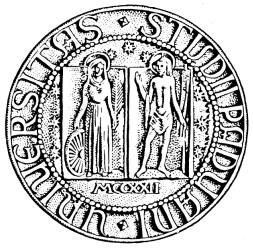


Università degli Studi di Padova
Facoltà di Ingegneria
Dipartimento di Ingegneria dell'Informazione

Tesi Specialistica

# Sviluppo di un framework basato su ULLA e CalRadio SDR per ottimizzazioni cognitive in reti 802.11

**Relatore:** Ch.mo Prof. Michele Zorzi

**Correlatore:** Riccardo Manfrin

**Laureando:** Luca Boscato

Anno Accademico 2009/2010

*"Desidero innanzitutto ringraziare il Professor Michele Zorzi, l'Ing. Riccardo Manfrin e l'Ing. Andrea Mior per la disponibilità e i consigli offertimi durante la stesura di questo lavoro. Inoltre, vorrei esprimere la mia sincera gratitudine ai miei compagni di corso, in particolare Moreno, Marco e Francesco per i numerosi consigli durante la ricerca. Infine, ho desiderio di ringraziare con affetto la mia famiglia per il sostegno ed il grande aiuto che mi hanno dato ed in particolare Lisa per essermi stata vicino ogni momento durante questo anno di lavoro."*

**Sommario**

Gli avanzamenti tecnologici degli ultimi anni, ci hanno condotto nell'ottica di una fornitura di servizi di rete del tipo "anytime, anywhere", assicurando comunque una comunicazione istantanea e sicura. Con l'introduzione della tecnologia wireless, i paradigmi sono cambiati: l'accesso alla rete é passato da statico a dinamico e in movimento, le infrastrutture da centralizzate sono divenute distribuite e la rete, é passata dall'essere un sistema passivo ad uno attivo.

Le tecnologie di rete attuali, non essendo state sviluppate sulla base di tali principi, limitano drasticamente l'adattività e le prestazioni delle comunicazioni. Questi fattori limitativi si trovano su diverse componenti del sistema di comunicazione: sui nodi della rete, nel protocollo di comunicazione utilizzato, nelle politiche di gestione, etc. e sono dovuti al disegno originario della rete, per la quale non erano previsti il supporto alla mobilitá e alla sicurezza.

In generale, se ogni nodo della rete apporta delle modifiche al proprio "modus operandi", senza informare gli altri nodi circa il proprio stato attuale, il risultato sará una serie di miglioramenti locali che non per forza garantiscono un miglioramento complessivo.

Una rete cognitiva, si occupa della riconfigurazione dei propri parametri operativi e della gestione delle risorse, richiedendo il supporto da parte di tutti gli elementi della stessa, in qualità di agenti osservatori e attuatori.

L'implementazione di una rete cognitiva si attua mediante il ciclo di cognizione, un processo interno alla rete volto ad incrementare la qualità del servizio (Qos), attraverso un'allocazione ottimizzata delle risorse. Quest'ultima avviene tramite la definizione di scelte e soluzioni non necessariamente in accordo con gli obiettivi locali della rete, ma garantendo la massimizzazione delle funzioni obiettivo che descrivono i requisiti del sistema a livello globale.

La tesi si occupa dello sviluppo di un framework per ottimizzazioni cognitive, basto sul middleware ULLA e sul dispositivo *CalRadio* SDR.

L'ambito della ricerca si integra nel progetto Europeo ARAGORN per l'implementazione di una rete cognitiva.

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Capitolo 1

# Reti Cognitive

## 1.1 Introduzione

I requisiti e le aspettative circa i servizi di rete sono in continua evoluzione. Con l'introduzione della tecnologia wireless, i paradigmi sono cambiati: l'accesso alla rete é passato da statico a dinamico e in movimento, le infrastrutture da centralizzate sono divenute distribuite e la rete, é passata dall'essere un sistema passivo ad uno attivo.

Gli avanzamenti tecnologici degli ultimi anni, ci hanno fatto fare molti passi avanti nell'ottica di una fornitura di servizi del tipo "anytime, anywhere", assicurando comunque una comunicazione istantanea e sicura. Tutta questa innovazione é comunque limitata dal disegno originario della rete Internet (e del protocollo TCP/IP stesso), per i quali non erano previsti il supporto alla mobilitá e alla sicurezza.

Pertanto, ogni avanzamento tecnologico necessita di un aumento nella complessitá dell'infrastruttura con conseguenti limitazioni nelle prestazioni.

Il principale motivo dell'inefficienza della rete, é dovuto alla difficoltá di configurare e gestire la rete: queste operazioni non sono automatizzate, ma vengono svolte da operatori umani. Si necessita quindi di meccanismi di apprendimento, manutenzione e riconfigurazione automatici, al fine di ottimizzare le operazioni di mantenimento e migliorare le performance durante il trasferimento dati. Queste caratteristiche, riassumibili nei termini: self-management,self-awareness e self-healing, sono alla base di un nuovo paradigma per le reti Wireless, la $4^{th}$ generazione (4G) [1],con il quale si vuole portare dell'intelligenza sulla rete.

Le tecnologie di networking attuali, limitano le capacitá della rete ad adattarsi, facendo si che le performance dei sistemi di comunicazione siano sub-ottimali. Questi fattori limitativi si trovano su diverse componenti del

sistema di comunicazione: sui nodi della rete, nel protocollo di comunicazione utilizzato, nelle politiche di gestione, etc., i quali non sono in grado di apportare in modo autonomo delle modifiche "intelligenti" al loro modo di operare.

Il problema principale, é che i protocolli di comunicazione nascondo ai singoli elementi della rete lo stato globale del sistema. Vale a dire che ogni cambiamento effettuato da un singolo componente, porterá dei miglioramenti solo per la porzione di rete da lui conosciuta, ovvero solo all'interno dello *scope* di quell'elemento. Pertanto, se ogni nodo della rete apporta delle modifiche al proprio "modus operandi", senza informare gli altri nodi circa il proprio stato attuale, il risultato sará una serie di miglioramenti locali che non per forza garantiscono un miglioramento complessivo. Inoltre, le tecnologie attuali prevedono che un protocollo o un nodo della rete, reagiscano in modo repentino ad un cambiamento, solitamente un problema, senza quindi preoccuparsi di come rispondano gli altri elementi del sistema.

Il termine *cognitive* é per l'appunto collegato a questa abilitá della rete ad essere a conoscenza del proprio stato operativo e dell'essere in grado di tarare i propri parametri funzionali in modo da adattarsi ai cambiamenti dell'ambiente o alle richieste degli utenti. Una rete cognitiva, richiede il supporto da parte di tutti gli elementi della rete (routers,switches, base stations,etc.), i quali devono promuovere delle attivitá di misurazione delle performance al fine di riconfigurare la rete nel modo piú consono. Tutte queste caratteristiche afferiscono al paradigma delle *reti attive* [2], le quali differiscono dalle reti cognitive per l'assenza del processo cognitivo, il quale consiste in azioni di apprendimento e adattamento.

L'implementazione di una rete cognitiva, al fine di migliorare l'utilizzo delle risorse, é un processo atto ad incrementare la qualitá del servizio (QoS), mantenendo l'operativitá su reti eterogenee e sottostando ai limiti imposti dalle tecnologie wireless.

Gli aspetti piú influenti nell'implementazione di una rete cognitiva sono:

- la complessitá della rete
- le limitazioni delle tecnologie wireless
- l'eterogeneitá della rete
- la qualitá del servizio (QoS)

**Complessitá della rete** La complessitá della rete é in funzione del numero di nodi, di routers, di collegamenti, nonché dei protocolli trasmissivi. L'introduzione di collegamenti wireless, ha notevolmente aumentato la complessitá

delle reti, in quanto é completamente cambiata la nozione di connettivitá: i nodi si possono spostare, possono collegarsi e scollegarsi dalla rete in modo irregolare, si possono creare delle mesh e il canale di comunicazione puó essere affetto da interferenze.

**Eterogeneitá** La rete Internet é composta di svariate tecnologie, applicazioni e protocolli di trasmissione. Tuttavia, nel protocollo TCP/IP, non ci sono layer dedicati all'eterogeneitá. Quel che viene fatto é suddividere la rete in segmenti, per ognuno dei quali eseguire poi un qualche tipo ottimizzazione. Sebbene esistano delle soluzioni [3] atte a migliorare le connessioni TCP/IP su reti eterogenee, le performance rimangono comunque deludenti nella maggior parte dei casi.

Migliorare le performance richiede una conoscenza delle tecnologie trasmissive di ogni singolo collegamento tra i nodi interessati nella comunicazione. Pertanto il processo di ottimizzazione dovrebbe essere distribuito su diversi domini, al fine di raggiungere gli obiettivi richiesti agli estremi della connessione (End-to-End goals).

## 1.2 Reti cognitive

L'interesse da parte della comunitá scientifica nei riguardi delle reti cognitive, si é sviluppato solo negli ultimi anni. Il termine "cognizione" , dal latino *cognoscere*, viene usato in molte discipline per descrivere fenomeni strettamente correlati al concetto di conoscenza, intelligenza e apprendimento. Nelle campo delle telecomunicazioni, l'uso di tecnologie cognitive é principalmente motivato dalla complessitá dei sistemi e dall'inefficienza dell'uso di semplici modelli decisionali.

Il concetto di rete cognitiva, si é sviluppato pari passo a quello di *radio cognitiva*. Mitola et Al.[4] discussero la possibilitá di creare una rete composta di radio cognitive, le quali prendevano decisioni in base alla conoscenza ottenuta tramite un processo di apprendimento automatico. In tale ambito, definirono il **ciclo cognitivo** come composto di sei processi: osservazione, orientamento, pianificazione, apprendimento, decisione, azione. Ogni nodo della rete deve quindi essere un elemento ri-configurabile, capace di "osservare" determinati parametri. Le altre operazioni, invece, costituiscono il *cognitive engine*.

## 1.3 Ciclo cognitivo

Il modello del ciclo cognitivo proposto da Mitola et Al. [4], prevede la presenza di sei processi. Tuttavia un modello semplificato, puó essere altrettanto efficace. In particolar modo, come mostrato in Figura 1.1, i processi necessari all'implementazione di un algoritmo cognitivo sono: l'osservazione, l'analisi, la decisione e l'azione.

Originariamente, questo ciclo era utilizzato in ambito militare per comprendere i processi di pensiero degli avversari. Tuttora viene largamente adottato anche al di fuori dell'ambito militare, in contesti che vanno dall'intelligenza artificiale al business planning.

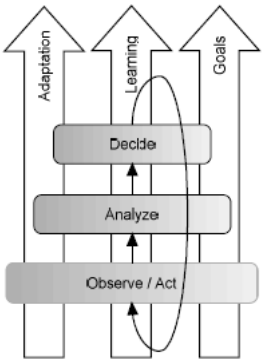



Figura 1.1: Schema base per il processo cognitivo

In Figura 1.1, gli elementi costituitivi del ciclo sono presentati come una piramide, dove alla base sono posizionati quegli elementi funzionali che saranno maggiormente distribuiti nella rete.

Ad esempio, gli elementi atti all'osservazione, dovranno semplicemente eseguire delle misure, come riportare la dimensione della CongestionWindows del protocollo TCP, o misurare il livello del segnale ricevuto.

Per quanto riguarda la funzionalitá "azione", anch'essa sará distribuita tra tutti gli elementi costitutivi della rete, in quanto l'azione intrapresa da questi sará un cambio dei parametri trasmissivi, come ad esempio il rate o il canale radio utilizzato. Questi elementi funzionali sono tipicamente passivi, mentre l'intelligenza sará collocata piú in alto nella piramide.

In questo modo l'analisi e le decisioni intraprese, deriveranno da un processo di feedback ottenuto dagli elementi osservatori e le azioni saranno poi coordinate verso gli attuatori.

