l'identificazione automatica in radiofrequenza di oggetti.

L'utilizzo sempre più massiccio e le prospettive future riguardanti la diffusione dei sitemi RFID nella vita di tutti i giorni (cap. 1) hanno infatti portato molte aziende e molti ricercatori alla necessità di trovare uno standard unico internazionale a cui attenersi sia per i componenti elettronici dei sistemi RFID, sia per l'utilizzo delle radiofrequenze, sia per i protocolli di comunicazione ad alto livello. Tale standard prende proprio il nome dell'organizzazione: EPCglobal [9]

Per i nostri scopi ci soffermeremo esclusivamente sui protocolli di comunicazione tra reader e transponder affrontando, nello specifico, il problema della tag-collision e della identificazione dei tag stessi che supporremo, come fatto per i protocolli Aloha, passivi e ReadOnly. Maggiori informazioni su EPCglobal si possono comunque trovare su : [10], [9] [11] [12].

L'intero standard [10] è comunque consultabile e scaricabile dal sito dell'organizzazione [9]

## 4.1 Operazioni del reader

Lo standard EPCglobal prevede per il reader[1] tre macro-operazioni di base per la selezione, l'identificazione e la gestione dei tag:

**SELECT:** il processo attraverso il quale un reader seleziona una popolazione di tag per le operazioni di identificazione e gestione degli stessi. Un reader può utilizzare più volte il comando *select* prima di passare alla fase di *inventory*[epc]

**INVENTORY:** è il processo di identificazione dei tag vero e proprio. Comincia con un comando *Query* e l'intervallo di tempo dell'intera *inventory* è suddiviso in slot in maniera simile ai protocolli Aloha. In ogni singolo slot può rispondere nessuno, uno o più tag. E'a questo livello che verrà dunque gestita la tag collision.

**ACCESS:** Il processo attraverso il quale un reader interagisce (in lettura o in scrittura) con ogni singolo tag che deve essere identificato in maniera univoca prima di potervi accedere.

Per i nostri scopi ci focalizzeremo esclusivamente sul processo di *inventory* e sui relativi comandi che verranno implementati e analizzati per mezzo del simulatore NS-2.

[1] che nelle specifiche viene chiamato *inventory*

### 4.1.1 Comandi dell' *inventory*

I comandi utilizzati e inviati ai tag durante la fase dell' *inventory* sono cinque:

**Query:** inizia un' inventory round specificando un valore intero Q che verrà utilizzato dai tag per generare un numero casuale indicante l'istante in cui poter cominciare a parlare. I tag memorizzeranno il numero generato in un proprio *slot counter*

**QueryRep:** indica ai tag di decrementare il proprio *slot counter* e di verificare che questo, una volta decrementato, sia uguale a zero; in tal caso i tag risponderanno inviando il proprio RN16 (vedere paragrafo 3.2).

**QueryAdjust:** simile alla QueryRep con la differenza che con questo si invitano i tag a rigenerare un nuovo valore per lo *slot counter*. Con questo comando verrà dunque passato un intero Q.

**ACK:** utilizzato per ringraziare, come segnale di avvenuto riconoscimento, un singolo tag. Tale comando ha al suo interno un valore RN16 identificativo del tag a cui è rivolto l'acknowldgement.

**NAK:** annulla il processo di riconoscimento. Può essere inviato a uno o più tag in seguito a diversi situazioni verificatesi durante la fase di riconoscimento come intereferenze o altro.

## 4.2 I tag

In base ai comandi ricevuti dal reader e al processo di gestione in corso, ogni singolo tag può trovarsi e transitare in uno tra sette possibili stati raggruppabili in tre gruppi in base al processo in corso.

**Ready:** stato iniziale in cui si trova un tag che rientra nell'area di lettura di un reader, ricevendone l'energia necessaria per essere attivato (solo nel caso di tag passivi), ma che non è ancora rientrato nel processo di identificazione

**Arbitrate, Reply, Ackonwledge:** sono gli stati in cui si trova un tag durante il processo di inventory. Nelle pagine successive verrà data una più dettagliata spiegazione a riguardo in quanto parte inerente i nostri studi.

**Open, Secured** utilizzati durante la fase di access in cui il reader interagisce con i tag

**Killed:** stato in cui si trova un tag in seguito ad un comando *kill* da parte del reader. In questo stato un tag viene definitivamente escluso da qualsiasi tipo di operazione del reader. L'operazione *kill* e il conseguente ingresso nello stato *killed* sono operazioni irreversibili.

Inoltre ogni tag possiede al proprio interno alcuni dati quali:

**Slot Counter:** utilizzato per memorizzare l'istante in cui poter inviare il proprio RN16 al reader

**RN16:** primo codice di riconoscimento per avviare, con una sorta di handshake, la comunizione con il reader

**PC, EPC, CRC-16:** dati identificativi veri e propri del tag

**altri** dati dipendenti dalla tipologia dei tag e dalle esigenze.

### 4.2.1 stato *Arbitrate*

Lo stato *Arbitrate* può esser visto come uno stato in cui un tag si trova in attesa che il proprio slot counter raggiunga il valore di zero. Un tag che si trova in questo stato decrementa il valore del proprio slot counter per ogni *QueryRep* ricevuta dal reader o la cambia, generando un nuovo numero casuale, nel momento in cui riceve una *QueryAdjust*. Non appena il valore dello slot counter raggiunge il valore zero, il tag in questione passa nello stato *Reply* e invia al reader un messaggio contenente il proprio RN16.

### 4.2.2 stato *Reply*

Un tag si trova in questo stato non appena il proprio slot counter ha raggiunto il valore di zero e, quindi, nel momento in cui comincia a parlare per identificarsi con il reader. Per farlo invia a quest'ultimo un messaggio contenente il proprio RN16 rimanendo in attesa di ACK; la ricezione di un valido ACK (contenente il proprio RN16) farà passare il tag nello stato *acknowledged*. Al contrario, se il tag in questione non riceverà l'ACK entro un tempo prestabilito, o riceverà un ACK con un RN16 differente dal proprio, o addirittura un NAK, tornerà nello stato arbritrate.

### 4.2.3 stato *Acknowledged*

Un tag si trova in questo stato non appena è stato identificato dal reader. In questo stato il tag resta in attesa di comandi specifici (facenti parte della sessione *Access*) che lo faranno passare in uno degli stati *Open* o *Secured*. Qualora non ricevesse alcuna istruzione entro un tempo prefissato il tag tornerà nello stato *arbitrate*.

## 4.3 Il processo di *Inventory*

Il processo di identificazione dei tag comincia con un comando *Query*, contenente un parametro Q, da parte del reader. I tag nello stato ready, non appena ricevono questo comando, generano un numero casuale compreso tra 0 e $x = 2^Q - 1$ inclusi e lo caricano nel proprio slot counter. I tag che hanno generato un valore $x = 0$ transitano immediatamente nello stato *reply* e rispondono immediatamente al reader, gli altri che hanno generato un valore $x \neq 0$ passano invece nello stato *Arbitrate* in attesa di un comando *QueryRep* o *QueryAdjust*.

Assumendo che nella situazione appena descritta soltanto un tag si trovi con $x = 0$ l'algoritmo di riconoscimento, nel suo andamento standard e senza inconvenienti di alcun tipo, procede nella maniera seguente [9]:

1. il tag risponde inviando un RN16 ed entra nello stato *reply*
2. il reader ringrazia il tag replicando con un ACK contenente lo stesso RN16 ricevuto
3. il tag, ricevuto l'ACK con il corretto RN16, passa nello stato *Acknowledge* inviando al reader i propri PC, EPC e CRC-16
4. il reader invia una *QueryRep* o una *QueryAdjust* causando la terminazione della fase di identificazione con il tag in questione. Ovviamente l'invio di uno di questi comandi porta i tag nello stato Reply a modificare il proprio Slot Counter con la conseguente trasmissione di RN16 da parte di quei tag il cui Slot Counter avrà raggiunto il valore di zero.

Può capitare che un tag, una volta inviato il proprio RN16, non riceva, causa perdita o interferenza sul canale radio, l'ACK da parte del reader

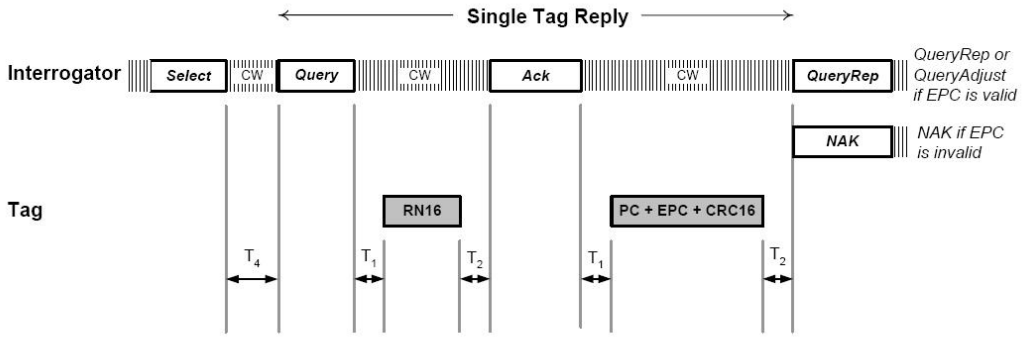


Figura 4.1: *Processo identificazione EPCglobal nel suo regolare andamento*

entro il tempo prestabilito $T_2$. Allo stesso modo può capitare che un tag riceva un RN16 differente da quello inviato: ciò è dovuto al fatto che, oltre alla perdita del proprio messaggio, un altro tag ha tentato di comunicare nello stesso istante riuscendo nella propria identificazione. Evidentemente l'RN16 ricevuto in broadcast dal reader è quello dell'altro tag. In entrambi i casi il tag in questione tornerà nello stato *arbitrate* in attesa di una nuova possibilità di identificazione.

Dal punto di vista del reader,come noto, si possono verificare, oltre al normale svolgimento dell'algoritmo, due situazioni: collisione tra risposte dei tag e assenza di risposte.

Nel caso in cui il reader rilevi una collisione invierà immediatamente una QueryRep o QueryAdjust interrompendo, di fatto, l'identificazione dei tag che hanno causato la collisione. Questi, ricevuto uno dei due comandi, torneranno nello stato *arbitrate* in quanto ancora non riconosciuti cambiando il valore del proprio slot counter.

Nel caso in cui il reader non riceva alcuna risposta entro un tempo presta-

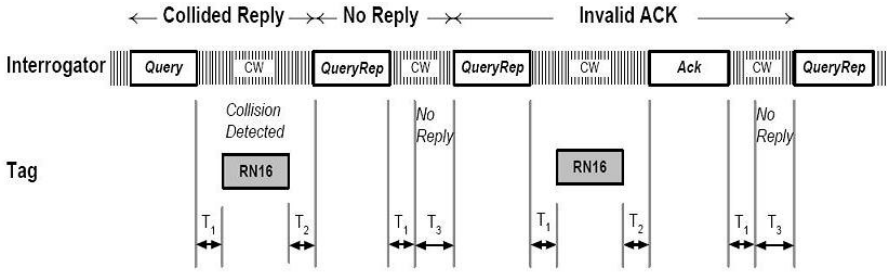


Figura 4.2: *Processo di identificazione dell'EPC global nel caso di collisioni, no reply, invalid ack*

bilito $T_3$, realizzerà, al contrario, che nessun tag sta tentando di identificarsi ed invierà, allo stesso modo, una QueryRep o una QueryAdjust.

Come descritto nelle pagine precedenti, il comando *QueryAdjust* ricalcola la dimensione del frame e invita i tag non ancora identificati a rigenerare un nuovo valore per lo Slot Counter. EPCglobal non stabilisce vincoli precisi su quando inviare questo tipo di comando al posto della QueryRep: ciò può esser fatto in maniera arbitraria in qualunque punto del frame in corso.

## 4.4 EPCglobal vs DFSA

In generale, possiamo quindi dire che l'EPCglobal è un protocollo fortemente basato sullo Slotted Aloha da cui si discosta apportando alcune modifiche. Come il più generale algoritmo, infatti, EPCglobal suddivide il frame temporale in slot per cercare di diminuire le collisioni e far parlare a turno i differenti tag. Mentre, però, nello Slotted Aloha la dimensione di ogni singolo slot è fissata fin dall'inizio dell'algoritmo e per tutta la durata del processo di identificazione, in EPCglobal questa può variare in base al verificarsi di alcune situazioni: sono, infatti, i comandi QueryRep e QueryAdjust a scandire

il confine tra uno slot e il suo successivo; i tag non fanno altro che modificare il proprio slot counter, che rappresenta proprio lo slot in cui parlare, in base alla ricezione di questi comandi ed inviare il proprio RN16 nel momento in cui lo slot conuter è uguale a zero.

Il vantaggio di questo sistema è che nel momento in cui il reader realizza che in un determinato slot non è possibile identificare un tag, per collisione o per assenza di messaggio, passa immediatamente ad analizzare lo slot successivo con il comando QueryRep senza attendere la terminazione del primo.

E' comprensibile, a questo punto, anche il motivo che ha spinto i ricercatori a fare una sorta di handshake (l'invio dell' RN16 e la relativa risposta) all'inizio della comunicazione tra un tag e il reader prima che questi si scambino i dati più significativi: è proprio attraverso questa prima fase di riconoscimento che il reader può capire anticipatamente se nello specifico slot vi sarà collisione o addirittura assenza di tag e comportarsi di conseguenza.

Un'altra fondamentale differenza tra lo Slotted Aloha e l'EPCglobal è la caratteristica di poter modificare la dimensione del frame, e dunque il numero di slot, in un punto arbitrario del frame stesso per mezzo del comando QueryAdjust; nella versione dinamica dello Slotted Aloha ciò è possibile solamente al termine del frame. La QueryAdjust, infatti, invita i transponder a generare un nuovo valore per il proprio slot counter a partire da un differente valore di Q: di fatto viene interrotta l'analisi del frame in corso per cominciare con uno nuovo con dimensioni potenzialmente differenti.

Nei nostri esperimenti quest'ultima potenzialità verrà sfruttata in moltemplici combinazioni: proveremo cioè ad utilizzare il comando QueryAdjust in

diversi punti del frame in corso e a ricalcolarne la dimensione in base ai risultati fino a quel momento raccolti. Da notare che, qualora i calcoli effettuati mediante la disuguaglianza di Chebyshev in un punto intermedio del frame confermino la dimensione attuale, non verrà inviato il comando QueryAdjust ma si proseguirà regolarmente con una serie di QueryRep fino al termine del frame. Quest'ultima caratteristica è fondamentale per la terminazione dell'algoritmo che altrimenti, qualora si inviasse costantemente una QueryAdjust in un punto intermedio del frame, non avrebbe modo di analizzare tutti gli slot e quindi di verificare la totale assenza di collisioni.

# Capitolo 5

# NS-2

Scritto in C++ e in OTcl (Tcl con estenzione Object-Oriented), NS-2 è un simulatore di rete open source ad eventi discreti sviluppato dalla UC Berkley [13] e poi esteso grazie alla collaborazione di diversi gruppi di ricerca e universitari. I vantaggi di questa politica di approccio risiedono nel fatto che è gratuitamente disponibile a tutti e che esiste una vasta comunità di utilizzatori che fornisce suppporto e continui nuovi aggiornamenti di librerie e moduli. Gli svantaggi principali sono che, oltre a quella piuttosto generica fornita dagli sviluppatori [14] manca una documentazione rigorosa e organizzata e che spesso nel codice sono presenti bug non segnalati.

Grazie a NS-2 è possibile simulare svariate tipologie di reti IP, implementare protocolli come TCP e UDP, sorgenti di traffico come FTP e Telnet, meccanismi e algoritmi di routing come Dijkstra, ecc...

La figura 5.1, schematizza in maniera generale la struttura di Ns-2 che, come vediamo, è fondamentalmente un interprete per il linguaggio OTcl, con al proprio interno uno scheduler di eventi e una libreria per implementare svariate tipologie di rete. Il risultato dell'interpretazione è una sim-

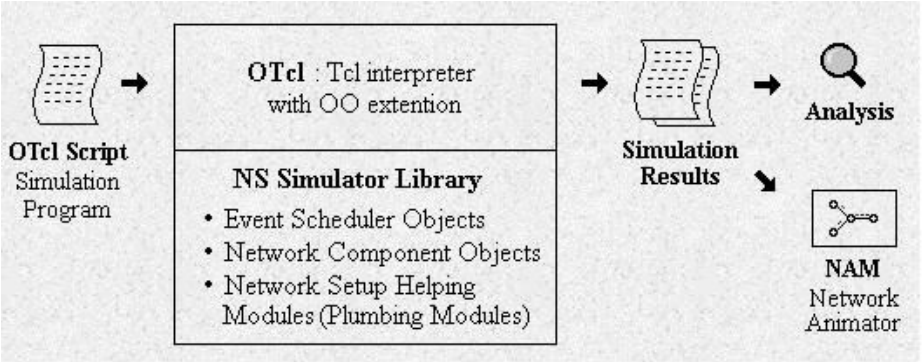


Figura 5.1: *Struttura generale di NS-2*

ulazione i cui risultati possono essere analizzati per mezzo di un file trace o dell'animatore grafico NAM.

Il simulatore adotta un linguaggio interpretato per descrivere la topologia della rete e la dinamica della simulazione. Si tratta di Otcl, una versione ad oggetti del linguaggio di scripting Tcl. Per contro il nucleo di NS è scritto in C++. Questa scelta non è casuale in quanto i progettisti hanno voluto coniugare la potenza espressiva di un linguaggio di programmazione a tutti gli effetti per la descrizione della rete con la velocità del codice generato dal compilatore C++. La complessità di gestione del software è probabilmente aumentata richiedendo l'introduzione di apposite estensioni atte al collegamento delle interfacce delle funzioni del core con le estensioni verso l'utente.

L'estensione, che fa da ponte (bind) tra le variabili e le funzioni C++, è una libreria chiamata TclCL, che crea una corrispondenza biunivoca tra la gerarchia delle classi da condividere tra C++ ed OTcl

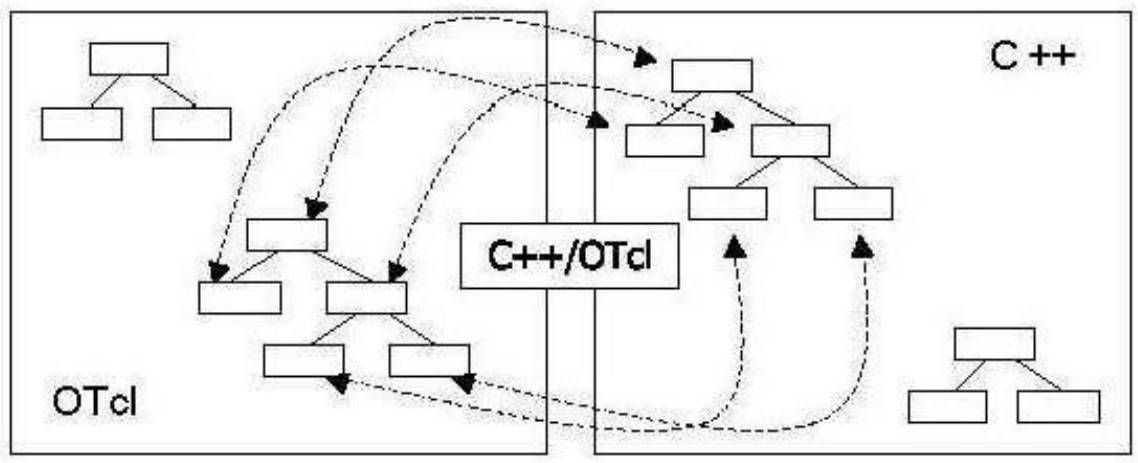


Figura 5.2: *Struttura delle classi in NS*

## 5.1 Usare o modificare NS-2

Si possono seguire due strade nell'uso di NS in dipendenza del tipo di problema che si deve affrontare.

Se la necessità è quella di avere uno strumento che consenta di relizzare delle simulazioni dei protocolli di rete esistenti, lo schema di utilizzo è quello tipico: si crea un modello della rete in esame, la si simula e si elaborano i risultati.

Questo percorso è semplice e immediato. NS offre la possibilità, per mezzo del lingaggio OTcl, di definire topologie di reti di qualunque tipo, di impostare le carattersitiche dei vari nodi, il modello di canale, il tipo di collegamenti. Alla struttura topologica si sovrappongono poi i modelli di traffico. Quelli più diffusi sono messi a disposizione dal software. I risultati delle simulazioni sono poi consultabili in due modi: o mediante un file traccia o attraverso il simulatore grafico NAM.

Nei prossimi paragrafi vedremo un esempio di simulazione di una semplice rete.

Una procedura alternativa potrebbe essere utile nel caso si vogliano sperimentare delle modifiche ai protocolli standard che si stanno studiando. In questo caso è possibile, parallelamente alla modellizzazione della rete, riscrivere i componenti del simulatore per ottenere i comportamenti della rete conformi alle varianti del protocollo in esame. Questo approccio, sicuramente più impegnativo, comporta, oltre alla conoscenza del linguaggio C++, una completa comprensione della struttura del simulatore e della sua organizzazione. Per fare un esempio, volendo creare un nodo di una rete, ciascuna delle funzioni svolte da ciscun livello ISO/OSI sarà scritta in C++, mentre il collegamento e la configurazione dei livelli avviene in Otcl. Una volta a disposizione il modello di nodo, la creazione della rete avviene tramite uno script Otcl (niente C++). Quindi volendo utilizzare delle funzioni preesistenti sarà sufficiente utilizzare il linguaggio OTcl; al contrario, se fosse necessario modficare queste funzioni elemantari o crearne di nuove è necessario utilizzare il C++ e capire come eseguire il collegamento fra quanto fatto in C++ e i comandi OTcl.

I nostri scopi prevedono, chiaramente, una modfica e una estensione del codice C++. Quello che infatti ci prefiggiamo è di creare una tipologia di rete ad hoc per simulare il comportamento dei sistemi RFID, non presenti in NS-2. Questo prevede, in particolare, la creazione di codice a livello MAC e l'implemntazione di un nuovo tipo di *Agent*, che, come avremo modo di approfondire nel prossimo capitolo, rappresentano punti terminali corrispondenti al livello transport di un nodo.

## 5.2 Lo Scheduer

Ciò che caratterizza NS è la sua struttura ad eventi discreti dove, ad esempio, un evento potrebbe essere, l'invio o la ricezione di un pacchetto da parte di un elemento della rete, oppure lo scadere di un timer. Per questo scopo, NS-2 ha al proprio interno uno schedulatore che gestisce l'insieme degli eventi in una lista ordinata ed esegue le seguenti azioni [15]:

- preleva l'evento successivo dalla lista
- scandisce l'avanzamento temporale della lista
- invoca il gestore dell'evento opportuno

Supponiamo ad esempio di avere una rete Ethernet dove il nodo A vuole spedire un pacchetto al nodo B. In un modello basato sulla riproduzione dettagliata del protocollo CSMA/CD il nodo A al tempo 0s comincerebbe il carrier sense e, in caso di canale libero, inizierebbe a trasmettere, ad esempio, al tempo 0.05s. Si supponga che al tempo 0.06s B cominci a ricevere il pacchetto di A e, al tempo 0.1s termini la ricezione e passi tutto il messaggio agli strati di rete superiori.

Un simulatore ad eventi discreti tratta questo processo definendo tre entità: il nodo A, il nodo B, e una coda che rappresenta la LAN. All'istante 0s, il nodo A inserisce nella coda della LAN il pacchetto; il nodo B, all'istante 0.1s, estrae dalla coda il pacchetto col messaggio a lui destinato. In figura 5.3 è schematizzato il principio di funzionamento dello schduler di NS-2.

In Ns-2 sono implementati due tipologie differenti di scheduler: *non-real-time* e *real-time*. I non-real-time a loro volta hanno tre differenti tipologie di

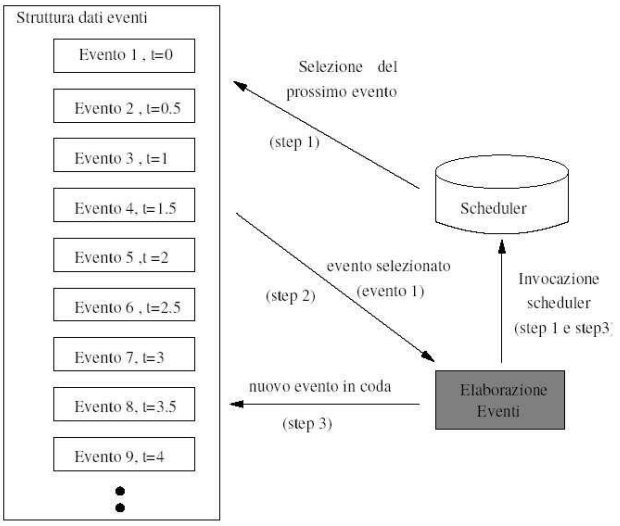


Figura 5.3: Gestione degli eventi in NS

implementazioni (*List*, *Calendar* e *Heap*) che, pur essendo differenti da un punto di vista logico, producono gli stessi risultati. Lo scheduler real-time viene utilizzato per le emulazioni nelle quali il simulatore deve interagire con reti reali [13], [16]. Lo scheduler impostato di default è il *List*, ma è possibile settarlo al momento dell'implementazione Otcl per mezzo dell'istruzione:

```
...
set ns[new Simulator]
$ns use-scheduler Heap
...
```

dove, come avremo modo di approfondire nel seguito, il primo comando crea un nuovo scheduler (**ns**), mentre il secondo ne stabilisce la tipologia (**heap**).