Un altro aspetto importante degli elementi funzionali, é la scalabilitá: l'implementazione delle funzionalitá (centralizzata o distribuita) puó essere

ripetuta su diverse scale, sia a livello di singolo nodo che a livello di rete. Ad esempio, nei singoli nodi, gli gli agenti osservatori e attuatori, saranno dei protocolli inter-livello capaci di misurare i parametri interni, mentre il processo di analisi e quello decisionale, saranno dei processi generici che potranno essere eseguiti in parallelo allo stack protocollare.

A livello di rete, invece, l'insieme dei nodi costituisce osservatori e attuatori, dove le informazioni relative ai vari livelli del protocollo vengono collezionate e aggregate da ogni singolo nodo, per poi essere spedite come un unico report al processo di analisi.

La piramide degli elementi funzionali di Figura 1.1 é inoltre tagliata da tre piani:

- Adattamento
- Apprendimento
- Obiettivi

### Adattamento

L'adattamento é un requisito fondamentale per tutti gli elementi, permettendo loro di rispondere ai cambiamenti dell'ambiente. Ad esempio, ad un agente osservatore puó esser chiesto di misurare a determinati istanti temporali dei parametri afferenti lo stack protocollare. In tale contesto, l'adattamento sta nel modificare questa temporizzazione a seconda dello scenario operativo e della configurazione della rete. Di norma, quei parametri che variano con una frequenza alta, saranno comunicati al modulo di analisi con una frequenza altrettanto elevata. Viceversa, quei parametri piú statici, verranno sommarizzati in report meno frequenti.

Va comunque preso in considerazione l'overhead dovuto alla comunicazione tra gli agenti osservatori e il modulo di analisi. Qualora entrambi gli elementi si trovino sullo stesso nodo, l'overhead puó essere trascurato, mentre se l'agente che si occupa dell'analisi si trova su un altro nodo della rete, la segnalazione da parte dei nodi osservatori, puó comportare un notevole dispendio di risorse di rete.

### Apprendimento

L'apprendimento permette agli elementi funzionali costitutivi della piramide di modificare il loro comportamento in base alle esperienze passate.

Esistono due classi di apprendimento:

- Informale

- Formale

L'apprendimento informale é implicito e puó essere svolto durante il ciclo cognitivo. Ad esempio, un dispositivo wireless puó incrementare il proprio rate trasmissivo, qualora veda un degrado nelle performance. Questa informazione é considerata informale.

L'apprendimento formale, invece, proviene da un trasferimento di informazioni dall'entitá che riceve la conoscenza. Tornando all'esempio precedente, quando un nodo modifica il proprio rate e lo comunica agli altri nodi, in questo caso sta diffondendo dell'informazione che porterá gli altri nodi ad un apprendimento formale.

#### Obiettivi

Il piano degli obiettivi, sta ad indicare gli obiettivi finali e locali che devono essere perseguiti attraverso il processo di ottimizzazione cognitiva. Solitamente gli obiettivi vengono espressi a livello end-to-end, e possono corrispondere alle esigenze degli utenti o delle applicazioni in termini di QoS.

## 1.4 Un modello per le reti cognitive

Una rete cognitiva dovrebbe porter garantire, sul medio-lungo periodo, performance end-to-end migliori rispetto a quelle garantite da una rete non cognitiva. Tale rete puó essere usata per ottimizzare l'uso e la distribuzione delle risorse, per il controllo d'accesso, QoS, sicurezza, etc. Le uniche limitazioni sono imposte dalla tecnologia sottostante e dalla flessibilitá del framework cognitivo. Pertanto, le reti cognitive non sono applicabili solamente al dominio delle reti wireless, ma possono anche essere implementate su reti ad-hoc, reti cablate o anche su reti eterogenee.

In [5], viene presentata un'architettura di riferimento per reti cognitive, ideata in base a quanto emerso da varie ricerche in tale ambito.

L'obiettivo principale di tale architettura, é di mantenere la consistenza con con lo stack protocollare TCP/IP. In secondo luogo si vuole creare un framework i cui elementi costitutivi siano semplici da configurare e da gestire, che ottimizzi le performance end-to-end, che implementi un processo cognitivo distribuito e che necessiti di un supporto minimo da parte della rete.

In Figura 1.2, é mostrata l'architettura proposta. Ogni layer del modello TCP/IP viene equipaggiato con dei moduli software aggiuntivi, per implementare gli agenti osservatori e attuatori. In questo modo possono essere

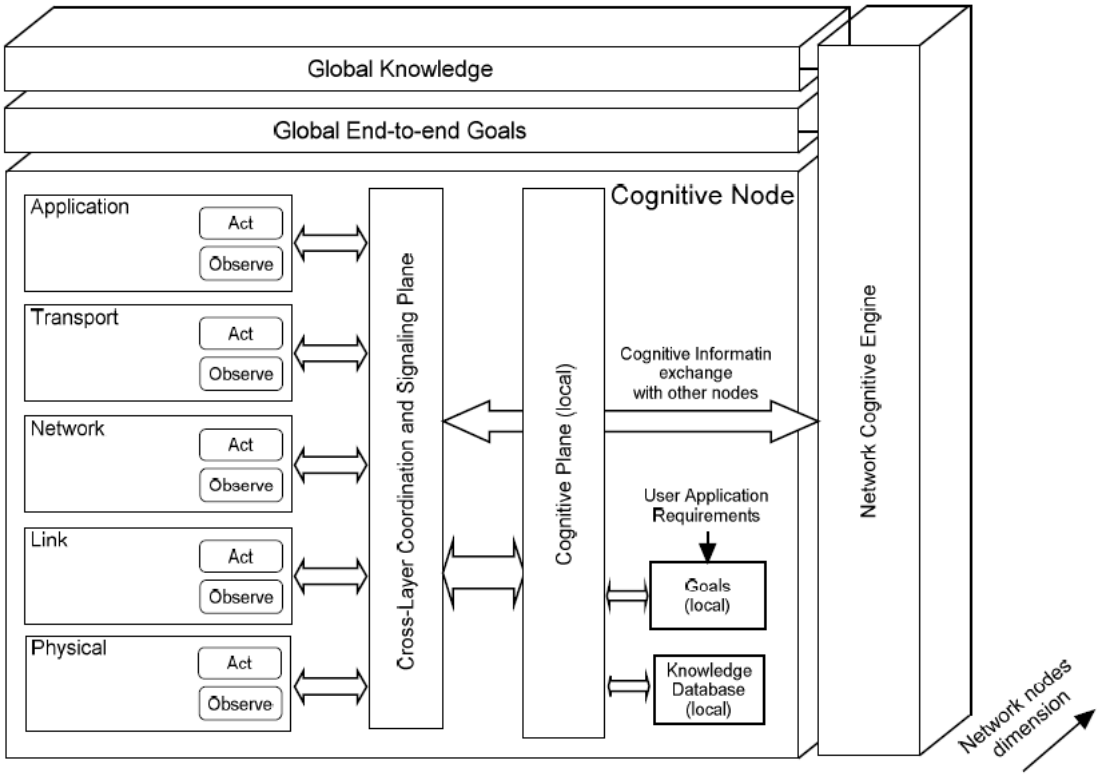



Figura 1.2: Architettura di rifermento per un framework cognitivo

estrapolate informazioni relative ad ogni layer, e allo stesso tempo si possono modificare i parametri funzionali.

L'informazione raccolta nei diversi layer del protocollo, viene poi inviata al modulo cognitivo (cognitive plan), implementato su ogni nodo. Tale processo esegue un'analisi dei dati e prende decisioni sulle operazioni da intraprendere. Inoltre, i risultati dell'analisi dei dati possono essere immagazzinati in un database locale denominato *local Knowledge database*. Gli obiettivi, definiti dalle applicazioni di rete in termini di QoS, sono dati in input al processo cognitivo locale e hanno valenza solamente locale.

Il livello cognitivo comunica con gli agenti posti nei vari layer dello stack TCP/IP tramite il livello *CCSP* (Cross-layer Coordination and Signaling Plane). Questo modulo si occupa di gestire i segnali da e verso gli agenti osservatori/attuatori, gestendo in modo diverso i messaggi diretti a layer diversi, come le temporizzazioni necessarie alle comunicazione.

Quanto appena presentato, é ció che avviene a livello di singolo nodo. La rete cognitiva, composta di piú nodi, viene gestita da un *NCE* (Network Congnitive Engine). Questo elemento é in grado di comunicare con tutti i

moduli cognitivi all'interno dei singoli nodi, quindi di coordinarli e gestirli. Il suo compito consiste nel raccogliere i dati provenienti dai vari nodi, compresi i vari obiettivi locali e le esigenze delle applicazioni. Tali richieste posso essere eseguite tramite uno scheduler che interroga periodicamente i vari agenti distribuiti nei nodi, oppure in modalitá on-demand, ovvero tramite richieste asincrone.

Questo approccio garantisce una buona scalabilitá del sistema in quanto combina un'architettura centralizzata (a livello di nodo) con un'architettura distribuita ( a livello di rete). Inoltre, a livello di singolo nodi, c'è una netta distinzione tra la parte che si occupa dell'analisi, dell'apprendimento e delle decisioni, da quella che si occupa della raccolta dati e dell'azione. La prima difatti é concentrata nel modulo cognitivo e in questo risiede tutta l'intelligenza del nodo. Viceversa, osservatori e attuatori, residenti su ogni layer dello stack TCP/IP sono solo strumenti. Pertanto, il protocollo di comunicazione non viene appesantito in nessun modo, mentre il carico elaborativo viene demandato al livello cognitivo.

## 1.5 Processo cognitivo

Il processo cognitivo é la parte piú importante per una rete cognitiva. Di fatto si occupa della gestione degli agenti, della raccolta e dell'analisi dati e prende le decisioni che, in base a tecniche di apprendimento, ritiene piú appropriate in risposta ai comportamenti rilevati e alle politiche definite. Il modo di operare del processo cognitivo dipende da come é stato implementato (approccio centralizzato o distribuito) e dalla quantitá di informazioni di cui dispone.

### 1.5.1 Contesto operativo

La rete cognitiva, opera al di sopra degli obiettivi dei singoli nodi, i quali si preoccupano di ottimizzare le performance dei propri flussi dati. Per ogni nodo passano svariati flussi dati, pertanto la rete cognitiva deve essere in grado di assegnare delle prioritá a tali flussi; questo viene fatto mantenendo un insieme di obiettivi end-to-end. I processi cognitivi nei singoli nodi, sono in generale autorizzati ad agire in funzione dei proprio obiettivi. In altri casi, invece, le politiche dei singoli nodi non devono andare in conflitto con le politiche definite globalmente.

La differenza di performance, tra l'operare secondo gli obiettivi locali, piuttosto che quelli globali, viene definito *the price of anarchy* in [6]. Con tale termine, viene definita la differenza in termini di performance, tra una

rete completamente gestita da un'unitá principale che ha piena conoscenza di tutta la rete e una rete autogestita, dove ogni nodo prende decisioni in modo autonomo e quindi anarchico.

La risposta al quesito di quale sia la migliore tra le due soluzioni, definisce le linee da seguire nell'implementazione di una rete cognitiva, in quanto questa deve trovare il giusto trade-off tra una conoscenza omniscente dello stato della rete e l'auto-gestione da parte dei singoli dispositivi. In [6], viene fatto notare come una rete dove ogni nodo si autogestisce porti a performance decisamente inferiori a quelle di una rete, ove regna un obiettivo comune tra tutti i nodi.

### 1.5.2 Modalitá operativa

L'efficacia delle decisioni prese da una rete cognitiva, sulle prestazioni di una rete, dipende fortemente dalla quantiá di informazioni disponibili al processo cognitivo. Per prendere decisioni congrue agli obiettivi end-to-end, una rete cognitiva ha bisogno di conoscere lo stato attuale della rete e gli obiettivi di ogni nodo. Quando la rete in questione é molto vasta, questo non é sempre possibile, in quanto lo scambio d informazioni porterebbe a costi troppo onerosi. Pertanto il processo cognitivo dovrá operare con una visione piú ristretta rispetto allo stato globale della rete. I possibili modi di operare sono due:

- **Approccio centralizzato:** un nodo centrale tiene traccia dell'intero stato della rete e degli obiettivi globali

- **Approccio distribuito:** i nodi della rete si scambiano le informazioni e non esiste alcuna autoritá centrale che gestisca la raccolta e la disseminazione delle informazioni.