## 5.3 Esempi di applicazioni in Otcl

Descriveremo adesso alcuni esempi per meglio capire la sintassi del lingauggio Otcl nonchè l'utilizzo di NS-2 Tutti gli esempi riportati sono stati presi da [13]. Altri esempi sono consultabili su [17].

### 5.3.1 Esempio 1: test

Il primo esempio è un semplice script in Tcl che mostra come creare e richiamare una procedura, come assegnare un valore alle variabili e come utilizzare i cicli iterativi.

```
# Writing a procedure called "test"
proc test {} {
    addReaction
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a -$b] * $c]
    for {set k 0} {$k < 10} {incr k} {
        if {$k < 5} {
            puts "k < 5, pow = [expr pow($d, $k)]"
        } else {
            puts "k >= 5, mod = [expr $d % $k]"
        }
    }
}
# Calling the "test" procedure created above
test
```

In Tcl, la parola chiave **proc** è usata per definire una procedura, seguita da il nome della procedura e dagli argomenti tra parentesi graffe. La parola chiave **set** è usata per assegnare un valore ad una varibile. **[expr..]** è per fare in modo che l'interprete calcoli il valore dell'espressione che segue all'interno delle parentesi quadre in cui è inserito. Da notare che per ottenere il valore assegnato ad una variabile si deve utilizzare il puntatore **$**. La parola chiave **puts** viene utilizzata per l'output su prompt.

Il risultato dell'esempio appena descritto è il seguente:

```
k < 5, pow = 1.0
k < 5, pow = 1120.0
k < 5, pow = 1254400.0
k < 5, pow = 1404928000.0
k < 5, pow = 1573519360000.0
k >= 5, pow = 0
k >= 5, pow = 4
k >= 5, pow = 0
k >= 5, pow = 0
k >= 5, pow = 4
```

### 5.3.2 Esempio 2: utilizzo degli oggetti

Questo esempio è molto semplice ma mostra il modo con cui un oggetto è creato e usato in OTcl.

```
# Create a class call "mom" and
# add a member function call "greet"
Class mom
mom instproc greet {} {
    $self instvar age_
    puts "$age_ years old mom say:
    How are you doing?"
}

# Create a child class of "mom" called "kid"
# and overide the member function "greet"
Class kid -superclass mom
kid instproc greet {} {
    $self instvar age_
    puts "$age_ years old kid say:
    What's up, dude?"
}

# Create a mom and a kid object set each age
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15

# Calling member function "greet" of each object
$a greet
$b greet
```

Come mostra il codice, vengono definite due classi *mom* e *kid*, dove kid è figlio della classe mom, e una funzione chiamata *greet* per ogni classe. Dopo la definizione delle classi, viene dichiarata un'istanza per ognuna delle due

classi e viene fissata la variabile *age* a 45 per l'istanza di *mom* e a 15 per *kid*; quindi viene chiamata la funzione *greet* per ognuna delle due istanze. La parola chiave **Class** si utilizza per creare una classe , mentre **instproc** si utilizza per definire una funzione all'interno della classe. **-superclass** identifica l'eventuale classe madre. **$self** ha la stessa funzione della parola chiave this in C++, mentre **instvar** controlla se la variabile a cui è già associata sia o meno già stata dichiarata: nel primo caso viene referenziata altrimenti creata. Infine, la parola **new** crea, come facile comprendere, un nuovo oggetto della classe a cui è associato.

Il risultato dell'esempio appena descritto è il seguente

```
45 years old mom say:
   How are you doing?
15 years old kid say:
   What's up, dude?
```

### 5.3.3 Esempio di una semplice simulazione

Vediamo adesso di come realizzare un semplice scenario di rete con Otcl.

Questa rete consiste di 4 nodi (no, n1, n2, n3) come mostra la figura 5.4. Il doppio collegamento tra n0 e n2, e n1 e n2 ha un'ampiezza di banda di 2Mbps e 10ms di ritardo; quello tra n2 e n3: 1,7 Mbps e 20 ms di ritardo. Un agent tcp è stato attaccato a n0 e stabilisce una connessione ad un tcp sink attaccato ad n3. Secondo il settaggio di default di NS-2, la massima dimensione che un pacchetto tcp può generare è di 1KByte. Un tcp sink agent genera ed invia ack al mittente (tcp agent) e scarta i pacchetti ricevuti. Un agent udp è attaccato ad n1 ed è connesso ad un agent null attaccato ad n3. Un agent null si limita a ricevere il pacchetto senza far niente. Un ftp e

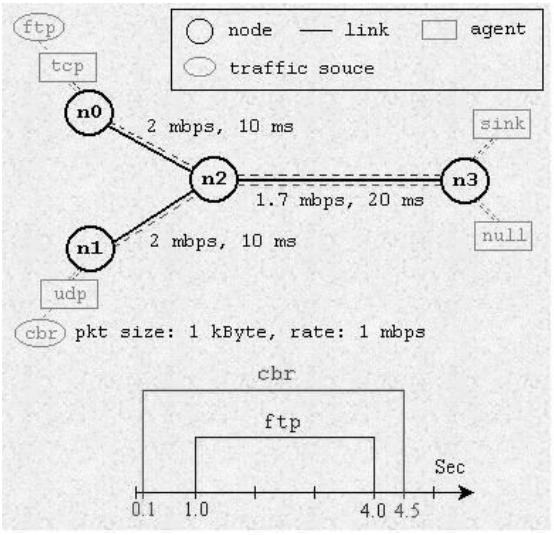


Figura 5.4: Scenario di una semplice rete realizzata con Ns2

un cbr sono attaccati rispettivamente a tcp e udp, mentre il cbr è configurato per generare pacchetti della dimensone di 1Kbyte con la frequenza di 1 Mbps. Il cbr comincia ad inviare a 0.1 sec e si ferma a 4.5 sec.; ftp parte a 1.0 e si ferma a 4.0.

Questo è il codice Otcl per implementare lo scenario appena descritto.

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}

#Create four nodes
```

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5


#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP


#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false


#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"
```

```
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

Diamo adesso una rapida spiegazione dei comandi più significativi, e che saranno utili ai nostri scopi, del codice appena mostrato.

**set *ns* [new simulator]**: genera un'istanza di NS simulator e le assegna la variabile *ns* Notare che attraverso questo comando viene anche creato lo scheduler per la simulazione che andremo ad implementare.

**set *n0* [$*ns* node**: genera un nuovo nodo di nome *n0*. Ad un nodo verrà poi associato un Agent con particolari funzioni e caratteristiche.

**$*ns* duplex-link $*n0* $*n2* 2Mb 10ms DropTail**: crea un collegamento tra i nodi *n0* e *n2* specificandone caratteristiche quali la larghezza di banda e il tipo di coda.

**set tcp [new Agent/TCP]**: crea un nuovo Agent TCP e lo associa alla variabile *tcp* . Un agent è un oggetto di base di NS-2, implementato in c++ con un apposito collegamento per il codice Otcl. Esistono diverse tipologie di Agent ognuna con caratteristiche proprie in base alla simulazione che si vuole implemntare. In questo caso viene creato un oggetto di TCP con tutte le caratteristiche che tale protocollo comporta. Come vediamo nel nostro esempio viene creato anche un Agent UDP. Un utente può anche creare un proprio nuovo Agent estendendo il codice C++ come vedremo nel successivo capitolo.

**$ns attach-agent $n0 $tcp**: *attach-agent* è la funzione che permette di associare un agent ad un nodo attribuendone quindi alcune particolari

funzionalità. Lo stesso comando è utilizzato, come possiamo vedere dalle linee successive del codice, per collegare un Agent (già connesso ad un nodo) con un oggetto di tipo *application* appena creato per mezzo della [new Application/...].

Una volta creata la topologia di tutta la rete, si passa all'implementazione vera e propria della simulazione, fornendo i comandi riguardanti gli istanti di invio dei pacchetti da parte dei singoli nodi attraverso i comandi:

```
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
```

Quindi il comando **$ns run** fa partire la simulazione.

## 5.4 I risultati della simulazione

Una volta terminata la simluazione NS-2 offre due possibilità (utilizzabili anche contemporanemente) per analizzarne i risultati: un semplice file trace.tr o per mezzo del simulatore grafico NAM. Per i nostri studi, si è preferito utilizzare il file trace sia per la maggiore quantità di informazioni che fornisce, sia per l'immediatezza dei risultati finali. Di fatto non avevamo interesse ad una simulazione grafica che, oltretutto, sarebbe risultata pesante anche per la mole di dati (in numero di nodi e di operazioni) da dover gestire.

### 5.4.1 File trace

Per imporre al simulatore di produrre un file trace è sufficiente aggiungere le seguenti linee di comando al file Otcl:

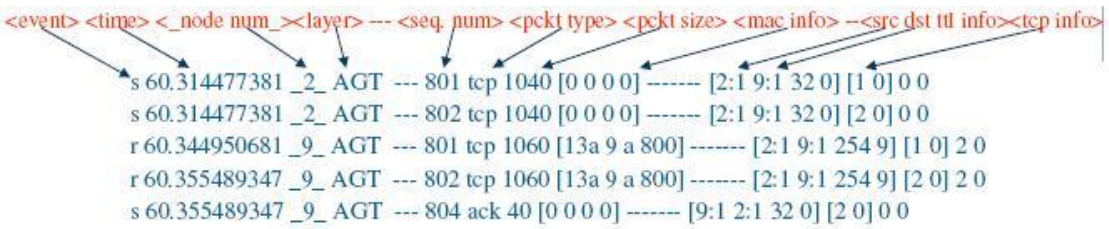


Figura 5.5: *Esempio file trace con relativa legenda*

```
set tf [open out.tr w]
$ns trace-all $tf
```

Come è facile comprendere, con queste linee viene detto al simulatore di creare un file trace *out.tr*, associarlo alla variabile *tf* e scriverci sopra tutti i passi della simulazione.

Una volta terminata la simulazione, aprendo il file *out.tr* ci troveremo di fronte ad una schermata come quella mostrata in figura 5.5 che mostra un esempio di simulazione.

Ogni operazione effettuata dal simulatore viene descritta in maniera dettagliata da ogni singola riga di testo. Andando ad analizzare i campi più interessanti, nel primo blocco notiamo: il primo dato indica la tipologia di evento che si è appena verificato (s = send, r = receive); il secondo l'istante di tempo (ovviamente della simulazione) in cui è avvenuto; il terzo e il quarto indicano rispettivamente il nodo e il relativo livello (AGT = agent, MAC = mac) in cui si è verificato.

Il secondo blocco di informazioni di ogni riga descrive caratteristiche riguardanti il pacchetto inviato quali tipologia (tcp, ack, ecc..) e dimensione [18]. Interessante come ad ogni pacchetto immesso nella rete venga associato un sequence number: questo fatto risulta molto utile in quanto,

così facendo, è possibile seguirne il percorso. Prendendo ad esempio la prima riga notiamo che il nodo 2 invia all'istante 60.314 un pacchetto a cui viene associato il seqence number 801; scendendo alla terza riga si può osservare che questo verrà ricevuto dal nodo 9 all'istante 60.344.

L'ultimo blocco di informazioni riguardano lo stato della rete ma non sono stati analizzati in quanto non interessanti per i nostri scopi futuri.

Notare che nel file trace vengono anche segnalate eventuali collisioni (indicati con una dicitura COL tra i primi due blocchi della riga in questione) tra pacchetti nel caso di simulazioni su mezzi di comunicazione condivisi.

Per quanto appena descritto, il file trace si è dimostrato uno strumento indispensabile per i nostri scopi sopratutto durante la fase realizzativa della rete RFID; è attraverso questo strumento, infatti, che potevamo verificare il corretto funzionamento delle comunicazioni nonchè la capacità da parte del reader di rilevare le evntuali collisioni.

# Capitolo 6

# Modifiche di NS-2

Coma anticipato nel capitolo precedente, i nostri scopi sono quelli di estendere il codice C++ in modo da poter supportare caratteristiche peculiari dei sistemi RFID e dei protocolli basati su Aloha. Una volta terminata questa fase sarà possibile effettuare qualunque tipo di simulazione per mezzo di piccoli script OTcl.

Per poter fare un lavoro di questo tipo è necessario uno studio più approfondito di NS-2 e della sua struttura interna.

## 6.1 Struttura di NS-2

La prima analisi da effettuare riguarda la struttura e la localizzazione dei diversi file e delle diverse directory che compongono il simulatore, in modo da capire dove andare ad effettuare le modifiche necessarie.

La directory di maggiore interesse è *../ns-allinone-2.1b/ns-2* [1]. In questa

[1] 2.1b ne indica la versione e può dunque variare

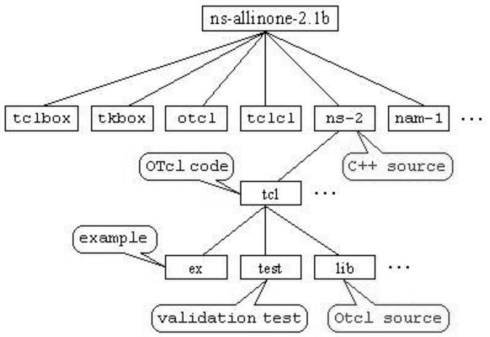


Figura 6.1: *File System di NS-2*

ci sono un po' le fonadamenta di tutto il simulatore ed è qui che troviamo le implementazioni (C++ e OTcl) di tutti i componenti di reti realizzabili con NS-2.

Ad esempio, per vedere come è stato realizzato un Agent UDP basta andare nella directory appena citata e cercare *./UDP*. All'interno troveremo i file *udp.h* e *udp.cc* con i codici richiesti.

In *../ns-allinone-2.1b/ns-2* troviamo anche l'implementazione di protocolli di livelli inferiori come quelli del livello MAC, raggruppati in apposite cartelle.

Un'altra directory di particolare interesse è *./tcl*. Al proprio interno troviamo, infatti, codici OTcl necessari, allo stesso modo di quelli in C++, per l'implementazione di elementi basilari di una rete quali nodi, agenti, packet ecc... Di questa directory analizziamo, in particolare, tre file indispensabili e di necessario utilizzo per un utente che voglia estendere NS-2.

***ns-lib.tcl*** : in questo file sono definite le classi e le funzioni utilizzate in fase di simulazione [13].

***ns-default.tcl*** : qui sono localizzabili i valori di default di alcuni parametri configurabili durante la fase implementativa. Notare che questi parametri sono definiti a livello C++ in quanto appartenenti a specifici protocolli e gestibili da Tcl per mezzo della funzione di collegamento *bind* che analizzeremo nelle pagine successive.

***ns-packet.tcl*** : come avremo modo di vedere, un elemento essenziale di NS-2 è il *packet* ovvero l'oggetto che simula uno specifico pacchetto di rete che si scambieranno i vari nodi. Nel momento in cui si crea un nuovo tipo di packet in base alle nostre esigenze, dobbiamo registrarne l'intestazione in questo file.

## 6.2 *Packet*

Un pacchetto NS, implementato in packet.h e packet.cc, è costituito da uno stack di header e da uno spazio opzionale per i dati (figura 6.2).

Per i nostri scopi risulta interessante, in particolar modo, il *cmn header* in quanto in questo vengono registrate informazioni rilevanti sul pacchetto in questione quali la tipologia (ptype) e la dimensione totale (size) usata dal programma per calcolare, ad esempio, il tempo necessario per trasportare il pacchetto su di un link con una determinata banda.

La caratteristica principale dell'*header cmn* è quella di essere in un certo senso accostata all'Agent; è quest'ultimo, infatti, che si occupa di crearlo inserendovi all'interno le informazioni di cui sopra.

Ns-2 prevede la possibilità di estendere il proprio codice creando tipologie nuove di pacchetto con maggiori informazioni interne o, ancora più precisa-

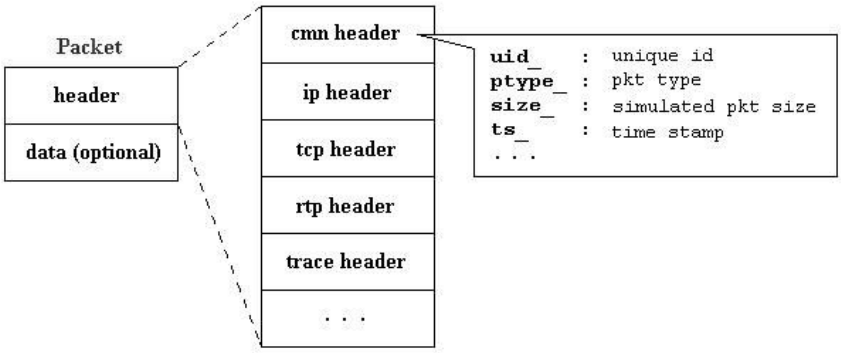


Figura 6.2: *Formato di un Packet di NS-2*

mente, in base ai dati che un Agent deve inviare. Come è facile comprendere, molto spesso, assieme all'estensione di un nuovo Agent (che analizzeremo nei prossimi due paragrafi), viene accostato un nuovo packet con un nuovo header.

## 6.3 *Agent*

Come abbiamo già avuto modo di annunciare nei precedenti paragrafi, un *Agent* è un'entità che rappresenta il livello di trasporto e che supporta meccanismi per la generazione e la ricezione di pacchetti. La classe denominata *Agent* ha un'implementazione parte in OTcl e parte in C++ [19] e può essere estesa qualora si voglia creare un Agent con proprie caratteristiche.

Per quanto riguarda la generazione di pacchetti viene utilizzata la funzione *allockpkt()* che riempe alcuni campi dell'header (quali *uid*, *ptype*, *size*, ecc...) del pacchetto che si sta creando, quindi genera un puntatore al pacchetto stesso. Notare che, qualora non esplicitamente dichiarato dall'utente,i campi dell'header veranno riempiti con i valori di default registrati in

*ns-default.tcl*.

Per quanto riguarda la ricezione, un Agent utilizza il metodo *recv()*. Questa funzione viene invocata dai nodi mittenti quando devono inviare un pacchetto ad un dato agente. La funzione recv() accetta due argomenti in ingresso, ma in molti casi gli agenti non fanno uso del secondo argomento, che viene posto al valore 0. Il primo argomento è invece il puntatore al pacchetto che deve essere ricevuto ed è quindi di importanza fondamentale. Molto spesso all'interno della recv() viene analizzato il contenuto dell'header del messaggio e, in base al contenuto, vengono richiamate differenti funzioni (quali ad esempio l'invio di un ACK al mittente).

Osserviamo due cose: in primo luogo, non tutti gli agenti attualmente implementati prevedono un proprio metodo recv(): solo gli agenti che possono essere usati come ricevitori di pacchetti prevedono questa funzione. Ovviamente, questo significa che il metodo recv() viene specializzato (tramite la tecniche della sovrapposizione) negli agenti utilizzabili come ricevitori, mentre invece non viene specializzato per gli agenti non destinati a ricevere pacchetti, i quali quindi ereditano il metodo Agent::recv() della loro classe base, anche se tale metodo non viene mai invocato. In secondo luogo, il metodo recv(), qualora previsto, risulta diverso da agente ad agente, come è ovvio che sia [19].

### 6.3.1 Collegamento con OTcl

Come ormai abbiamo visto più volte, un oggetto di uno specifico Agent viene creato e manipolato a più alto livello per mezzo dell'OTcl. Per poter fare questo occorre creare un collegametno tra il codice di script e quello in C++.

A questo scopo NS-2 mette a disposizioni due metodi da inserire nel codice C++ al momento della creazione di un nuova tipologia di Agent:

- bind()
- common()

**bind()** crea un collegamento tra una variabile OTcl ed una C++. Generalmente viene inserita all'interno del costruttore del nuovo Agent che vogliamo creare e la sua sintassi è del tipo

```
bind("numeroSlot_", &numSlot_);
```

dove il primo parametro rappresenta una variabile OTcl, la seconda quella in C++. Così facendo, ogni qual volta si vorrà far riferimento alla variabile C++ *numSlot* durante una simulazione, da OTcl sarà sufficiente richiamrla con il nome indicato nel primo parametro.

Ad esempio, nel seguente codice:

```
set reader [new Agent/RFID]
$reader set numeroSlot_ 2
```

la variabile in C++ numSlot di un reader, istanza dell'Agent RFID, verrebbe inizializzata a 2.

**common()** è una sorta di interprete di comandi scritti in OTcl. E' attraverso questo metodo che è possibile richiamare funzioni scritte in C++ durante una simulazione OTcl. Ogni estensione della classe Agent ha al proprio interno una definizione di questo metodo. Una possibile implementazione del metodo common può essere la seguente:

```
int RFID::command(int argc, const char*const* argv) {
      if(argc == 2) {
           if(strcmp(argv[1], "call-my-priv-func") == 0) {
                 MyPrivFunc();
```

```
                return(TCL_OK);
        }

                if (strcmp(argv[1], "Send") == 0) {
                        SendPacket();
                        return (TCL_OK);
                }
    }
    return(Agent::command(argc, argv));
}
```

In questo esempio *common()*, appartenente alla classe RFID (che rappresenta una possibile estensione di Agent), è in grado di interpretare due possibili comandi Otcl:

- *call-my-priv-func* che richiamerà il metodo MyPrivFunc()
- *Send* che richiamerà SendPacket()

A livello Tcl sarà quindi saranno sufficienti le seguenti righe per eseguire il metodo SendPacket su un'istanza della classe RFID.

```
set reader [new Agent/RFID]
$reader Send
```

## 6.4 Estensione di *Packet* e *Agent*

Negli ultimi due paragrafi sono stati descritti due elementi fondamentali di NS-2 quali l'Agent e il Packet accennando ad una possibile estensione delle classi in cui sono implementate. Proprio quest'ultima potenzialità ci ha permesso di creare tipologie di Agent e di Packet peculiari a quelle dei sistemi RFID e dei protocolli basati su Aloha.

Lo scopo dei nostri studi è quello di confrontare tra loro prestazioni di RFID che utilizzano il protocollo Slotted Aloha per risolvere i problemi di tag collision con RFID che utilizzano EPCGlobal. Queste due tipologie di reti sono state implementate in due fasi indipendenti e in ognuna di esse sono stati realizzati un nuovo header per la classe packet e due nuovi agent differenti per emulare i diversi comportamenti del reader e dei tag.

### 6.4.1 Descrizione implementativa

Per prima cosa dobbiamo creare un nuovo header per la classe packet contenente i campi per le informazioni che reader e transponder si dovranno inviare per la simulazione del protocollo.

```
struct hdr_MessageRFID {
        int Id;
        double send_time;
        double rcv_time;
        char ret;
        int seq;
        int inviato;
        double wait_send;
        int NumeroSlot;
        double SizeSlot;
        int destinatario;
        static int offset_;
        inline static int& offset() {return offset_;}
        inline static hdr_MessageRFID* access (const Packet* p) {
                return (hdr_MessageRFID*) p->access(offset_);
}
};
```

Una volta creati i campi del nuovo header è possibile passare all'estensione della classe *PacketHederClass*

```
static class RFIDHeaderClass : PacketHeaderClass {
public:
           RFIDHeaderClass(): PacketHeaderClass
                 ("PacketHeader/MessageRFID", sizeof(hdr_MessageRFID){
           }
           }class_MessageRFIDhdr
```

Una volta creato il nuovo header, vediamo in quale maniera è stato implementato il Reader (nel file ReaderAgent.h,.cc) di un sitema RFID estendendo la classe Agent. Nel file ReaderAgent.h sono stati semplicemente inserite tutte le variabili e i metodi che verranno poi utilizzati in ReaderAgent.cc

```
class ReaderAgent : public Agent {
public:
     ReaderAgent();
     int numSlot_;
        ...
        ...
        ...
        }
```

Quindi l'implementazione vera propria che si articola in due fasi. Nella prima viene creato il collegamento con il codice OTcl estendendo la classe TclClass

```
static class ReaderClass : public TclClass {
public:
        ReaderClass() : TclClass("Agent/MessageRFID") {}
        TclObject* create(int, const char*const*) {
                return(new ReaderAgent());
                }
}class_MessageRFID;
```

Grazie a questo collegamento sarà possibile creare un'istanza di ReaderAgent durante la simulazione in OTcl per mezzo del comando:

```
set MyReader [new Agent/MessageRFID]
```

Quindi, viene implementato il costruttore di ReaderAgent in cui facciamo anche utilizzo delle *bind* di cui abbiamo parlato sopra.