Entrambi gli approcci hanno pro e contro: dal problema del *single point of failure* a quello dell'eccessivo *overhead* nel caso distribuito ... Come é stato presentato in 1.4, puó anche essere applicato un approccio ibrido, dove ad esempio la distribuzione degli obiettivi avviene in maniera distribuita tra i nodi della rete, mentre un nodo funge da *repository* per lo stato globale della rete.

### 1.5.3 Feedback

Un aspetto comune a qualunque modello cognitivo, é il cosiddetto *feedback loop*.

Il termine *feedback* descrive la situazione in cui le informazioni o i risultati provenienti da un evento passato, influenzeranno lo stesso evento nel presente o nel futuro. Quando tale evento é parte di una catena di eventi legati tra loro da una relazione di tipo causa-effetto a formare un ciclo, allora si dice che tale evento porta del feedback a se stesso.

Nelle reti cognitive, il ciclo di feedback puó essere schematizzato come in Figura 1.3 , ovvero composto di quattro processi: osservazione, analisi, decisione e azione.(vedi sezione 1.3 a pag. 4).

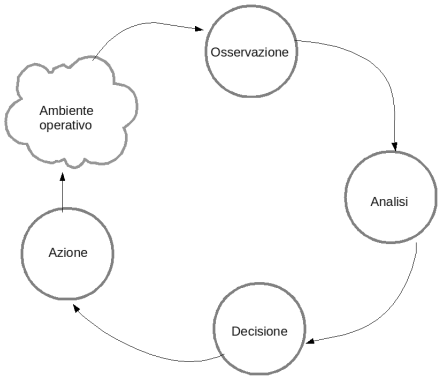



Figura 1.3: feedback loop

Il ciclo mostra la sequenza di operazioni da seguire per prendere decisioni in seguito a input provenienti dall'ambiente operativo. Questo ciclo deficie comunque di qualche componente importante, come ad esempio la mancanza di un obiettivo generale da inserire in feedback ai processi di analisi e decisione. Un altro componente mancante é il modulo dell'apprendimento, il quale serve a prevenire errori provenienti dalle iterazioni passate, sulle iterazioni future.

I feedback loop di questo tipo funzionano lo stesso (cioè anche in mancanza di queste componenti) poiché l'ambiente operativo, per quanto complesso sia, non é totalmente casuale. La struttura complessa di tali ambienti, non puó essere studiata attraverso analisi teorica, mentre lo si puó fare attraverso l'applicazione ripetitiva di iterazioni tipo test/response da parte del ciclo di feedback.

Quanto appena descritto é l'idea che porta all'implementazione di un framework cognitivo come quello descritto in 1.4, o come quello presentato da Thomas e DaSilva in [7].

Quello che manca alla nostra descrizione, é la vera e propria intelligenza della rete, ovvero la cognizione.

Gli algoritmi cognitivi, fanno parte della famiglia dei “machine learning algorithms”, materia studiata sin dai primi anni ’60. La definizione di machine learning é:

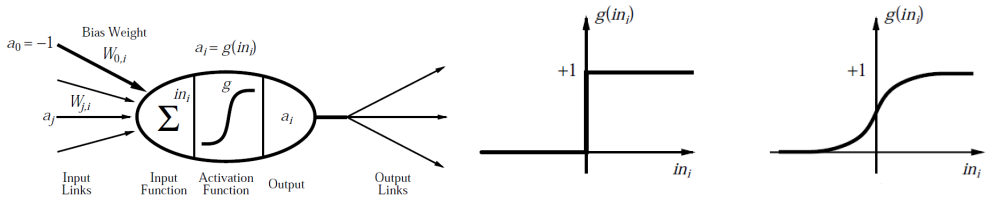
> *ogni algoritmo che migliora le proprie performance attraverso l’esperienza acquisita durante un periodo, senza avere completa conoscenza dell’ambiente in cui opera.*

Seguendo questa definizione, possono esser sviluppati un gran numero di algoritmi di intelligenza artificiale, che verranno ora brevemente enunciati. La scelta di quale algoritmo utilizzare, dipende principalmente dall’obiettivo primario che la rete cognitiva si pone e da come questo viene definito. In alcuni casi, dove la rete é particolarmente complessa, possono essere in esecuzione piú processi cognitivi, basati su diversi algoritmi, ognuno di questi atto a realizzare un dato obiettivo.

### 1.5.4 Reti neurali

Le neuroscienze hanno permesso di stabilire che la struttura cerebrale è caratterizzata dalla presenza di cellule neuronali con comportamenti vari e, soprattutto, da pattern di interconnessioni neuronali diversi a seconda del compito cognitivo. Per i modelli artificiali è stata seguita una metafora simile: sono stati studiati diversi tipi di neuroni e diverse architetture associandovi le modalitá di elaborazione concepite per implementare un determinato compito cognitivo.

In Figura 1.4 é rappresentato il modello matematico di un neurone, definito nel 1943 da McCulloch e Pitts. La funzione di attivazione deve rispondere




(a) Modello matematico di un neurone (b) Funzione di Attivazione

Figura 1.4: Modello matematico di un neurone e funzioni di attivazione

a due requisiti: primo, si vuole che il neurone si attivi solo per l’input corretto e sia “inattivo’ quando viene stimolato da un input errato. Secondo, la funzione di attivazione deve essere non lineare, altrimenti l’intera rete neurale risulterebbe essere una semplice funzione lineare.

Come mostrato in Figura 1.4b sono possibili due tipi di funzione di attivazione: la *threshold function* e la *sigmoid function*. Il vantaggio di usare la funzione sigmoide, è che questa é differenziabile.

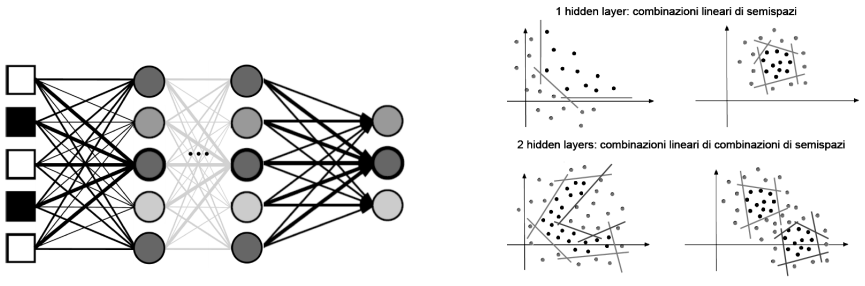
### Struttura della rete neurale

Le due principali strutture di una rete neurale sono quella aciclica o **feed-forward** e quella **ciclica** o ricorrente. Una rete aciclica rappresenta una funzione del suo input corrente, pertanto non ha stati interni,

Una di tipo ciclico, invece usa il suo output come feedback, con l'effetto che i livelli di attivazione della rete formano un sistema dinamico che puó raggiungere uno stato stazionario oppure oscillare. Inoltre, la risposta della rete all'input, dipende anche dallo stato iniziale del sistema il quale a sua volta dipende dagli input precedenti, ovvero supporta una sorta di memoria a breve termine.

### Funzionamento

Solitamente, vengono implementate reti di tipo feed-forward. In queste, ogni neurone svolge il compito di un separatore lineare. Pertanto, se una rete vuole classificare n categorie, con dati in input provenienti da un dataset linearmente separabile, sarà necessario costruire una rete neurale con un neurone in uscita per ogni classe. Nel caso i dati non siano separabili, si deve ricorrere a reti neurali multistrato. Ad esempio, se si usa un solo *hidden layer*, si ottengono delle separazioni lineari di semispazio. Se ne vengono usati due, si ottengono combinazioni lineari di combinazioni di semispazi. Vedi Figura 1.5.




(a) Rete neurale multistrato (b) 1 e 2 hidden layers

Figura 1.5: Rete neurale multistrato

Le reti neurali artificiali sono di scarso interesse senza il paradigma centrale dell'apprendimento, che viene ispirato al corrispondente paradigma neu-

robiologico. Apprendere in una rete neurale artificiale corrisponde a modificare il valore dei pesi delle connessioni sinaptiche. Tale processo è influenzato dagli esempi che concorrono a sviluppare concetti. I dati e l'interazione con l'ambiente concorrono con diversi protocolli allo sviluppo di competenze cognitive. In particolare, si individuano tre diverse modalitá di apprendimento:

- l'apprendimento con supervisione,
- l'apprendimento con rinforzo,
- l'apprendimento senza supervisione.

Nell'apprendimento con supervisione e con rinforzo, la rete neurale deve sviluppare un concetto sulla base alle interazioni con un supervisore, che provvede a istruire la rete, fornendo informazioni sul concetto. Si parla di rinforzo, quando l'informazione fornita dall'esterno è parziale, mentre di supervisione, quando il supervisore fornisce l'informazione completa sul concetto

In generale, i tre protocolli di apprendimento descritti, sono formulabili come ottimizzazione di una funzione dei pesi della rete neurale. Nel caso dell'apprendimento con rinforzo e dell'apprendimento con supervisione, per rendere il comportamento della rete neurale conforme alla supervisione, occorre minimizzare una funzione di errore che dipende dalla scelta dei pesi e misura l'errore rispetto alle informazioni del supervisore. Nel caso dell'apprendimento senza supervisione, l'auto organizzazione per similaritá dei dati puó ancora, generalmente, formularsi come l'ottimizzazione di una funzione di armonia. Il problema di ottimizzare funzioni in grossi spazi, é generalmente difficile per la potenziale presenza di minimi locali, che puó rendere inefficaci le classiche euristiche di ottimizzazione basate sulla tecnica di massima discesa del gradiente.

### 1.5.5 Algoritmi genetici

Gli Algoritmi Genetici (AG), proposti nel 1975 da J.H. Holland, sono un modello computazionale idealizzato dall'evoluzione naturale darwinista.

Ogni individuo ha sue caratteristiche e proprietà specifiche, manifestate esternamente e "visibili", che ne costituiscono il fenotipo. É il fenotipo a dettare le possibilità e i limiti delle interazioni dell'individuo con l'ambiente in cui vive. I due principi fondamentali dell'evoluzione sono la variazione genetica e la selezione naturale. Affinchè la popolazione possa evolvere, gli individui che la costituiscono devono anzitutto avere una ricca varietà di

fenotipi. Può allora scattare la selezione, che premia la sopravvivenza, la longevità e la riproduzione degli individui piú adatti.

I meccanismi generatori della varietà del fenotipo sono sostanzialmente due: un processo combinatorio dei geni e le mutazioni geniche casuali. Le mutazioni producono nuovi geni, alcuni dei quali si tramandano alle generazioni successive, mentre altri scompaiono e il cosiddetto pool di geni, nel quale "pesca" la selezione naturale, cambia continuamente.

Solitamente, i GA sono usati per risolvere problemi di ricerca e ottimizzazione. Considerano una popolazione di cromosomi (individui) che rappresentano soluzioni possibili per un certo problema, la qualitá di un individuo (cioè quanto è buona la soluzione per il problema) viene misurata mediante una funzione di fitness. In un certo senso, la funzione di fitness indica l'adattabilità all'ambiente: gli individui che meglio si adattano (fit) hanno più probabilità di riprodursi e di trasmettere i propri geni alle generazioni future. Pertanto un GA è una procedura di ricerca iterativa il cui scopo é l'ottimizzazione della funzione di fitness. Partendo da una popolazione iniziale, un GA produce nuove generazioni che contengono solitamente individui migliori delle precedenti: l'algoritmo evolve verso l'ottimo globale della funzione di fitness. In realtà, non é garantito che un GA trovi una soluzione ottima globale, ma si può dire che un GA è in grado di trovare soluzioni buone in tempi ragionevoli.

### Funzionamento

Nel modello tradizionale, i cromosomi sono stringhe di bit di lunghezza fissa e tutte le generazioni hanno la stessa dimensione (numero di individui).Ogni cromosoma rappresenta un punto nello spazio di ricerca. Gli operatori di ricerca più importanti sono la ricombinazione (o crossover) e la mutazione.