```
ReaderAgent::ReaderAgent() : Agent(PT_RFIDREADER){
           bind ("packetSize_", &size_);
           bind ("NumeroSlot_", &NumSlot_);
           bind ("FrameDinamico_", &SizeDinamica_);
           NumSlot_ = 30;
           SlotColl = 0;
           ...
           ...
```

Il metodo *command()* di ReaderAgent sarà relativamente semplice in quanto potrà ricevere il solo comando send che richiamerà il metodo SendPacket() dando il via al processo di identificazione degli RFID.

Concludiamo quindi analizzando in che modo il Reader utilizza il l'HeaderPacket appena creato, lo invia all'interno della *SendPacket()* e lo analizza in caso di *recv()*

```
void ReaderAgent::SendPacket() {


        Packet* pkt = allocpkt();

        hdr_cmn *ch = HDR_CMN(pkt);
        hdr_ip* iph = HDR_IP(pkt);
        ch->ptype()= PT_RFIDREADER;

        hdr_MessageRFID* ph = hdr_MessageRFID::access(pkt);
        ph->wait_send = 0;
        ph->NumeroSlot = numSlot_;
        ...
        send (pkt,(Handler*) 0);
```

Come possiamo vedere dal codice, una volta creato il pacchetto pkt per mezzo della *allockpkt()*, vengono creati due puntatori all'header_cmn e all'header_ip per poterne riempire alcuni campi, quali il *ptype*. Quindi viene creato l'oggetto *ph* del nuovo header appena creato (hdr_MessageRFID), aggiunto a *pkt* per poter trasmettere le informazioni in esso contenute. Dopo aver riempito i campi di ph il pacchetto può essere inviato per mezzo della *send*.

La recv() procede nella stessa maniera, ma in maniera inversa: come descritto in precedenza, una volta invocata restituisce un puntatore al pacchetto ricevuto in modo da poter leggere l'informazione inviata dal mittente.

```
void ReaderAgent::recv(Packet* pkt, Handler*) {

        hdr_ip* hdrip = hdr_ip::access(pkt);
        hdr_MessageRFID* hdr = hdr_MessageRFID::access(pkt);

        int n = hdr->seq;

        ....
```

Prima di poter procedere all'utilizzo dei nuovi componenti implementati è stato necessario effettuare ulteriore moodifiche in alcuni file interni di Ns-2. Per prima cosa dobbiamo inserire in *pacekt.h* i riferimenti ai nuovi pacchetti utilizzati

```
enum packet_t {
...
PT_RFIDREADER
..
}
...
class p_info {
public:
        p_info() {
        name_[PT_RFIDREADER] = "MessageRifid"
```

In seconda analisi dobbiamo aggiornare il file ns-default.tcl (con i valori di default di alcuni campi quali la size) e ns-packet.tcl con il nuovo pacchetto. La creazione del Tag per lo Slotted Aloha e di tutto il sistema (pacchetto, reader, tag) per la realizzazione dell'EPCGlobal procede in maniera analoga a quella che abbiamo appena visto. I passi appena illustrati sono infatti abbastanza genereci e sono grosso modo gli stessi ogni qual volta si voglia creare un nuovo Agent e un nuovo Packet.

Ovviamente ci saranno molte differenze negli algoritmi interni dei diversi Agent in base al ruolo e ai comportamenti che dovranno avere all'interno della simulazione. Prendendo ad esempio il Tag, questo avrà il metodo *common()* vuoto, in quanto gli unici comandi che può ricevere provengono dal Reader durante un ciclo di interrogazioni.

Allo stesso modo sia il Reader che il Tag di un sistema EPCGlobal avranno una struttura interna più complessa per poter emulare tutte le fasi del protocollo in questione, ma anche in questo caso le procedure di estensione degli Agent e dei Packet procedono sulla falsariga di quelli appena descritti.

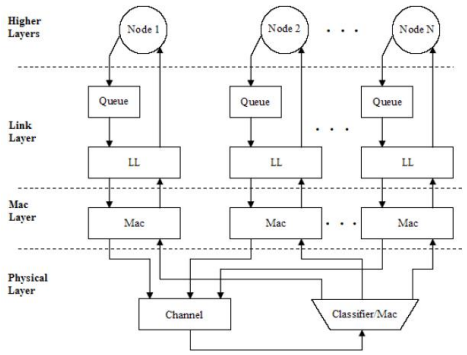


Figura 6.3: *Schema pila protocollare NS-2*

## 6.5 Rete Wireless e livello MAC

Fino ad ora, non abbiamo preso in considerazione un'aspetto fondamentale per i nostri scopi: una rete RFID è, ovviamente basata su ambiente wirelss. NS-2 offre in maniera ottimale la possibilità di simulare reti in ambiente wireless utilizzando anche protocolli di livello di rete più bassi quali il MAC.

In questo paragrafo daremo una descrizione generale dell'ambiente wireless in NS-2 descrivendo le modifiche effettuate a livello MAC per meglio simulare una rete RFID.

NS-2 organizzato secondo una struttura modulare in cui ciascun oggetto esegue le proprie elaborazioni e poi ne comunica i risultati agli oggetti adiacenti. In particolare, un nodo (W)LAN ha differenti oggetti che simulano indipendentemente i tre livelli più bassi del modello ISO-OSI: *LL, MAC e PHY*. Un apposito modello è schematizzato in figura 6.3

Un pacchetto generato dai livelli superiori viene accodato nel Link Layer, poi passato al MAC layer, che definisce le regole per accedere al canale fisico, implementato dall'oggetto Channel, che a sua volta provvede a simulare la

propagazione e a far giungere in istanti opportuni i pacchetti ai diversi nodi in ascolto sullo stesso mezzo. Il pacchetto ricevuto si trova eventualmente ad effettuare il percorso inverso e a risalire i diversi layer, giungendo fino al livello applicativo del nodo destinatario. Come avviene realmente nelle reti wireless, tutti i nodi vedono sul canale tutti i pacchetti trasmessi (se sono in visibilità radio), ma sarà compito del livello MAC di ciascun nodo accettare e trasferire ai livelli superiori solo quelli destinati al nodo stesso. La maggiore complessità in ciascun nodo risiede proprio nell'oggetto MAC, che deve provvedere a simulare oltre al funzionamento dei protocolli di accesso al canale propri del particolare livello MAC simulato, anche gli eventuali meccanismi di carrier sense e collision detection mentre quelli di propagazione sono simulati dall'oggetto Channel.

Ognuno di questi oggetti è implementato da una classe C++ che fornisce mebri pubblici per interfacciarsi con gli altri oggetti e per essere configurabili durante la simulazione OTcl. All'inizio del codice OTcl è infatti possibile andare a configurare parametri del sistema di trasmissione con cui poi configureremo i nodi della rete. Il seguente codice mostra una possibile configurazione.

```
set val(chan)      Channel/WirelessChannel    ;# Channel Type
set val(prop)      Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)     Phy/WirelessPhy            ;# network interface type
set val(mac)       Mac/802.11                 ;# MAC type
set val(ifq)       Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)        LL                         ;# link layer type
set val(ant)       Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)    50                         ;# max packet in ifq
```

Come appena descritto, un ruolo fondamentale nella gestione dei pacchetti che arrivano o partono da un nodo lo riveste il livello Mac. Per questo motivo e con l'intento di realizzare una rete il più simile possibile a quel-

la RFID, sono stati implementati livelli MAC con apposite caratteristiche. Senza addentrarci troppo sui dettagli implementativi, a livello MAC un nodo RFID deve:

- inoltrare immediatamente un pacchetto in uscita ( e dunque proveniente dal livello Agent) senza fare alcun tipo di carrier sense
- rilevare eventuali collisioni tra pacchetti entranti e, eventualmente, comunicarle all'Agent in modo che questo possa gestirle secondo il proprio protocollo (Slotted Aloha o EPCGlobal)
- scartare eventuali pacchetti non destinati a lui.

Pur essendo praticamente uguali, sono stati implementati due protocolli MAC distinti per lo SLotted ALoha e per l'EPCGlobal rispettivamente richiamabili inserendo

```
set val(mac)      Mac/Aloha
set val(mac)      Mac/EPC
```

nella parte di configurazione mostrata in alto.

# Capitolo 7

# Simulazioni e risultati

Una volta terminata l'estensione di NS-2, si può passare alla simulazione dei protocolli Slotted Aloha ed EPCglobal.

I due protocolli in questione verranno simulati in meniera separata e, per ognuno di questi, proveremo a cambiare alcuni paramteri interni in modo da poter trovare la migliore configurazione possibile. Quindi, in ultima analisi, i due protocolli verranno confrontati tra loro per cercare di stabilire quale si adatti meglio ad un sistema RFID.

## 7.1 Slotted Aloha

Come abbiamo visto nel cap. 3, un reader che utilizza lo Slotted Aloha stabilisce una framesize iniziale espressa in numero di slot e invita ogni tag a scegliere in maniera casuale lo slot in cui trasmettere. Terminato il ciclo di interrogazioni per quel frame, il reader, qualora avesse riscontrato delle collisioni, può decidere per la nuova fase di identificazione se lasciare invariato

il numero di slot del frame (BFSA) o ricalcoralo attraverso la disuguaglianza di Chebyshev (DFSA).

Nelle nostre simulazioni abbiamo provato entrambe le versioni dello Slotted Aloha partendo da molteplici dimensioni iniziali del frame e fissando alcuni parametri comuni [1]:

- La dimensione di un pacchetto contenente dati è stata fissata a 16 byte, quella di un ack ad 1 byte.
- La velocità di trasmissione è stata fissata a 0,1 Mbps
- La dimensione temporale di un singolo slot è stata tarata in modo da poter contenere il tempo necessario all'invio dei dati da parte del tag e del relativo ack e arrotondata per eccesso a 0,0015 secondi.
- Il raggio d'azione del reader è di 5 metri con i tag che si dispongono in maniera casuale.

In tabella 7.1 possiamo vedere i risultati di una simulazione in cui si dovevano identificare 50 tag utilizzando le due tipologie differenti di Slotted Aloha (statico e dinamico) e variando il numero di slot iniziali del frame.

| | Numero Slot Iniziali | | | | | | |
|---|---|---|---|---|---|---|---|
| | **10** | **25** | **35** | **50** | **75** | **100** | **200** |
| **BFSA** | 0,7163s | 0,2867s | 0,2556s | 0,2865s | 0,3360s | 0,4107s | 0,6172s |
| **DFSA** | 0,2522s | 0,2352s | 0,2202s | 0,2235s | 0,2371s | 0,2605s | 0,3867s |

Tabella 7.1: *Simulazione Slotted Aloha con 50 tag*

[1] Questi parametri sono stati fissati basandosi su reali specifiche tecniche consultabili anche nelle specifiche dell'EPCglobal

Ogni singolo risultato riportato in tabella rappresenta il tempo medio (espresso in secondi) di riconoscimento dei 50 tag con quella particolare configurazione ed è stato ottenuto ripetendo la simulazione per 100 volte.

Si può notare da subito come il Dinamic Frame Slotted Aloha sia più veloce del protocollo con frame statico indipendentemente dalla dimensione iniziale del frame. Inoltre, in base ai dati raccolti durante le simulazioni, si è potuta notare una maggiore stabilità dei risultati del DFSA nei confronti del BFSA; i risultati ottenutinelle nelle 100 simulazioni di ognuna delle configurazioni con protocollo statico hanno infatti un'indice di dispersione che arriva anche al 19% (contro l8% del dinamico) a testimonianza di una forte fluttualità delle prestazioni.

Aumentando il numero di tag da identificare, i risultati confermano sempre di più i risultati ottenuti nelle simulazioni precedenti.

| | Numero Slot Iniziali | | | | | | |
|---|---|---|---|---|---|---|---|
| | **25** | **50** | **75** | **100** | **125** | **150** | **200** |
| **BFSA** | 0,8255s | 0,5272s | 0,5317s | 0,6117s | 0,6734s | 0,7245s | 0,8677s |
| **DFSA** | 0,4844s | 0,4626s | 0,4357s | 0,4535s | 0,4502s | 0,4798s | 0,5279s |

Tabella 7.2: *simulazione Slotted Aloha con 100 tag*

| | Numero Slot Iniziali | | | | | | |
|---|---|---|---|---|---|---|---|
| | **50** | **75** | **100** | **125** | **150** | **200** | **300** |
| **BFSA** | 0,9036 | 0,8144s | 0,8252s | 0,8650s | 0,9478s | 1,0754s | 1,3621s |
| **DFSA** | 0,7068s | 0,6928s | 0,6594s | 0,6613s | 0,6709s | 0,6909s | 0,7006s |

Tabella 7.3: *simulazione Slotted Aloha con 150 tag*

Analizzando le tabelle riportate e la figura 7.1, si può notare anche un'altra caratteristica comune: in tutti e quattro gli scenari, la migliori configu-

| | Numero Slot Iniziali | | | | | | |
|---|---|---|---|---|---|---|---|
| | **75** | **100** | **125** | **150** | **200** | **300** | **400** |
| **BFSA** | 1,1279s | 1,0853s | 1,1024s | 1,2715s | 1,4397s | 1,5717s | 1,8666s |
| **DFSA** | 0,9337s | 0,9253s | 0,8860s | 0,8976s | 0,9113s | 0,9473s | 1,0546s |

Tabella 7.4: *simulazione Slotted Aloha con 200 tag*

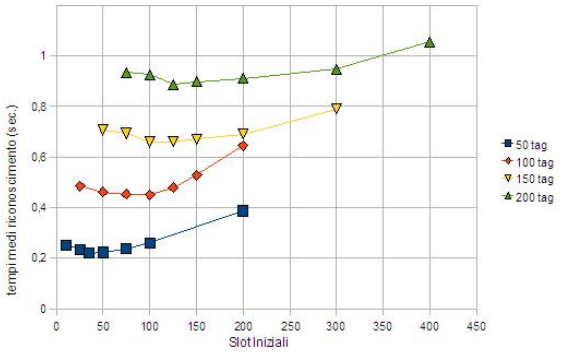


Figura 7.1: *Tempi medi di identificazione dei tag utilizzando DFSA in funzione della framesize iniziale*

razioni, oltre a basarsi sul protocollo dinamico, sono quelle che partono con una dimensione del frame più piccola rispetto al numero di tag da identificare o, meglio ancora, compresa tra il numero dei tag e la metà di questi.

Questa osservazione può risultare utile ai fini di una installazione di una rete RFID; se è infatti vero che un reader non può sapere, per definizione propria del sistema, il numero di tag che si presenteranno ad ogni ciclo di interrogazione può però farne una stima basandosi su esperienze passate e rimodificare dinamicamente le proprie impostazioni di base per cercare di ottimizzare le prestazioni.

Nelle simulazioni descritte sopra, sono stati osservati anche gli aspetti riguardanti la quantità di slot necessaria al riconoscimento dei tag in ques-

tione. I risultati, mostrati nelle tabelle 7.5 7.6 7.7, confermano la superiorità del DFSA e la leggera ottimizzazione delle prestazioni partendo con dimensioni del frame leggermente inferiori al numero di tag da identificare.

| | Numero Slot Iniziali | | | | | | |
|---|---|---|---|---|---|---|---|
| | **10** | **25** | **35** | **50** | **75** | **100** | **200** |
| **BFSA** | 440,5 | 196,5 | 172,2 | 198,5 | 234,0 | 289,0 | 436,0 |
| **DFSA** | 162,2 | 152,2 | 143,3 | 145,9 | 155,7 | 171,9 | 258,0 |

Tabella 7.5: *Slotted Aloha con 50 tag: numero medio degli slot totali utilizzati*

| | Numero Slot Iniziali | | | | | | |
|---|---|---|---|---|---|---|---|
| | **25** | **50** | **75** | **100** | **125** | **150** | **200** |
| **BFSA** | 573,2 | 355,0 | 361,5 | 426,0 | 467,5 | 501,0 | 616,0 |
| **DFSA** | 317,5 | 303,9 | 286,5 | 298,8 | 297,9 | 317,6 | 349,9 |

Tabella 7.6: *Slotted Aloha con 100 tag: numero medio degli slot totali utilizzati*

| | Numero Slot Iniziali | | | | | | |
|---|---|---|---|---|---|---|---|
| | **50** | **75** | **100** | **125** | **150** | **200** | **300** |
| **BFSA** | 601,5 | 550,5 | 566,0 | 598,7 | 655,5 | 750,0 | 954,0 |
| **DFSA** | 465,8 | 457,4 | 435,4 | 438,0 | 444,3 | 457,5 | 525,4 |

Tabella 7.7: *Slotted Aloha con 150 tag: numero medio degli slot totali utilizzati*

## 7.2 EPCGlobal

Il protocollo di comunicazione EPCglobal, come ampiamente descritto nel cap. 4, può essere visto come una modfica di DFSA: come quest'ultimo, infatti, divide il frame di lettura in intervalli di tempo, di quantità uguale

| | Numero Slot Iniziali | | | | | | |
|---|---|---|---|---|---|---|---|
| | **75** | **100** | **125** | **150** | **200** | **300** | **400** |
| **BFSA** | 769,2 | 735,0 | 752,5 | 598,7 | 878,0 | 1000,0 | 1110,0 |
| **DFSA** | 617,4 | 612,5 | 586,4 | 594,7 | 604,2 | 629,2 | 700,7 |

Tabella 7.8: *Slotted Aloha con 200 tag: numero medio degli slot totali utilizzati*

ad una potenza di due, scanditi dal comando QueryRep, ed offre la possibiltà aggiuntiva di ridimensionare il frame senza attendere la terminazione dello stesso. Quest'ultima possibilità è stata largamente utilizzata durante le simulazioni effettuate con NS-2. Fissato, infatti, il numero di slot iniziali sono state simulate diverse varianti del protocollo cambiando l'istante in cui analizzare i risultati fino a quel momento ottenuti ed, eventualmente, ridimensionare il frame.

Come abbiamo fatto per lo Slotted Aloha sono state fissate alcune impostazioni di base valide per tutte le simulazioni effettuate:

- Basandosi sulle specifiche ufficiali dell'EPCglobal sono state fissate le seguenti dimensioni dei pacchetti:
  - Query = 3 byte;
  - QueryRep e QueryAdjust = 1 byte;
  - Ack = 1 byte;
  - RN16 = 1 byte;
  - PC+EPC+CRC16 = 16 byte;
- La velocità di trasmissione è stata fissata a 0,1 Mbps.

| | Punto in cui ricalcolare la size | | | | | | |
|---|---|---|---|---|---|---|---|
| | **10%** | **15%** | **25%** | **50%** | **75%** | **90%** | **100%** |
| **Q = 3** | 0,3713s | 0,2667s | 0,2075s | 0,1758s | 0,1720s | 0,1650s | 0,1619s |
| **Q = 4** | 0,3348s | 0,2590s | 0,2107s | 0,1767s | 0,1692s | 0,1620s | 0,1602s |
| **Q = 5** | 0,3457s | 0,2608s | 0,2100s | 0,1727s | 0,1676s | 0,1622s | 0,1595s |
| **Q = 6** | 0,3511s | 0,2600s | 0,2159s | 0,1943s | 0,1750s | 0,1830s | 0,1848s |
| **Q = 7** | 0,3437s | 0,2654s | 0,2143s | 0,1994s | 0,2008s | 0,2278s | 0,2390s |

Tabella 7.9: *Prestazioni EPCglobal con 50 tag*

- Il tempo di attesa del reader prima di stabilire l'assenza di messaggi nello slot in corso ed inviare una QueryRep/QueryAdjust è di 0,001 secondi.

- Il raggio d'azione del reader è di 5 metri con i tag che si dispongono in maniera casuale.