Il **crossover** combina i geni, tipicamente di due individui, per produrre individui figli che ereditano caratteristiche da entrambi i genitori.

La **mutazione** reintroduce nella popolazione materiale genetico perduto.

La ricerca genetica realizza un compromesso tra "exploitation" della soluzione disponibile migliore ed "exploration" dello spazio di ricerca.

Exploitation ed exploration corrispondono, rispettivamente, a ricerca locale e ricerca globale: exploitation eccessiva puó portare l'algoritmo a convergere ad una soluzione non accettabile (la ricerca resta intrappolata in un ottimo locale), exploration eccessiva puó non sfruttare appropriatamente la conoscenza già disponibile rendendo il processo di ricerca molto lento (un esempio è la ricerca casuale).

Per un problema di ottimizzazione, la funzione di fitness puó coincidere con la funzione obiettivo (o una sua trasformazione). I GA sono procedure di

massimizzazione, quindi valori di fitness piú alti sono associati ad individui migliori (problemi di minimizzazione vengono di solito riformulati). A volte la fitness di un cromosoma é misurata in maniera implicita, valutando la qualitá della corrispondente soluzione rispetto al problema.

Una nuova generazione, è ottenuta a partire dalle precedenti, secondo i seguenti passi:

1. **valutazione:** si valuta ogni individuo mediante la funzione di fitness
2. **selezione per riproduzione:** gli individui migliori sono selezionati per la riproduzione.
3. **crossover:** si applica l'operatore di crossover agli individui del pool generato dalla fase precedente.
4. **mutazione:** l'operatore di mutazione é applicato con una certa probabilitá (probabilitá di mutazione) agli individui a cui è stato applicato il crossover.
5. **selezione per rimpiazzamento e sopravvivenza:** la nuova generazione contiene gli individui della popolazione (appena creata dalle operazioni precedenti) ma puó includere anche altri individui.

### 1.5.6 Expert Systems

Un sistema esperto (SE) é un programma che tenta di riprodurre il comportamento di un esperto umano in uno specifico dominio. Un SE si fonda sulla competenza umana registrata nella cosiddetta base di conoscenza (ad esempio sotto forma di regole), aggiornabile in base all'esperienza. Come avviene per l'esperto umano, il sistema esperto puó operare su dati qualitativi e incompleti. Puó infatti utilizzare forme di ragionamento approssimato, attraverso tecniche probabilistiche o facendo ricorso alla cosiddetta "fuzzy logic", un tipo di logica a piú valori, spezzo detta logica sfumata, In particolare, un SE:

- fornisce le stesse risposte o gli stessi consigli che fornirebbe l'esperto umano;
- é in grado di giustificare la propria risposta.

I SE si usano per risolvere problemi la cui soluzione richiede una considerevole esperienza umana. I tipici campi di utilizzo sono la diagnostica, la progettazione, il controllo, la pianificazione, l'istruzione, etc. In Figura 1.6 viene mostrata la tipica architettura di un SE.

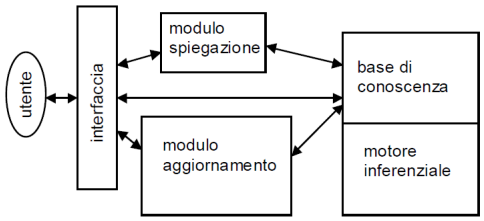



Figura 1.6: expert systems

Il compito dell'*interfaccia* é quello di rendere la comunicazione tra l'utente ed il SE la piú naturale possibile.

La *base di conoscenza* memorizza la conoscenza dell'esperto. Tale conoscenza consiste, essenzialmente, di fatti e regole.

Il *motore inferenziale* é la parte attiva del sistemae usa la base di conoscenza per inferire nuovi fatti e produrre soluzioni.

I sistemi esperti si dividono in due categorie principali:

- Sistemi esperti basati su regole
- Sistemi esperti basati su alberi

I primi sono dei programmi composti da regole della forma "IF condizione THEN azione". Data una serie di fatti, i SE grazie alle regole di cui sono composti, riescono a dedurre nuovi fatti.

Un sistema esperto basato su alberi, dato un insieme di dati ed alcune deduzioni, crea un albero che classifica i vari dati. Nuovi dati verrebbero analizzati dall'albero e il nodo di arrivo rappresenterebbe la deduzione.

### 1.5.7 Logica Fuzzy

La teoria Fuzzy nasce nella metá degli anni 60, quando il professore Lotfi A. Zadeh espone la necessitá di utilizzare concetti imprecisi nella risoluzione di problemi della vita reale. In un articolo del 1965, Zadeh pubblicó il primo articolo che introduceva il concetto di insiemi fuzzy, i quali, a differenza degli insiemi tradizionali, inglobano il concetto di grado di appartenenza di un elemento all'insieme. Con questo concetto si é resa possibile la rappresentazione di concetti imprecisi che tutti noi utilizziamo normalmente nel linguaggio naturale.

Nella logica classica, un predicato (una frase) può essere o vera o falsa. Non esistono vie di mezzo. Un uomo è alto oppure no, un oggetto è rosso oppure no e così via. In logica fuzzy, invece, non si specifica che un uomo

misura 1.70 m di altezza, bensì che è alto o basso o, ancora, che sia abbastanza alto.

### Insiemi fuzzy

Un insieme fuzzy é caratterizzato dal fatto che il grado di appartenenza di ogni elemento all'insieme puó essere un qualunque numero reale tra 0 e 1. Un insieme fuzzy A é definito quindi da una funzione di appartenenza $\mu_A : X \rightarrow [0,1]$, essendo X l'universo di definizione. L'universo X é un insieme convenzionale (o *crisp*). Per gli insiemi fuzzy, sono definite le operazioni di complemento, unione ed intersezione:

$$\overline{A}(u) = 1 - A(u) \tag{1.1}$$

$$(A \cup B)(u) = min(A(u), B(u)) \tag{1.2}$$

$$(A \cap B)(u) = max(A(u), B(u)) \tag{1.3}$$

L'intersezione fuzzy standard (operatore di minimo) produce l'insieme fuzzy più grande fra quelli prodotti da tutte le possibili intersezioni fuzzy (maggiore incertezza).

L'unione fuzzy standard (operatore di massimo) produce l'insieme fuzzy più piccolo fra quelli prodotti da tutte le possibili unioni fuzzy (minore incertezza).

Una delle applicazioni di maggiore successo della logica fuzzy è lo sviluppo di sistemi esperti e sistemi di controllo in cui la conoscenza è espressa mediante regole fuzzy. Il controllo fuzzy (fuzzy control) è l'applicazione della logica fuzzy al controllo, cioè al comportamento di un agente che riceve certi input. Le entità che hanno a che fare con un controllo sono generalmente gli ingressi (sensori) e le azioni che possono essere intraprese in risposta a tali ingressi. Nel controllo fuzzy gli ingressi sono input fuzzy, ovvero non precisi. Dunque non avremo la temperatura è $35°C$, bensì "fa caldo", "fa molto caldo" ... Tali input vengono processati usando le regole fuzzy (fuzzy rules), che possono essere descritte mediante un linguaggio naturale, ad esempio: SE fa caldo, ALLORA abbassa la temperatura. Dopo che gli input sono stati processati, il risultato viene defuzzicato, cioè reso di nuovo un valore preciso) e l'output viene utilizzato per il controllo (ad esempio, abbassare la temperatura).

## 1.6 Radio cognitive vs reti cognitive

Ci sono spesso dei fraintendimenti circa l'ambito delle rete cognitive e quello delle radio cognitive.

L'elemento che accomuna radio e reti cognitive, é la definizione di un processo cognitivo, basato sull'osservazione e la misurazione per ottimizzare la riconfigurazione dei parametri operativi.

Le radio cognitive forniscono un metodo d'accesso dinamico ed efficiente allo spettro del segnale radio, cambiando i propri parametri di trasmissione e ricezione in modo da evitare le interferenze con gli altri dispositivi di comunicazione.

Le principali funzioni di una radio cognitiva sono:

- **Sensing dello spettro radio** al fine di rilevare gli utilizzatori primari e le porzioni libere dello spettro.
- **Gestione dello spettro radio** per scegliere la frequenza piú adatta alla trasmissione tra quelle disponibili.
- **Condivisione dello spettro radio** con gli altri dispositivi cognitivi al fine di utilizzare in modo equo le risorse.
- **Mobilitá** in modo da liberare lo spettro qualora un utente primario venga identificato.

Le radio cognitive hanno lo scopo di ottimizzare l'accesso al canale al fine di massimizzare l'utilizzo dello spettro e questo viene fatto sia al livello fisico (spectrum sensing) sia al livello link (coordinamento e e scheduling delle trasmissioni).

Invece, le reti cognitivi, operano su uno scenario piú vasto e complesso, composto di diverse tecnologie (wireless, linee ottiche, ethernet...) e di diversi apparati (routers e nodi) che non sempre sono stai progettati per cooperare tra loro.

In definitiva, la differenza principale sta nell'ambito dell'ottimizzazione: le radio perseguono obiettivi di ottimizzazione locale, mentre le reti cognitive, vogliono ottimizzare le performance end-to-end.

## 1.7 Radio cognitive

Il concetto di radio cognitiva, presentato inizialmente da Mitola [4, 8] e in accordo con la Federal Communications Commission, definisce un sistema radio che esegue in modo continuo operazioni si sensing sullo spettro del canale,

identifica in maniera dinamica i canali liberi e va quindi ad operare su quelle porzioni dello spettro in cui gli altri sistemi radio non stanno trasmettendo. Un'altra definizione di radio cognitiva viene data in [9] :

> Una radio cognitiva é un sistema di comunicazione wireless, che utilizza in modo intelligente ogni informazione disponibile circa:
>
> - attivitá
> - condizione dei canali
> - informazioni provenienti da altri nodi
>
> attraverso le quali condividere lo spettro.

In altre parole, lo scopo principale di una radio cognitiva, é quello di determinare quali porzioni dello spettro (spectrum hole), non sono utilizzate in un determinato istante temporale. Seguendo alla lettera questa definizione, tuttavia, non viene utilizzato al meglio lo spettro disponibile; una soluzione piú efficiente prevede di consentire la trasmissione da parte di una radio cognitiva su un dato canale, fintanto che tale trasmissione non crea un disturbo alla radio primaria (ovvero quella alla quale era giá stato assegnato il canale) superiore ad una determinata soglia.

In [10] viene definita una nuova metrica per stimare l'interferenza sul canale: la *temperature interference*. Con il termine **temperature interference**, si definisce il massimo livello di interferenza accettabile al ricevitore, per la frequenza di banda interessata. Quindi, una radio cognitiva, puó coesistere su un canale con un'altra radio (non cognitiva o legacy), a patto che l'*interference temperature* al ricevitore della radio legacy, non ecceda la massima *temperature interference* consentita.

### 1.7.1 Software Defined Radio (SDR)

Quando si parla di radio cognitive, ci si riferisce in modo implicito a dispositivi SDR che svolgono qualche funzionalitá aggiuntiva... Un transceiver le cui funzioni di comunicazione sono realizzate da un software in esecuzione su un processore ad-hoc, viene definito **Software radio**. Sfruttando il medesimo hardware, le funzioni di ricezione e trasmissione possono essere realizzati da diversi algoritmi, ognuno dei quali puó implementare un determinato standard. Una software radio, implementa tutti i layers di un sistema di comunicazione, anche se solitamente, ai fini delle radio cognitive, vengono maggiormente caratterizzati i livelli fisico e link dello stack TCP/IP.

Una radio cognitiva, altro non é che una SDR che esegue sensing dell'ambiente operativo, tiene traccia dei cambiamenti e reagisce una volta tratte le

conclusioni. Puó quindi esser vista come un'unitá autonoma in un ambiente di comunicazione, che scambia con una certa frequenza informazioni circa il suo stato con il resto della rete. In [11], viene data una suddivisone delle Software Defined Radio, in base al loro ambiente operativo:

1. Sistema multi banda: qualora la SDR riesca a coprire diverse frequenze di funzionamento (e.g., la rete GSM 900, GSM 1800, GSM 1900).