La tabella 7.9 mostra i risultati di una simulazione in cui si dovevano identificare 50 tag per mezzo del protocollo EPCglobal e con diverse configurazioni di base del protocollo stesso. Come si può notare, infatti, abbiamo provato lo stesso scenario partendo da differenti dimensioni iniziali del frame (espresse tramite il parametro Q[2]) e per ognuna di questa abbiamo fissato il punto, espresso in percentuale, in cui analizzare i risultati fino a quel momento ottenuti ed eventualmente inviare la QueryAdjust per ridimensionare il frame.

Osservando i risultati delle simulazioni si può subito notare un aspetto importante: è consigliabile utilizzare le QueryAdjust nelle parti finali del frame se non addiruttura al termine di questo nel caso in cui si utilizzino

[2]Si ricorda, dal cap. 4, che Q è il parametro utilizzato dal reader, e passato ai, tag per identificare la dimensione del frame pari a $2^Q$

dimensioni iniziali relativamente piccole. La motivazione di questo può esser legata al fatto che, facendo mediante Chebishev l'analisi degli slot fino a quel momento controllati in uno stato avanzato del frame, abbiamo una mole di informazioni molto più grande da poter analizzare con una maggiore attendibilità dei risultati e conseguente precisione più elevata riguardo alla successiva dimensione del frame.

Quest'ultima osservazione viene avvalorata anche da un altro dato. La grande quantità di simulazioni effettuata per ogni singola configurazione ha evidenziato una forte variabilità dei risultati nei casi in cui si procedeva all'utilizzo di Chebishev, e ad una eventuale QueryAdjust, in fasi iniziali di lettura del frame. Ad esempio, il risultato ottenuto dalla media delle simulazioni con configurazione Q=3 e analisi del frame al 10% porta con se un indice di dispersione superiore al 40% e i risultati oscillano da un minimo di 0,206s a un massimo di 1,084s. Aumentando la dimensione iniziale del frame i risultati migliorano ma non in maniera determinante mantenendo un indice che si aggira intorno al 30%. Evidentemente, la minore quantità di dati su cui lavorare porta a considerazioni troppo aleatorie riguardante il numero previsto di tag e il conseguente ridimensionamento del frame per minimizzare le collisioni.

La situazione generale descritta, cambia leggermente partendo da dimensioni grandi del frame (quali Q=7) rispetto al numero di tag da identificare; in questo caso la posizione migliore in cui effettuare la QueryAdjust tende a spostarsi leggermente verso il centro. Evidentemente, superato un certo numero di slot, il tempo necessario per ascoltare gli slot successivi del frame risulta troppo elvato rispetto alla quantità di informazione che può

| | Punto in cui ricalcolare la size | | | | | | |
|---|---|---|---|---|---|---|---|
| | **10%** | **15%** | **25%** | **50%** | **75%** | **90%** | **100%** |
| **Q = 3** | 0,6043s | 0,4893s | 0,4117s | 0,3598s | 0,3404s | 0,3304s | 0,3306s |
| **Q = 4** | 0,5974s | 0,4907s | 0,4093s | 0,3564s | 0,3425s | 0,3313s | 0,3304s |
| **Q = 5** | 0,6064s | 0,4876s | 0,4055s | 0,3498s | 0,3340s | 0,3286s | 0,3258s |
| **Q = 6** | 0,5932s | 0,4851s | 0,4079s | 0,3513s | 0,3330s | 0,3269s | 0,3245s |
| **Q = 7** | 0,5889s | 0,4647s | 0,4316s | 0,3746s | 0,3563s | 0,3712s | 0,3800s |
| **Q = 8** | 0,5842s | 0,4839s | 0,4204s | 0,3922s | 0,4238s | 0,4587s | 0,4850s |

Tabella 7.10: *Prestazioni EPCglobal con 100 tag*

aggiungere a quella già ottenuta nella prima parte.

Occorre comunque notare che il punto in cui ricalcolare la dimensione del frame viene stabilito all'inizio dell'intero processo di identificazione e rimane tale per tutto il suo svolgimento Fissare quindi il calcolo di Chebyshev in un punto intermedio del frame può portare a buoni risultati nell'immediato (in quanto abbiamo una dimensione del frame grande ) ma può portare a prestazioni peggiori non appena questo viene ridotto in numero di slot.

Quest'ultima osservazione è probabilmente collegata ad un altro dato che evdiente mostrato dalla tabella e che conferma, in un certo senso, quanto già avevamo intuito con lo Slotted Aloha: le prestazioni migliori si ottengono fissando la dimensione iniziale del frame in un numero inferiore a quello dei tag presenti, o quanto meno previsti, in un determinato processo di identificazione.

Le tabelle 7.10, 7.11, 7.12 mostrano rispettivamente i risultati ottenuti in scenari con 100, 150 e 200 tag.

Una prima osservazione da notare, aumentando il numero di tag da identificare, riguarda l'istante in cui effettuare i calcoli la QueryAdjust. Abbiamo appena che le prestazioni migliori si ottengono utilizzando Chebyshev

| | Punto in cui ricalcolare la size | | | | | | |
|---|---|---|---|---|---|---|---|
| | **10%** | **15%** | **25%** | **50%** | **75%** | **90%** | **100%** |
| **Q = 3** | 0,8192s | 0,7269s | 0,6241s | 0,5484s | 0,4917s | 0,5009s | 0,5026s |
| **Q = 4** | 0,8330s | 0,7347s | 0,6237s | 0,5440s | 0,4950s | 0,4990s | 0,5000s |
| **Q = 5** | 0,8400s | 0,7360s | 0,6217s | 0,5450s | 0,4930s | 0,4990s | 0,4997s |
| **Q = 6** | 0,8183s | 0,7432s | 0,6246s | 0,5419s | 0,4889s | 0,4930s | 0,4937s |
| **Q = 7** | 0,8062s | 0,7365s | 0,6148s | 0,5348s | 0,5006s | 0,5076s | 0,5219s |
| **Q = 8** | 0,8104s | 0,7497s | 0,6270s | 0,5714s | 0,5251s | 0,5424s | 0,5674s |

Tabella 7.11: *Prestazioni EPCglobal con 150 tag*

| | Punto in cui ricalcolare la size | | | | | | |
|---|---|---|---|---|---|---|---|
| | **10%** | **15%** | **25%** | **50%** | **75%** | **90%** | **100%** |
| **Q = 4** | 1,1119s | 0,9649s | 0,8164s | 0,7108s | 0,6800s | 0,5559 | 0,4542 |
| **Q = 5** | 1,0857s | 0,9651s | 0,8199s | 0,6941s | 0,6683s | 0,6482s | 0,6523s |
| **Q = 6** | 1,0892s | 0,9704s | 0,8174s | 0,6941s | 0,6683s | 0,6482s | 0,6523s |
| **Q = 7** | 1,1037s | 0,9715s | 0,8200s | 0,7030s | 0,6646s | 0,6530s | 0,6560s |
| **Q = 8** | 1,1110s | 0,9289s | 0,8628s | 0,7718s | 0,7095s | 0,7430s | 0,7780s |
| **Q = 9** | 1,1018s | 0,9380s | 0,8327s | 0,7915s | 0,8595s | 0,9188s | 0,9750s |

Tabella 7.12: *Prestazioni EPCglobal con 200 tag*

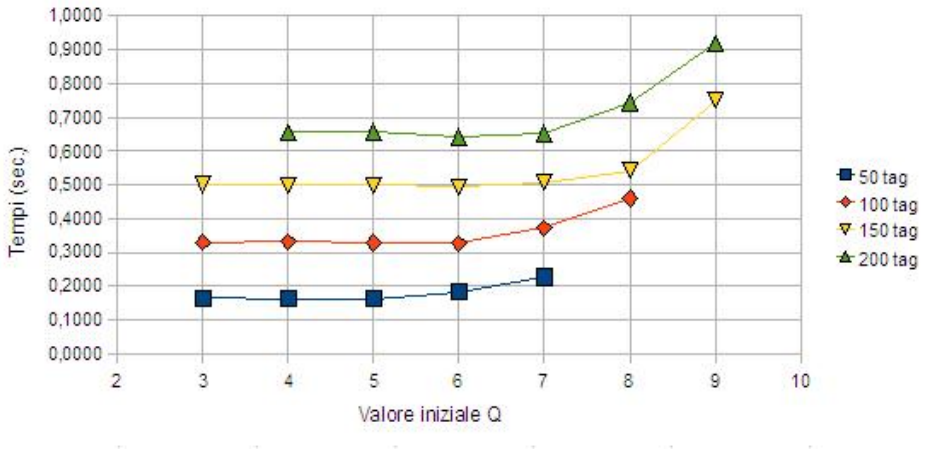


Figura 7.2: *Prestazioni EPCGlobal in funzione di Q con QueryAdjust al 90%*

al termine del frame.Aaumentando il numero di tag notiamo, invece, che, seppur con differenze minime, i risultati migliori si ottengono effettuando la QuqryAdjust in posizioni comprese tra il 75% e il 90% del frame.

L'aspetto particolare che, comunque, emerge fortemente da queste nuove simulazioni è legato alla dimensione iniziale del frame: nonstante un aumento del numero di tag presenti nell'area di lettura del reader, le migliori prestazioni si ottengono mantenendo dimensioni relativamente basse della framesize e in tutti casi i migilori risultati si ottiengono per Q=6 e Q=5.

Quest'ultimo aspetto può rappresentare un punto di forza dell'EPCglobal: possiamo infatti affermare che fissando la dimensione iniziale del frame a Q=6 o Q=5 e stabilendo di fare l'eventuale QueryAdjust in un punto compreso tra il 75% e il 100% del frame stesso, otterremo risultati in generale ottimali indipendentemente dal numero di tag da riconoscere.

La figura 7.2 descrive le prestazioni di EPCglobal al variare della dimensione iniziale $2^Q$ del frame e fissando la QueryAdjust al 90% del frame evidenziando come le prestazioni migliori si ottengano partendo da dimensioni

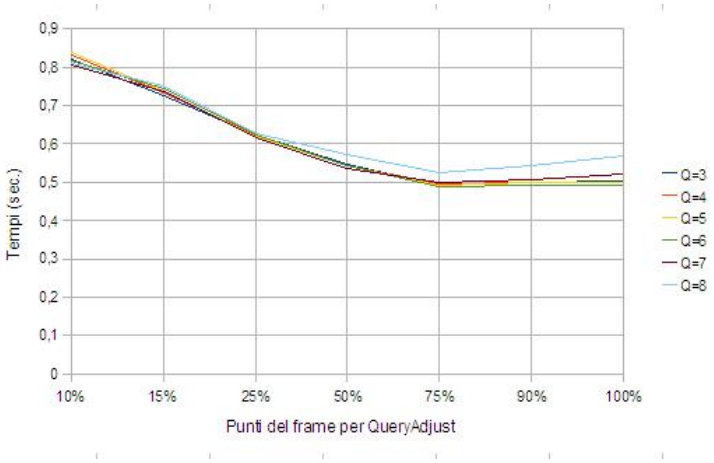


Figura 7.3: *Prestazioni EPCglobal al variare del momento in cui effettuaree la QueryAdjust*

del frame relativamente basse.

## 7.3 EPCglobal vs DFSA

Dalle simulazioni descritte negli ultimi due paragrafi emergono in maniera abbastanza nitida le migliori performance dello stadard EPCglobal nei confronti di DFSA, e, a maggior ragione, di BFSA.

I grafici in figura 7.4 descrivono l'andamento dei due protocolli in funzione della lunghezza iniziale del frame e con QueryAdjust, per l'EPCglobal, al 90% del frame. Per fare un confronto ancora più accurato, sono state anche fatte simulazioni dello Slotted Aloha con dimensioni iniziali del frame uguali a potenze di due.

I risultati che emergono dai grafici mostrano in maniera evidente la superiorità del protocollo EPCGlobal. Questo infatti, non solo offre prestazioni migliori partendo dalla medesima framesize, ma riesce a mantenersi su liv-

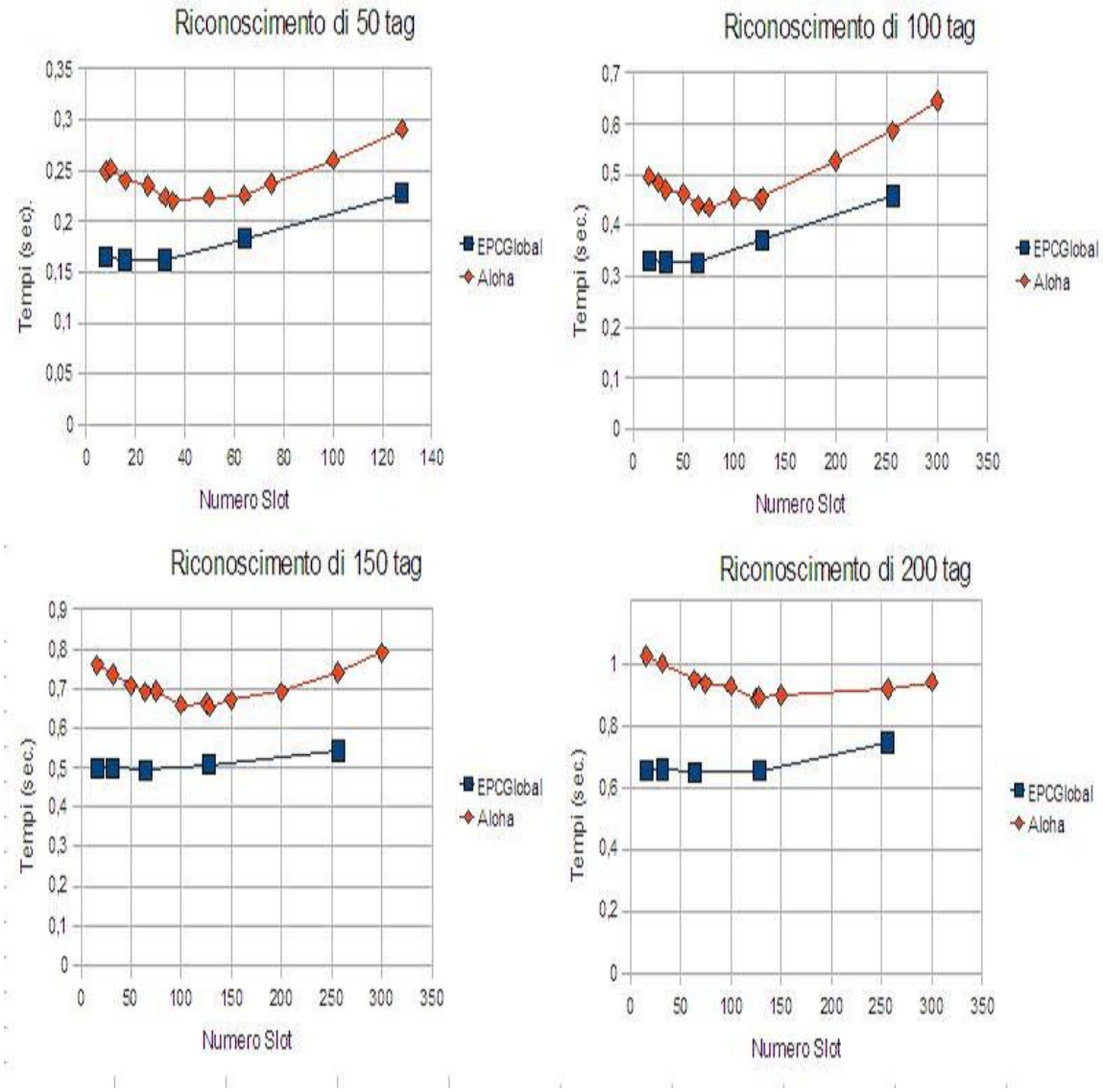


Figura 7.4: *Prestazioni EPCglobal e DFSA al variare della dimensione iniziale del frame*

elli quantomeno uguali a quelle dello Slotted Alhoha anche nei casi pessimi. Inoltre, le curve che rappresentano DFSA tendono ad avere un andamento generale meno regolare rispetto a quello di EPCGlobal, che si mantiene su prestazioni più o meno costanti, almeno per dimensioni iniziali relativamente piccole, e mantiene comunque una crescita dei tempi più lieve.

Quest'ultima caratteristica può risultare molto efficace durante una simulazione RFID: durante un ciclo di identificazione, infatti, il reader non ha alcuna conoscenza riguardo al numero di tag che andrà ad identificare. Come è stato più volte sottolineato, infatti, la dimensione iniziale del frame viene generalmente scelta basandosi su esperienze passate del reader stesso. La forte variabilità delle situazioni in cui ci si può trovare può, però, portare a scenari in cui si devono identificare un numero di tag completamente diverso da quello atteso, con una conseguente dimensione del frame impostata dal reader non ottimale per ottenere prestazioni accettabili. La forte stabilità di EPCGlobal garantisce, infatti, buone prestazioni anche in caso di scelte sbagliate riguardo la dimensione iniziale del frame.