2. Sistema multi standard: se la SDR riesce ad operare con diversi standard (e.g., ULTRA-FDD e ULTRA-TDD per l'UMTS) oppure su reti diverse (e.g., GSM, UMTS, WLAN).

3. Sistema multi servizio: quando la SDR fornisce diversi servizi (e.g., telefonia, trasporto dati, streaming video ...)

4. Sistema multicanale: se supporta la trasmissione e la ricezione su piú canali in contemporanea.

### 1.7.2 Radio cognitive user & technology centric

Mitola e Maguire [8], nel descrivere la radio cognitiva, si concentrano sopratutto nel linguaggio da utilizzare per la rappresentazione della conoscenza, (RKRL - *Radio Knowledge Representation Language*).

In [11], invece viene dato un focus diverso nel descrivere tali dispositivi e in particolare, viene fatta una distinzione tra le funzioni che una radio cognitiva puó svolgere. Le proprietá di una radio cognitiva, possono essere **user centric** oppure **technology centric**.

Al primo gruppo afferiscono tutte quelle funzionalitá "utili" all'utente, come la capacitá da parte della radio di rispondere a query del tipo: "Che indirizzo ha quel ristorante?", oppure di fornire informazioni circa lo stato del traffico lungo un determinato percorso.

Della categoria *technology centric*, appartengono invece tutte quelle funzionalitá utili alla rete, come ad esempio la localizzazione, il tracking, monitoraggio dello spettro... Le funzionalitá *user centric* possono facilmente essere implementate attraverso delle query verso dei database ed é un tipo di intelligenza che puó essere attivata nella rete "su chiamata".

In un sistema dove l'accesso al canale viene condiviso da piú apparati, per ottene una gestione ed un utilizzo efficiente dell'intero spettro, una radio cognitiva deve applicare degli algoritmi ben piú complessi che nel caso *user centric*. É possibile immaginare che l'algoritmo che gestisce l'accesso al mezzo trasmissivo, partendo da una richiesta da parte dell'utente (o di

un'applicazione), in primo luogo scegla il rate di trasmissione dei dati, successivamente la modalitá di trasmissione e infine calcoli la banda richiesta per la trasmissione. Fatto questo, la radio cognitiva deve trovare la risorsa piú appropriata per effettuare la trasmissione. Dovrá quindi conoscere la propria posizione e quella della base station di destinazione, cosa é in grado di fare, ovvero di quali tecnologie dispone e le rispettive caratteristiche. Al fine di stabilire quali siano le possibili interferenze, puó ad esempio rilevare la presenza di segnale nelle frequenze adiacenti la propria e/o lo standard trasmissivo utilizzato da tali frequenze. Quindi, una radio cognitiva dovrebbe implementare le seguenti funzionalitá:

1. sensori di posizione (e.g., GPS)
2. possibilitá di fare sensing dello spettro
3. algoritmi di apprendimento e ragionamento
4. capacitá di ascoltare prima di trasmettere, in modo da evitare disturbi da parte di *hidden stations*
5. algoritmi per garantire l'equitá nelle comunicazioni (*fair*)
6. condividere le informazioni tramite un'interfaccia

## 1.8 Modelli di riferimento

La funzione principale per cui viene progettata una radio cognitiva, é il sensing dello spettro. In letteratura, sono stati definiti tre paradigmi, i quali definiscono i modi di operare delle radio, in seguito all'operazione di sensing.

### 1.8.1 Paradigma underlay

Questo paradigma consente agli utenti della radio cognitiva, di usare il canale se l'interferenza causata dai dispositivi non cognitivi é al di sotto di una data soglia.

Piú in dettaglio, il paradigma *underlay* parte dal presupposto che la radio cognitiva sia a conoscenza dell'interferenza causata dalle proprie trasmissioni nei riguardi dei ricevitori delle radio non cognitive. La radio cognitiva, in questo caso svolge il ruolo di "utente secondario", in quanto cerca di non interferire con le comunicazioni degli "utenti primari", ovvero delle radio legacy. Come giá accennato, i dispositivi cognitivi sono abilitati alla trasmissione, solo le l'interferenza rilevata dai dispositivi non cognitivi é inferiore ad

una soglia prefissata. Per soddisfare tale requisito, si possono usare diverse tecniche, come l'uso di antenne multiple per tenere il segnale "cognitivo" lontano da quello dei dispositivi legacy.

Un'altra tecnica consiste nell'usare un segnale a banda larga (spread spectrum), per cui viene migliorato il rapporto segnale/rumore e viene quindi permessa la trasmissione simultanea di piú utenti.

L'utilizzo combinato di tecniche di spread spectrum e ultra-wide-band [1] (UWB), rappresenta un'altra tecnica per soddisfare il paradigma *underlay*. L'interferenza causata dal trasmettitore cognitivo, puó essere approssimata sfruttando la reciprocitá, a patto che la radio cognitiva riesca a rilevare le trasmissioni nei pressi dei dispositivi non cognitivi coinvolti. In alternativa, la radio cognitiva, puó adottare una strategia conservativa, per cui limita la propria potenza trasmissiva cosí da assicurare che il segnale rimanga al di sotto della soglia.

Il paradigma underlay, a causa delle restrizioni che impone, consente agli utenti di effettuare trasmissioni solo di breve durata.

### 1.8.2 Paradigma overlay

Il paradigma *overlay* prevede che la radio cognitiva, attraverso sofisticati algoritmi di codifica e signal processing, migliori la comunicazione delle radio non cognitive, ottenendo comunque della banda aggiuntiva per le proprie comunicazioni.

Il presupposto base di questo paradigma é che la radio cognitiva sia in possesso delle informazioni necessarie per decodificare le trasmissioni dei dispositivi legacy. Nel caso le radio non cognitive usino dei codici pubblici (secondo uno standard), tali informazioni possono essere facilmente ottenute. In questo modo, le radio cognitive possono ottenere i messaggi degli utenti, semplicemente decodificando le trasmissioni. Per semplicitá, il paradigma prevede che anche i messaggi (non cognitivi) siano conosciuti al ricevitore cognitivo non appena comincia la trasmissione. Questa assunzione, risulta impraticabile per la prima trasmissione, tuttavia risulta valida nel caso di ritrasmissione: nel caso la prima trasmissione venga correttamente ricevuta dalla radio cognitiva (e quindi riesce a decodificare il messaggio) mentre il vero destinatario (dispositivo non cognitivo) non riesce a decodificare il messaggio a causa di interferenze o fading.

[1] Con il termine ultra wideband (UWB) si indica una tecnologia sviluppata per trasmettere e ricevere segnali mediante l'utilizzo di impulsi di energia di durata estremamente ridotta (da poche decine di picosecondi a qualche nanosecondo). La brevitá dell'impulso rende l'UWB poco sensibile alle interferenze dovute alla riflessione dell'onda stessa.

La conoscenza dei messaggi e/o dei simboli di codifica usati dai dispositivi non cognitivi puó essere sfruttata in diversi modi, per mitigare o eliminare l'interferenza rilevata dai ricevitori cognitivi e non. Per fare questo si possono usare tecniche come il *dirty paper coding* [2] (DPC).

Inoltre il dispositivo cognitivo puó utilizzare la conoscenza acquisita per usare parte della sua potenza trasmissiva per le proprie comunicazioni e un'altra parte per assistere le comunicazioni non cognitive, fungendo da nodo di relay.

Scegliendo in modo opportuno il partizionamento della potenza, si puó far si che l'incremento del rapporto segnale-rumore del segnale non cognitivo, sia esattamente l'offset necessario per annullare il disturbo dovuto dalle trasmissioni cognitive. Questo garantirebbe agli utenti non cognitivi che il loro rate trasmissivo resti invariato, anche in presenza di trasmissioni cognitive.

### 1.8.3 Paradigma interwave

Il paradigma *interwave*, prevede che la radio cognitiva analizzi lo spettro e trovi i "buchi" spettrali nei quali trasmettere, senza gravare sulle comunicazioni giá esistenti.

Questo paradigma aderisce appieno alla motivazione iniziale che ha portato allo studio delle radio cognitive, ovvero al concetto di comunicazione opportunistica. L'idea viene appunto dallo studio dell'occupazione dello spettro, dove si é visto che una grande porzione di questo non viene utilizzata per la maggior parte del tempo. I dispositivi cognitivi, possono quindi approfittare di questi *spectrum holes* per effettuare le proprie comunicazioni e migliorare cosi l'occupazione dello spettro.

La tecnica *interwave* richiede una conoscenza delle attivitá trasmissive dei dispositivi legacy, e questo puó essere ottenuta monitorando periodicamente lo spettro radio e identificando le porzioni occupate. Successivamente la comunicazione potrá avvenire nelle frequenze "libere", producendo un'interferenza minimale verso gli altri utenti.

Mentre le tecniche *overlay* e *underlay* permettono ai dispositivi cognitivi e a quelli legacy di comunicare in maniera concorrente, lo scopo principale delle tecniche *interwave* é di evitare le comunicazioni simultanee.

Inoltre l'approccio cognitivo, richiede diversi quantitativi di informazioni preliminari: i sistemi underlay necessitano di conoscere l'interferenza da loro

[2] Dirty paper coding (DPC) é una tecnica per la trasmissioni efficiente di dati digitali attraverso un canale di comunicazione che é soggetto a qualche interferenza nota al trasmittente. La tecnica consiste nell'operare una pre-codifica dei dati in modo da cancellare gli effetti dell'interferenza.

prodotta e rilevata dai dispositivi non cognitivi; i sistemi interwave hanno bisogno di conoscere l'attivitá radio dei dispositivi attivi (che puó essere ottenuta tramite sensing del canale radio). Infine, i sistemi overlay sono tra quelli che necessitano della quantitá di informazione maggiore, in quanto devono conoscere i simboli di codifica e possibilmente i messaggi dei dispositivi non cognitivi. Per un trattamento piú approfondito dei tre paradigmi presentati, ci si riferisca a [9].

## 1.9 Stato dell'arte delle reti cognitive

Presentiamo ora alcune iniziative di ricerca nell'ambito delle reti cognitive.

### 1.9.1 $E^2R$

$E^2R$ (End-to-End Reconfigurability).Il progetto si concentra sulle reti di nuova generazione (Next Generation Network) e sfrutta svariate reti, come quella cellulare, rete fissa e tecnologie wireless. L'obiettivo principale di questo progetto, é di integrare in modo completo lo stack TCP/IP con degli strumenti di riconfigurazione.

Tuttavia, questo rappresenta anche l'inconveniente principale: il supporto simultaneo alla riconfigurazione di tutti i layers, da parte di tutti i dispositivi coinvolti, é un problema non banale e ne limita la distribuzione.

### 1.9.2 m@ANGEL

La piattaforma m@ANGEL introduce un approccio speciale, per trattare i problemi di mobilitá in reti eterogenee, tramite il supporto della cognizione. Il processo cognitivo viene implementato nel punto d'accesso, ovvero tra la base station e l'utente mobile. Il sistema é composto di nodi, chiamati entitá m@ANGEL, ognuna delle quali si occupa del monitoraggio, dell'intermediazione delle risorse, della gestione degli obiettivi e della riconfigurazione dei dispositivi. Eínoltre prevista una forma di cooperazione tra le varie entitá m@ANGEL, ma questo avviene solo per i nodi spazialmente vicini e questo meccanismo non viene propagato fino al core della rete.

### 1.9.3 CogNet

CogNet(Cognitive Complete Knowledge Network).Il progetto propone una nuova architettura cognitiva, atta a mantenere un certo grado di astrazione dallo stack protocollare TCP/IP. Ogni layer del protocollo, viene esteso da

un modulo, chiamato *Intra-layer Cognition Module*, che non é altro che un agente software con funzioni di monitoraggio, controllo e coordinamento. Ogni modulo é interconnesso tramite il *Cognitive Bus*, il quale operare in parallelo allo stack protocollare. La proprietá distintiva di questa architettura, risiede nella distribuzione delle funzioni cognitive su tutti i layer. Inoltre un'architettura siffatta, permette una riduzione dell'overhead dei messaggi scambiati nella rete e una semplificazione del processo cognitivo.

Tuttavia le performance sembrano essere strettamente dipendenti da come viene implementata la traduzione degli obiettivi end-to-end in obiettivi e impostazioni locali di ogni singolo layer.

### 1.9.4 Thomas et Al.

Thomas et Al. [7, 12] propongono un modello basato su tre livelli. Il livello piú alto é responsabile di tradurre le richieste delle applicazioni e degli utenti in obiettivi interpretabili dal processo cognitivo. Nel livello intermedio, possono essere implementati diversi algoritmi cognitivi, con diversi gradi di distribuzione tra i nodi della rete. Il livello inferiore del modello, corrisponde alla *Software Adaptable Network* (SAN), che é composta dai nodi e dai sensori della rete. La comunicazione tra i nodi della SAN e il livello intermedio, ovvero con il processo cognitivo, avviene mediante delle API.

### 1.9.5 Gelenbe et Al.

Gelenbe et Al.[13] hanno proposto l'idea di una rete di pacchetti cognitivi, dove le capacità di routing e controllo del flusso, vengono spostate dai nodi della rete ai pacchetti. I "pacchetti cognitivi" si auto-instradano e apprendono come evitare i percorsi congestionati e quelli dove verrebbero scartati. Ogni pacchetto contiene una mappa e una porzione di codice che verrà eseguito ogni qual volta il pacchetto viene ricevuto da un router.

### 1.9.6 SPIN

SPIN (Software programmable Intelligent Network), presentata da Lake et Al.[14], mette insieme le idee base di diverse tecnologie, come IP, PSTN, reti cellulari etc. al fine di superare le principali limitazioni delle reti IP, come ad esempio l'impatto dei cicli di feedback su larga scala nelle performance di rete). L'architettura di SPIN è composta di tre livelli interconnessi tra loro:

- forwarding plane: offre funzionalità di monitoring e switching, comunicazione orientata alla connessione e connectionless . . . inoltre fornisce misurazioni dirette e indirette, quali jitter, packet loss, banda, e latenza.

- Control/management plane: Gestisce il *forwarding plane* in base alle misurazioni ottenute.

- Cognitive plane: costituisce l'intelligenza dell'intero sistema e offre funzionalitá di amministrazione.

## 1.10 Parametri comparativi

Vediamo ora alcuni parametri valutativi e successivamente una tabella comparativa delle soluzioni appena presentate, secondo tali parametri.

**Consistenza con il protocollo TCP/IP** Questo parametro valuta il grado modifiche che é necessario apportare al protocollo standard TCP/IP.

La maggior parte degli approcci non é "TCP-friendly", sopratutto a causa della dipendenza degli elementi riconfigurabili dello stack protocollare. Negli esempi riportati precedentemente, l'unico schema consistente con il protocollo TCP/IP é CogNet, che d'altro canto è stato progettato appositamente per mantenere un'astrazione dai livelli del protocollo. In CogNet, tale obiettivo é stato raggiunto, introducendo su ogni layer del protocollo un'interfaccia (interfaccia cognitiva) , che permette l'interazione tra gli elementi interni ai livelli e il motore cognitivo.

**Riconfigurabilitá** Indica il grado di riconfigurabilitá richiesto dal framework cognitivo.

Soluzioni come m@ANGEL necessitano di riconfigurabilitá al livello più basso del protocollo, mentre soluzioni come E2R o SPIN, richiedono che l'intero protocollo sia riconfigurabile.

**Processo cognitivo** Uno degli aspetti principali che si devono considerare nell'implementazione del processo cognitivo, é la sua architettura che può essere centralizzata o distribuita.

Uno schema centralizzato permette un miglior controllo, mentre lo scopo di un'architettura distribuita é di ridurre la complessitá ed accrescere la *fault tollerance*.

**Livello di supporto richiesto alla rete** Indica il livello di cooperazione richiesto agli elementi della rete (routers,switches), per lavorare in modo opportuno.

Approcci come E2R, m@ANGEL e SPIN richiedono un elevato supporto da parte della rete, mentre soluzioni come quelle proposte da Gelenbe et

Al.,Thomas et Al. oppure CogNet, necessitano di supporto più moderato. Va sottolineato che una rete cognitiva, tende a perseguire gli obiettivi end-to-end introducendo dell'intelligenza nel core della rete, piuttosto che nei nodi terminali.

### 1.10.1 Comparazione degli schemi proposti

Presentiamo ora una tabella comparativa degli schemi proposti in 1.9, secondo i parametri di riferimento sopra-indicati.

| **Architettura** | **Consistenza con TCP/IP** | **Riconfigurabilitá** | **Processo cognitivo** | **Livello di supporto richiesto** |
|---|---|---|---|---|
| $E^2R$ | NO | Intero stack protocollare | Centralizzato o parzialmente distribuito | Alto |
| m@ANGEL | NO | Livello più basso del protocollo | nell'accesso alla rete | Alto |
| Gelenbe et Al. | NO | Intero stack | - | moderato |
| CogNet | SI | moduli interni ai livelli | distribuito | moderato |
| SPIN | NO | separazione tra il modulo cognitivo e quelo di gestione | distribuito | Alto |
| Thomas et Al. | NO | elementi riconfigurabili | distribuito | moderato |

Tabella 1.1: Comparazione delle architetture

## 1.11 Stato dell'arte delle radio cognitive

La maggior parte degli studi relativi alle radio cognitivi, riguardano il **Dynamic Spectrum Access/Allocation** (DSA), oppure il livello PHY, e in ambo i casi l'hardware utilizzato svolge un ruolo cruciale in termini di riconfigurabilità. In linea di principio, la ricerca sulle radio cognitive può utilizzare qualunque tipo ti transceiver, purché questo sia flessibile, e le Software Defined Radio, sono particolarmente adatte a questo scopo.

Attualmente, la piattaforma più popolare è la **GNURadio**, la quale include un pacchetto software open source molto flessibile e compatibile con un numero limitato di dispositivi hardware. L'hardware più diffuso è il USRP GNURadio, un dispositivo sperimentale, composto da una scheda madre alla quale possono essere collegati fino a quattro schede radio, operanti in diverse frequenze (1-2800 GHz). Questo dispositivo è molto popolare grazie al costo relativamente contenuto e alla grande disponibilità di software.

**WARP-boards** è una radio sviluppata presso la Rice University (USA), che offre anche supporto alla tecnologia MIMO. [3] La diffusione limitata di questa piattaforma, trova le cause nei costi elevati dell'hardware, nella complessità di implementazione e nella scarsa quantità di codice open source disponibile.

L'**University of Kansas** ha sviluppato un'altra architettura (proprietaria). Presentata nel 2007, questa usa un processore Intel standard e offre la particolarità di essere compatibile anche con il sistema operativo Micosoft Windows. La piattaforma attualmente è proprietaria e probabilmente viene utilizzata solo all'interno della Kansas University.

La piattaforma **BEE2**, sviluppata all'Univerità di Berkeley per per esperimenti sull'*ultra-high speedbaseband processing*, viene anche utilizzata per ricerche sulle radio cognitive. Visto che anche questa piattaforma è proprietaria, costosa e la sua programmazione è relativamente complicata, è poco diffusa, se non all'interno dell'università di Berkeley.

Una delle piattaforme commerciali con un alto rapporto qualità/prezzo è la **LyrTech/Texas Instruments SDRboard**. La radio in questione offre sia un processore DSP[4] che FPGA[5]. La scheda risulta essere costosa, ma è sicuramente una delle migliori piattaforme in commercio, disponibile anche con software libero.

L'University of California at San Diego (UCSD), ha rilasciato recentemente la piattaforma **CalRadio**. In sostanza è una radio compatibile con lo standard 802.11b con il livello MAC riprogrammabile.

Un numero sempre più elevato di gruppi di ricerca, si stanno dedicando alle radio cognitive... I siti principali della ricerca in quest'ambito sono:VirginiaTech

[3]Multiple-input and multiple-output (MIMO) consiste nell'uso di antenne multiple sia lato trasmettitore che ricevitore al fine di migliorare le performance

[4]Il Digital Signal Processor, è un microprocessore ottimizzato per eseguire in maniera estremamente efficiente sequenze di istruzioni ricorrenti nel condizionamento di segnali digitali.

[5]Field Programmable Gate Array sono dispositivi digitali la cui funzionalità sono programmabili via software. L'uso di tali componenti comporta alcuni vantaggi: si tratta infatti di dispositivi standard le cui funzionalità non vengono impostate dal produttore e quindi possono essere prodotti su larga scala a basso prezzo. La loro genericità li rende adatti a un gran numero di applicazione come consumer, comunicazioni, automotive...

(USA),Trinity College a Dublino (Irlanda), WinLab presso la Rutgers University (USA), Kansas University (USA) e la RWTH Aachen University.

Anche alcune compagnie private mostrano interesse verso le radio cognitive, come ad esempio: Motorola (USA,Europa), Nokia (USA, Europa), ST Microelectronics (Europa), Microsoft (USA, Europe), Extreme Spectrum (USA) e Vanu (USA).

Nell'Unione Europea, i progetti di maggior rilievo sono il progetto E3 (Motorola, France) e l'appena concluso E2R. Va inoltre segnalato un progetto di ORACLE sull'*Opportunistic Radio Communications*.

## 1.12 Aragorn

Le architetture proposte dei precedenti paragrafi, sono alcune iniziative o ricerche nell'ambito delle reti cognitive.

Questa tesi è parte integrante del progetto europeo ARAGORN (Adaptive reconfigurable access and generic interfaces in radio networks). L'obiettivo primario del progetto ARAGORN è la ricerca e lo sviluppo di un *Cognitive Resource Manager* per la collaborazione di reti wireless. L'approccio del progetto è fondato sull'utilizzo di radio cognitive, in modo da garantire un uso efficiente delle risorse. Al progetto collaborano sette paesi dell'Unione Europea.

### 1.12.1 Obiettivi del progetto

L'obiettivo di ARAGORN non è tanto quello di sviluppare tools di simulazione per il design di radio cognitive o concentrarsi sulla ricerca nell'ambito delle software defined radio, piuttosto di produrre un prototipo commercializzabile. Il progetto spazia dalla generalità dei sistemi wireless, all'allocazione dinamica dello spettro.

Uno degli intenti è di estendere il paradigma delle radio cognitive ai livelli più alti del protocollo, prendendosi a carico anche i problemi dovuti allo scheduling, tutto questo servendosi di sofisticati algoritmi di Intelligenza Artificiale.

Oltre ad uno studio generico sull'applicazione delle tecnologie cognitive, il progetto pone particolare attenzione all'uso di tali tecniche nelle bande ISM, dato che negli ultimi anni, queste bande sono sfruttate da un numero di dispositivi e tecnologie sempre maggiore (802.11, Bluetooth, ZigBee...).

### 1.12.2 Approccio tecnico

Il progetto non mira solamente a generare nuovi metodi e conoscenze, ma anche ad avere un'influenza pratica per i futuri dispositivi wireless. L'applicazione delle metodologie provenienti dal machine learning ai sistemi wireless, ha un alto potenziale per spostare l'attuale paradigma, verso una gestione più efficiente delle reti wireless.

L'intento di ARAGORN è progettare una generica interfaccia per il CRM, in grado di accedere a tutte le informazioni di stato dei vari dispositivi e di modificarne i parametri funzionali. Il **Cognitive Resource Manager** (CRM) sviluppato da ARAGORN, servirà per ottimizzare la gestione delle risorse (in termini di spettro), per modificare la configurazione a livello rete e di link, utilizzando le informazioni provenienti dalle applicazioni, dai livelli di rete e di trasporto del protocollo TCP/IP e dai livelli inferiori dello stack protocollare.