I grafici in figura 7.5 mostrano il numero di slot utilizzati dai due protocolli in funzione della dimensione iniziale del frame. La caratteristica particolare è che, a differenza delle prestazioni temporale mostrate nelle pagine precedenti, da questo punto di vista EPCglobal e Slotted Aloha offrono praticamente gli stessi risultati per qualsiasi tipo di combinazione.

Questa osservazione può portare a conclusioni molto importanti riguardo EPCGlobal: se il numero di slot utilizzato dai due protocolli per un intero ciclo di identificazione è lo stesso, ma il tempo impigato è differente, vuol dire che a far la differenza è la diversa lunghezza, in termini temporali, dei

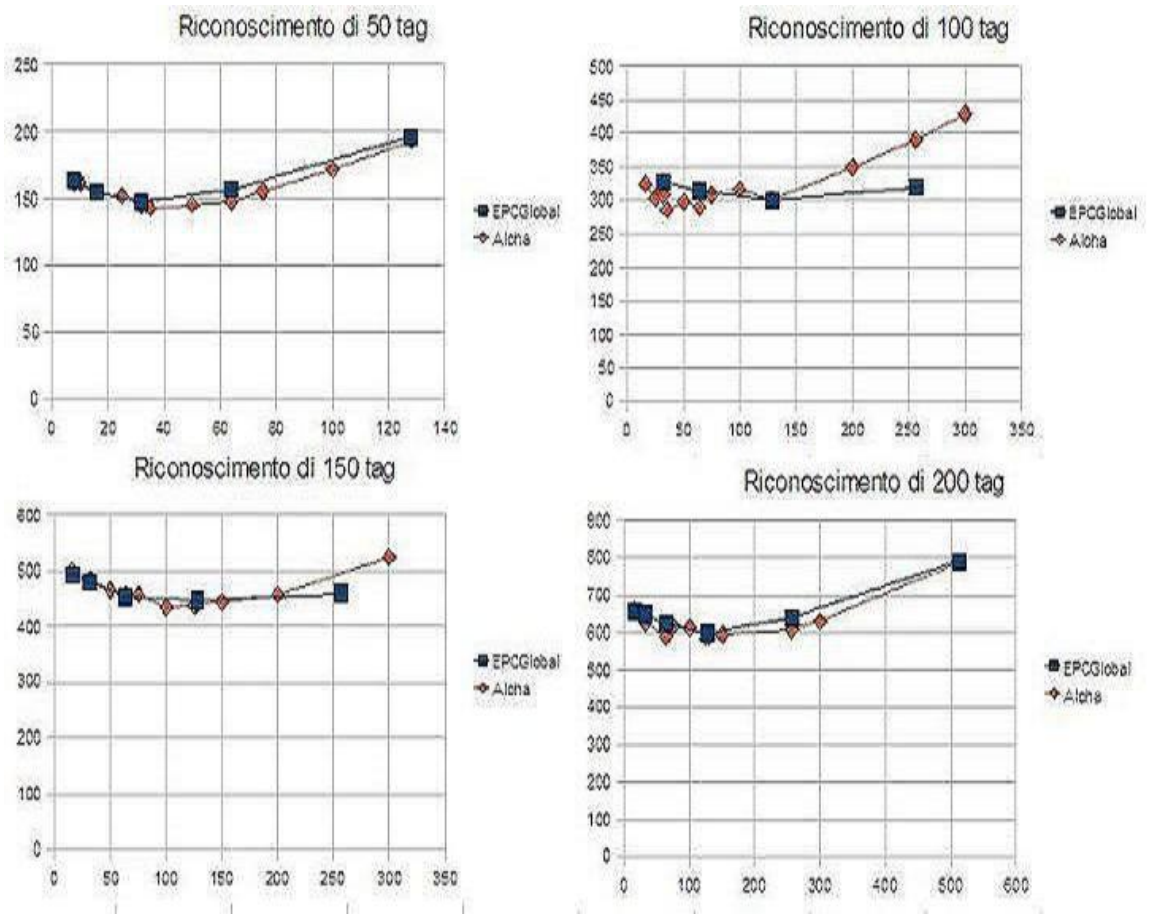


Figura 7.5: *Numero totale di slot utilizzati in funzione della dimensione iniziale del frame*

singoli slot dei due algoritmi. EPCGlobal, infatti, ha la caratteristica di interrompere anticipatamente l'ascolto da parte del reader in un ipotetico slot, se risulta vuoto o con collisione, per passare direttamente al successivo.

Possiamo dunque concludere che EPCGlobal migliora di gran lunga le prestazioni offerte da Slotted Aloha non tanto per la possibilità di ridimensionare in corsa la dimensione del frame (operazione che, come abbiamo visto, viene fatta nelle parti conclusive, proprio come per DFSA), ma per l'introduzione di una sorta di handshake tra reader e tag che avviene all'inizio di ogni slot e che permette di decifrarne lo stato.

# Capitolo 8

# Conclusioni

I sistemi di identificazione a radiofrequenza stanno entrando a far parte della quotidianetà dell'uomo in maniera sempre più massiccia. Oltre agli ormai collaudati Telepass e ai sistemi di antitacchieggio dei negozi, gli RFID si stanno diffondendo come sistema di identificazione di oggetti, in sotituzione dei barcode, di persone, nei documenti di identità e di animali. Molti i settori, quindi, che già utilizzano o che presto utlizzeranno questo sistema: supermercati, magazzini, biblioteche, reti di trasporto, metropolitane.

L'uso così massiccio di questa tecnologia, anche a livello globale, ha portato diverse industrie alla necessità di accordarsi su un unico standard internazionale a cui attenersi: l'EPCglobal. Questo standard prevede anche un protocollo, basato sul modello di Aloha, di alto livello per gestire la comunicazione tra reader e trasponder ed evitare la tag-collision.

Simulato in uno scenario RFID appositamente implementato per mezzo di NS-2, EPCglobal si è dimostrato migliore rispetto ad uno Slotted Aloha con frame dinamico sia in termini di velocità che di efficienza: EPCglobal sembrerebbe infatti garantire migliori prestazioni anche con la solita configu-

razione iniziale indipendentemente dal numero di oggetti che dovranno essere identificati.

Per questi motivi EPCGlobal si sta dimostrando un buon punto di riferimento per le aziende che vogliano equipaggiare i loro prodotti con etichette RFID: questa teconologia è stata infatti progettata per ambiti, come il tracciamento nei magazzini, dove l'obiettivo principale è la massima facilità e velocità di lettura anche di uno svariato numero di etichette.

Nonostante stiano cominciando a diffondersi in maniera rapida, gli RFID sono, comunque, ancora al centro di numerosissimi studi sotto molti punti di vista e potrebbero essere oggetto di ulteriori ricerche. Prendendo il problema della tag collision studiato in questa tesi, ad esempio, oltre a quelli basati su Aloha si potrebbero affrontare delle analisi sulle prestazioni di protocolli basati su alberi binari a cui si è fatto riferimento nel capitolo 3.

Un'altra fonte di studio è rappresentata dalla reader collision che costituisce un'altra grande problematica di questo sistema. La grande diffusione dei sistemi RFID porterà inevitabilmente all'installazione e all'utilizzo di uno svariato numero di reader anche in zone relativamente ristrette con conseguenti sovrapposizioni dei segnali che danno orgine alla reader collision.

Come abbiamo visto, poi, gli RFID potrebbero col tempo anche gestire dati particolarmente delicati dal punto di vista della riservatezza con conseguenti preoccupazioni dei garanti della privacy e non solo. La sicurezza nel campo degli RFID sta diventando, per questo motivo, oggetto di molti studi da parte di ricercatori e sviluppatori e potrebbe rappresentare la base anche per un ulteriore analisi futura.

# Bibliografia

[1] Stefano Quintarelli. The internet of things. *Equi Liber*, Milano, 9 Luglio 2003:1–14, 2003.

[2] Katerine Albrecht. Una società etichettata. Rivista Le Scienze n. 483, Novembre 2008.

[3] Alessio Conti. Il problema della collisione tra readers nei sistemi rfid. Master's thesis, University di Pisa, 2006-2007.

[4] Maurizio A. Bonuccelli, Francesca Lonetti, and Francesca Martelli. Instant collision resolution for tag identification in rfid networks. *Ad Hoc Networks*, 5(8):1220–1232, 2007.

[5] Comunicato garante privacy per l'uso in italia delle etichette rfid - pagina web: http://www.garanteprivacy.it/garante/doc.jsp?id=1109493.

[6] D.W. Engels and S.E. Sarma. The reader collision problem. *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, 3:6 pp. vol.3–, Oct. 2002.

[7] Nitin H. Vaidya Harles-Francois Natali. Expanding horizon and capture effect in rfid systems. Technical report, University of Illinois at Urbana-Champaingn, dicembre 2007.

[8] Kin Seong Leong, Mun Leng Ng, Alfio R. Grasso, and Peter H. Cole. Synchronization of rfid readers for dense rfid reader environments. In *SAINT Workshops*, pages 48–51, 2006.

[9] Sito ufficiale epcglobal:www.epcglobalinc.org.

[10] EPCglobal.inc. Specification for rfid air interface - epc radio-frequency identity protocols, class-1 generation-2 uhf rfid, protocol fo communiocations at 860 mhz - 960mhz, version 1.1.0; www.epcglobalinc.org, 2006.

[11] Juan Ignacio Aguierre. Epcglobal: A universal standard. Technical report, B.S. Mechanical & Electrical Engineering, ITESM Massachussetts Istitute of Technology, June 2007. Cambridge, MA 02142.

[12] Cheng Jin, Sung Ho Cho, and Ki Yong Jeon. Performance evaluation of rfid epc gen2 anti-collision algorithm in awgn environment. *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*, pages 2066–2070, Aug. 2007.

[13] Mark Claypool Jae Chung. Ns by example - tech. report - http://nile.wpi.edu/ns/. WPI Worcester Polytechnic Institute, Computer Science.

[14] Ns-2.26 sources original documentation: http://www-rp.lip6.fr/ns-doc/ns226-doc/html/.

[15] Andrea Pasquini. Alogoritmi broadcast per reti intervicolari in contesti urbani. Master's thesis, Università di Pisa, 2005/2006.

[16] Daniel Mahrenholz and Svilen Ivanov. Real-time network emulation with ns-2. In *DS-RT*, pages 29–36, 2004.

[17] Giorgio Ventre. Network simulator ns-2 - reti di calcolatori 2. Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II.

[18] Eitan Altman and Tania Jimenez. Ns simulator for beginners. Technical report, univ. de Los Andes, Merida, Venezuela and ESSI, 4 December, 2003.

[19] Sandro Petrizzelli. Network simulator: - web doc. - http://users.libero.it/sandry/umts/ns/ns.htm.

# Ringraziamenti

*Finalmente eccomi qua. Dopo anni di studi e di sacrifici sono finalmente arrivato al raggiungimento di un obiettivo voluto con tanta determinazione e a quello che resterà sicuramente uno dei giorni più importanti della mia vita.*

*Se sono riuscito in tutto questo il merito è non soltanto mio, ma anche di molte persone che mi sono state vicine in questi anni come dei veri e propri compagni di viaggio e senza i quali probabilmente non ce l'avrei mai fatta.*

*Il primo grande, immenso ringraziamento va chiaramente a mia mamma, a Lorenzo, a Guido e a tutti miei parenti per aver creduto in me in tutti questi anni, in alcuni casi molto più di quanto abbia fatto io stesso, per avermi sostenuto economicamente ma soprattutto moralmente in tutti gli studi universitari.*

*Ringrazio quindi il prof. Maurizio Bonuccelli sia per avermi offerto la possibilità di lavorare con lui a questa tesi di laurea, che si è rivelata molto interessante e coinvolgente ben oltre le più rosee aspettative, sia per la grande disponibilità mostratami in questi mesi di lavoro. Sempre parlando di tesi, un doveroso ringraziamento alla dott. Francesca Martelli per avermi seguito e aiutato molto durante*

*tutti questi mesi sopportando anche alcuni miei momenti di, spero comprensibile, nervosismo.*

*Ripensando a tutte le ore passate in territorio pisano non posso non citare i componenti del gruppo storico, che con il suo 'gioco di squadra', sbaragliava (...forse sto un po' esagerando..!!) tutti gli esami e con i quali ho passato spensierate mattinate in facoltà: il Proc, Luca, Tiziano, Andrea, Eva. Un sentito grazie anche a tutti gli altri amici frequentati durante l'università: Elena, in realtà amica di vecchia data, Angela, Manuel e molti altri tra cui le ultimissime conoscenza fatte in dipartimento durante la tesi come Donatella e lo Strollo.*

*Infine, ultimi in questa pagina ma sicuramente non nei miei pensieri, gli amici di tutti i giorni. Quelli che da sempre ci sono e che sempre ci saranno: Tommaso, l'imprenditore Flo, il Pre, Bafio, il Mos, Alessandra, Valentina, Teresa, Paola, Rachele, Sara e tutti gli altri...*

*Si lo so, avrò sicuramente dimenticato qualcuno...ma sto andando di fretta... il tempo stringe....devo consegnare...perdonatemi...sarà per la prossima...!!!*

*Davvero grazie a tutti! Se oggi è un giorno tanto importante il merito è anche vostro...!*