### 1.12.3 Risultati attesi

La principale aspettativa di questo progetto, è di applicare il sistema Cognitive Resource Manager (CRM) ai dispositivi che operano delle bande ISM. L'intero sistema e il CRM, sono sviluppati su hardware riconfigurabile con avanzato controllo software. La maggior parte degli esperimenti viene svolta sulle bande ISM dei 2.4 GHz. Quello che si vuole dimostrare tramite gli esperimenti, è la praticabilità dell'approccio scelto e allo stesso tempo, si vuole sfruttare il progetto per incorporare questa tecnologia nei dispositivi commerciali.

### 1.12.4 Ambito operativo della tesi

La tesi si colloca nel progetto ARAGORN, in primo luogo nella definizione delle interfacce per l'hardware utilizzato e poi per l'implementazione di tali interfacce e dei metodi di raccolta delle informazioni.

Seguendo lo schema in Figura 2.1 a pag 34, nella tesi ci si occupa della definizione e dello sviluppo dell'Unified Link Layer API (ULLA), fornendo quindi i metodi d'accesso (API) per i layers superiori e per le tecnologie sottostanti. Inoltre, si implementa la comunicazione del layer ULLA con l'hardware CalRadio. Per tale hardware vengono definite le interfacce di comunicazione con ULLA, al fine di integrare CalRadio con lo schema di riferimento e vengono inoltre implementati i metodi di sensing e raccolta delle informazioni sull'hardware specificato. Infine viene implementata un'interfaccia grafica con la quale interagire con ULLA al fine di rilevare parametri

significatici circa lo stato dello spettro radio. Nei capitoli successivi verrà descritto in maggior dettaglio il lavoro svolto, si daranno le specifiche per le API di comunicazione, e come sono stati implementati i metodi di raccolta dei dati e di settaggio dei parametri sull'hardware CalRadio.

# Capitolo 2

# ULLA Framework

Come già accennato, ULLA è un modulo dell'architettura ARAGORN. In questo capitolo presenteremo brevemente il progetto ARAGORN, quindi la sua architettura e alcuni scenari operativi. Successivamente si parlerà del middleware a supporto delle radio cognitive, quindi del framework ULLA, che è stato esteso nell'ambito della tesi.

**Progetto ARAGORN** La missione del progetto ARAGORN è di accrescere le prestazioni in termini di riconfigurabilità, d'efficienza spettrale e di performance nei sistemi wireless attuali e futuri, tramite l'utilizzo di radio cognitive dotate di intelligenza e meccanismi propri del machine-learning.

## 2.1 Architettura di ARAGORN

In Figura 2.1 è mostrata l'architettura modulare del progetto. L'architettura di ARAGORN si basa sull'introduzione di un Cognitive Resource Manager (CRM), che ha lo scopo di ottimizzare la gestione delle risorse spettrali. Il CRM è in grado di modificare i parametri a livello di rete e link, basandosi sulle informazioni provenienti dalle applicazioni, dalla rete, dal livello di trasporto e dagli altri livelli della pila OSI.

Questo modulo può essere considerato il cervello dell'architettura, operante con tecniche di machine learning al fine di ottimizzare le comunicazioni wireless. Al fine di semplificare lo sviluppo del CRM, la sua integrazione in sistemi diversi e la sua interazione con i diversi layers del protocollo (livello applicazione, livello rete, ...), sono state definite delle interfacce generiche.

Oltre ai parametri della rete, gli input del CRM consistono anche di dati quali la posizione geografica, utili per migliorare il processo di decision-making. L'architettura è inoltre in grado di usare informazioni immagazz-

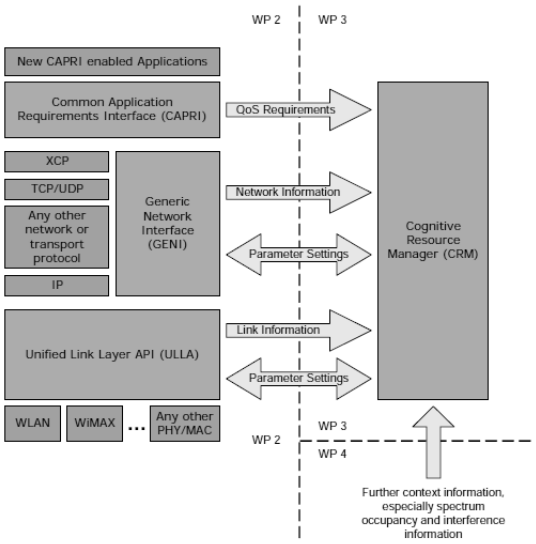



Figura 2.1: Architettura del progetto ARAGORN

inate in uno storico e dati distribuiti al fine di migliorare le performance globali e attuare un uso più efficiente delle risorse.

Il CRM è quindi un modulo che opera in modo parallelo allo stack TCP/IP. In ARAGORN lo stack protocollare è stato ulteriormente arricchito dalle interfacce di comunicazione da e verso il CRM, verso le applicazioni e verso i device fisici.

In particolar modo, l'interfaccia **CAPRI** (Common Application Requirements Interface) ha lo scopo di tradurre i requisiti delle singole applicazioni in obiettivi end-to-end da passare al modulo CRM sotto forma di requisiti di QoS.

L'interfaccia **GENI** (Generic Network Interface) opera in parallelo allo stack TCP/IP e fornisce al CRM le informazioni di livello rete. Tale interfaccia permette inoltre si manipolare i parametri dei livelli coinvolti (IP,Trasporto,...).

L'interfaccia **ULLA** (Unified Link Layer API), verrà trattata in maggior dettaglio in questo capitolo. In ogni caso questa si occupa di fornire al CRM le informazioni di livello Link e PHY e tramite le API fornite, di modificare i parametri funzionali dei dispositivi fisici.

## 2.2 Scenari operativi

La connettività di rete gioca un ruolo vitale nella vita di tutti i giorni, è uno dei fattori chiave per la nostra conoscenza nonchè un elemento essenziale nel mondo lavorativo.

A causa dell'incremento della mobilità, sia da parte delle persone che dei processi, ci si aspetta che la connettività sia disponibile sempre e ovunque ci si trovi. allo stesso tempo, la banda richiesta alla rete per lo scambio di contenuti e applicazioni, cresce esponenzialmente.

Gli scenari operativi, possono essere innanzitutto classificati in base al contesto:

- **Pubblico**: negozi, spostamenti (e.g.: pendolari)
- **Lavorativo**: in ufficio
- **Privato**: a casa

Una seconda classificazione può essere fatta in base alla priorità di utilizzo della rete:

- **bassa**: download di file
- **normale**: browsing di pagine web
- **alta**: VoIP, mail, IM
- **emergenza**: 112,113,118,... (911)

Considerando ad esempio l'ambito domestico, vediamo ora un esempio di come la tecnologia ARAGORN permetta di migliorare le performance, utilizzando l'ottimizzazione dello spettro del canale.

**Scenario** I veloci progressi delle tecnologia wireless, hanno alimentato la diffusione delle reti wireless domestiche, tanto che circa il 54% delle case in Europa hanno un computer e il 34% usano un router Wifi. Le reti casalinghe non sono composte solamente da PC, routers, PDAs ... ma anche da altri elettrodomestici che usano il canale radio, come ad esempio telecamere wireless, altoparlanti wireless, telefoni VOIP, controller per videogiochi, etc. Inoltre, grazie alla continua diminuzione dei costi delle soluzioni wireless, nel prossimo futuro sempre più dispositivi (frigoriferi, lavatrici,... ) saranno equipaggiati con connettività wireless al fine di poterli controllare a distanza. Nell'ipotetica casa del futuro, tutti i dispositivi saranno comandabili via radio, usando magari tecnologie differenti, come ad esempio Wifi, infrarosso,

802.15.4 etc. La connettivtà wireless permette loro di collegarsi ad internet o di creare dei collegamenti tra dispositivi.

Data la varietà di dispositivi, una rete domestica avrà a che fare con diversi tipi di traffico: da una comunicazione regolare con poco dispendio di banda, ad un traffico a burst, con richieste di molta banda per brevi periodi di tempo (ad esempio per applicazioni video, audio o videogiochi). Questo comportamento variabile non viene gestito dalle reti wireless attuali, in quanto l'attuale stack protocollare era stato definito per le reti cablate e quindi non fornisce un adeguato livello di adattabilità. Inoltre, le reti wireless domestiche sono soggette anche a disturbi provenienti da altri dispositivi o da altre reti wireless vicine. Va inoltre sottolineato che una corretta configurazione o una diagnosi su una rete, non sono attività che può svolgere un utente standard.

Questa visione di "casa connessa", introduce diverse richieste all'infrastruttura wireless, come molta banda (garantita), bassa latenza, uso efficiente dello spettro e facilità d'uso. Per supportare tutte queste richieste, devono essere apportate significative modifiche a livello radio, MAC, ai protocolli di routing e nelle applicazioni. Una delle limitazioni principali nell'architettura attuale, è data dagli Access Point, i quali centralizzano il controllo dei parametri radio. Attualmente molti dispositivi sono in grado di stabilire connessioni peer-to-peer, senza bisogno di passare per un Access Point. Una connessione di questo tipo, porta notevoli vantaggi: innanzitutto si riesce a raddoppiare il bit rate nella trasmissione tra peers, inoltre permette ad ogni nodo di mantenere il pieno controllo dei parametri della propria radio e pertanto si è liberi di apportare modifiche al fine di ottimizzare l'utilizzo dello spettro. In Figura 2.2 vengono mostrate le due tipologie d'accesso. La rete

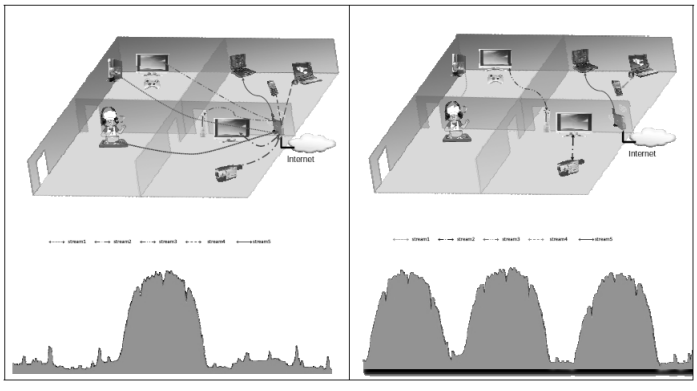



Figura 2.2: Rete domestica: accesso centralizzato (AP) vs connessioni ibride

che utilizza l'Access Point, vede la competizione da parte di tutti i dispositivi

(compreso l'Access Point) nell'accesso alla stessa porzione dello spettro radio. Un approccio ibrido, ovvero decentralizzato per le comunicazioni punto punto e infrastruttutato per la coordinazione, i singoli device possono modificare a piacimento i loro parametri trasmissivi e scegliere il canale trasmissivo più adeguato. Questo porta ad un utilizzo migliore dello spettro, come si può notare in Figura 2.2.

I principali obiettivi di ottimizzazione della comunicazione in una rete domestica sono:

- **Reazione dinamica alle interferenze**

  La maggior parte dei sistemi wireless è sensibile alle interferenze e possiedono solo una limitata capacità di resistenza.

- **Interventi proattivi per evitare interferenze e collisioni**

  Gli interventi proattivi prevedono l'utilizzo di tecniche di machine learning al fine di evitare le collisioni e le interferenze, basandosi su esperienze passate.

- **Miglioramento della QoS tramite ottimizzazione dell'utilizzo delle risorse**

  Questa categoria tratta la qualità del servizio per diverse applicazioni. L'approccio di ARAGORN vuole essere diverso da quello usato attualmente nelle reti, cercando di ottimizzare l'allocazione delle risorse tra applicazioni concorrenti.

- **Ottimizzazione dell'utilizzo dello spettro**

  Questa categoria si concentra sul miglioramento dell'utilizzo delle risorse radio nell'ambiente domestico.

- **Ottimizzazione dell'utilizzo dello spettro tra l'abitazione e le reti vicine**

  L'interferenza tra le reti wireless di due casa vicine, è un fattore che influenza notevolmente le performance. Il sistema ARAGORN permette di limitare tale interferenza riducendo la potenza trasmissiva al minimo richiesto e cerca di sopraffare il fenomeno di frammentazione dello spettro dovuto ad ottimizzazioni locali.

- **Configurazione iniziale e mantenimento della Qos tramite un gateway centralizzato**

  Questa categoria tratta la configurazione iniziale dei dispositivi wireless e il mantenimento della QoS qualora una nuova applicazione inizi ad utilizzare la rete.

La tesi si occupa principalmente dell'ottimizzazione dell'utilizzo dello spettro, tramite il sensing dei canali radio. Pertanto tratteremo ora solamente il questa casistica di ottimizzazione.

### Negoziazione dinamica dei parametri radio

Supponiamo che Bob voglia vedere un film in HD, il quale è salvato su un media center. Seleziona il video e inizia la riproduzione. Il video verrà trasmesso sul display,usando la tecnologia wireless 802.11g.

**senza ARAGORN** Bob deve configurare manualmente i parametri trasmissivi sia del media server che del display. In alternativa può utilizzare i parametri di default, dettati dall'Access Point. Pensando a Bob come un utente standard, questo auspicabilmente sceglierà di usare le impostazione predefinite, o al più eseguirà un settaggio casuale dei parametri.

**con ARAGORN** Il sistema ARAGORN, presente su ogni dispositivo, è a conoscenza della configurazione attuale dell'ambiente wireless. Pertanto, in seguito ad un'operazione di negoziazione, nella quale vengono configurati in modo ottimale i parametri radio (banda, frequenza, etc.), la connessione verrà stabilita senza che Bob debba intervenire.

Il processo di negoziazione, tiene conto dell'utilizzo attuale dello spettro e delle situazioni passate, nonchè delle richieste di banda e latenza... L'insieme delle configurazioni ottimali, è formato da diversi parametri relativi al protocollo e alla radio, come il canale usato, il protocollo MAC usato e i relativi parametri, il massimo rate trasmissivo supportato dalla sorgente, il bit-rate del video, etc.

### Migliorare l'utilizzo dello spettro tramite la conoscenza dell'utilizzo

Supponiamo ora che, finchè Bob sta guardando il film, suo figlio Ben cominci a giocare con l'Xbox, la quale è collegata al display tramite un collegamento wireless. Se le due connessioni utilizzano la stessa frequenza, dovranno competere per l'accesso al canale.

**senza ARAGORN** É spesso difficile per gli utenti non esperti configurare i parametri radio in modo corretto. É addirittura impossibile che un utente modifichi in modo dinamico i parametri della trasmissione, senza interrompere l'applicazione.

**con ARAGORN** Grazie all'attività di spectrum scanning, il CRM di ARAGORN possiede una conoscenza accurata dell'ambiente wireless. Quando inizia la negoziazione della connessione di Ben, il CRM è a conoscenza dell'attuale trasmissione dello stream di Bob, pertanto configurerà la nuova trasmissione, in modo che utilizzi un'altra frequenza trasmissiva. In questo modo, lo spettro è sfruttato meglio e sia Bob che Ben avranno un'esperienza più gratificante.

## 2.3 Supporto Middelware per Radio Cognitive su sistemi Linux

Nei sistemi Linux, è possibile accedere alle impostazioni dei dispositivi wireless, mediante dei metodi standard, denominati *Wireless Extensions*. Questi metodi sono supportati dalla maggior parte dei driver. Esistono inoltre una serie di tools per interagire con questi metodi, noti come *wireless tools*.

### 2.3.1 Wireless Extensions

Le Wireless Extensions sono una collezione di API generiche che permettono di ottenere informazioni e statistiche dalle schede wireless, oltre a permettere la modifica di alcuni parametri funzionali. Permettono inoltre la ricezione di segnali asincroni provenienti dai device. Queste API sono state sviluppate da J.Tourrilhes al fine di fornire un maggior supporto alla tecnologia 802.11. Consistono in un insieme di funzioni (handlers) ognuna delle quali si occupa di gestire una specifica *ioctl*. Ogni driver esporta la "lista" degli handlers supportati e che quindi potranno essere invocati tramite le librerie *iwlib*. Quando le Wireless Extensions vengono abilitate nella configurazione del kernel, possono essere gestiti gli eventi generati dai device. Non tutti i dispositivi implementano le le Wireless Extensions, e nel caso lo facciano, non è detto che siano implementate tutte le funzionalità disponibili nella librerie *iwlib*.

### 2.3.2 Wireless Tools

I *wireless tools*, sono un insieme di utility atte a manipolare le Wireless Extensions. Solitamente consistono in interfacce testuali con le quali modificare i parametri funzionali supportati dai dispositivi wireless (invocando per l'appunto le Wireless Extensions).

**iwconfig** É simile al comando ifconfig, solo che opera sulle interfacce wireless. Viene usato per impostare parametri generici e mostrare statistiche quali l'ID della rete, il canale utilizzato ...

**iwlist** Questo tool viene utilizzato per trovare le reti wireless a distanza di rilevamento, gli Access Point e i loro MAC address e i rispettivi SSID.

**iwpriv** Fornisce la lista delle estensioni specifiche di un device.

**iwspy** Gli utenti possono specificare una lista di indirizzi di rete da monitorare. Il tool restituisce statistiche circa la qualità del segnale proveniente dal tali indirizzi e aggiorna le statistiche ogni qual vota riceve un pacchetto da uno di quegli indirizzi.

**iwevent** Mostra gli eventi generati dal dispositivo wireless selezionato.

## 2.4 Supporto Middleware per Radio Cognitive su sistemi Windows

Windows Server 2008 e Windows Vista, includono nuove funzionalità legate alle tecnologie di networking, come una nuova implementazione del protocollo TCP/IP conosciuto come **the Next Generation TCP/IP Stack**, la Windows Filtering Platform, etc. [15]. La Next Generation TCP/IP, supporta sia l'architettura IP versione 4 sia la versione 6, le quali condividono il Transport layer. L'architettura è mostrata in Figura 2.3.

Lo stack TCP/IP fornisce tre principali API verso applicazioni e servizi:

- **WSK** usate dalle socket in kernel space.
- **Windows socket** usate in user space.
- **TDI** usate da NetBIOS over TCP/IP e altri client TDI.

Dato che la tesi verte sull'implementazione in ambiente Linux, si rimanda alla knowledge base Microsoft, per ulteriori informazioni circa l'architettura (http://technet.microsoft.com/en-us/default.aspx).

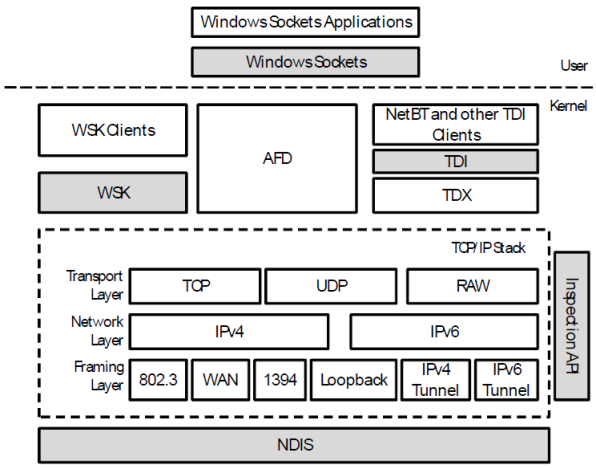



Figura 2.3: Architettura dello stack TCP/IP in Windows Vista e Windows Server 2008

## 2.5 ULLA

ULLA (Unified Link Layer API) è il middleware a supporto dei device Cal-Radio. Sviluppato durante il progetto Gollum [16], definisce un framework generico per la programmazione del livello Data Link dello stack protocollare.

Il framework multi piattaforma, viene usato per astrarre i dispositivi specifici in un'interfaccia generica. In particolare ULLA costituisce uno strato intermedio nello stack e va inserito al di sopra del livello Link. In Figura 2.4, viene mostrato dove si colloca il modulo ULLA. Come si nota in Figura,

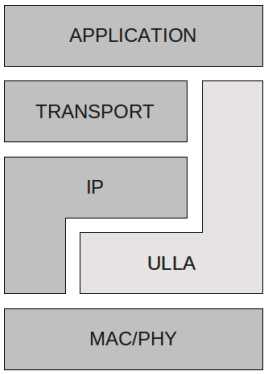



Figura 2.4: Integrazione del layer ULLA nello stack

il modulo ULLA, è collegato direttamente al livello applicativo dello stack protocollare. Questo significa che ULLA offre delle API tramite le quali le applicazioni possono interagire con il framework.

Lo scopo per cui è stato progettato ULLA, è quello di definire un'interfaccia unica ed astratta per diversi tipi di tecnologie, come reti wireless, Bluetooth, WSN, etc. In sostanza, al si sotto del framework può essere implementata qualunque tecnologia, purchè questa esponga verso ULLA l'interfaccia prestabilita. Quando verrà spiegata l'architettura di ULLA, questi concetti risulteranno più chiari.

Allo stesso modo, possono essere sviluppate svariate applicazioni che accedono ai dati forniti da ULLA e, attraverso le API verso il livello applicativo, queste applicazioni possono sia richiedere statistiche di livello fisico e di link, sia impostare parametri funzionali.

Grazie ad ULLA, gli sviluppatori di applicazioni, non dovranno preoccuparsi dell'hardware sottostante, in quanto questo viene astratto dal middleware. Nell'ottica del progetto ARAGORN, il CRM altro non è che un'applicazione che interagisce con il framework ULLA. Il fatto che ULLA sia multi piattaforma, consente lo sviluppo di applicazioni su diversi sistemi operativi. Nella tesi si è adoperato e sviluppato il framework nella sua versione per il solo sistema GNU/Linux.

## 2.6 Stato dell'arte del framework ULLA

Il framework è stato sviluppato nel progetto GOLLUM [16] e offre un'architettura modulare estendibile. Secondo quanto riportato nel sito del progetto ULLA [17], la definizione della API, specificate nei *deliverables*, e la corrente implementazione, non sono completamente conformi.

Senza addentrarsi nello specifico di cosa sia effettivamente implementato, ULLA permette di interrogare un generico dispositivo wireless, circa il suo stato, acquisendo statistiche di livello MAC e PHY. L'interrogazione avviene tramite dei tool, che sfruttano le API di ULLA. Le richieste vengono fornite con una sintassi simile all'SQL (usato nella programmazione dei DataBase).

Il framework altro non è che un middleware che permette di astrarre un qualunque dispositivo wireless e di eseguire operazioni di richiesta o impostazione di parametri. La flessibilità di ULLA permette di interfacciarsi non solo con dispositivi che implementano la tecnologia 802.11, come ad esempio le WSN. A tal proposito, il progetto ULLA ha sviluppato del software apposito [18].

Le API di ULLA ,pertanto, permettono di interagire con i device in modo da manipolare parametri di livello MAC e PHY e allo stesso modo di recuperare informazioni circa quei parametri, in modo sincrono e asincrono, richiedendo parametri singoli o multipli, su una o più tecnologie. Il progetto

ULLA fornisce la base per lo sviluppo di applicazioni che interagiscono con "qualunque" dispositivo wireless, tramite l'utilizzo di API ben definite.

La piattaforma, come è stata rilasciata, risulta essere un valido punto di partenza per lo sviluppo di applicazioni e per l'integrazione di dispositivi fisici. In particolare, si possono distinguere tre direzioni sulle quali intervenire per estendere il framework ULLA:

1. Sviluppo di applicazioni che utilizzano il framework
2. Integrazione di nuovi dispositivi
3. Sviluppo del framework

**Sviluppo di applicazioni** Questa categoria indica la creazioni di applicazioni che, basandosi sulle API di ULLA, collezionano statistiche dai dispositivi hardware per poi rielaborarle. La versione rilasciata di ULLA, offre alcuni tool che utilizzano le API a questo scopo.

**Integrazione di nuovi dispositivi** ULLA offre un'astrazione dei dispositivi fisici. Ogni dispositivo che vuole interagire con il framework ULLA, dovrà essere sviluppato in modo tale che le API di ULLA, possano interrogarlo correttamente.

L'attuale versione del framework fornisce un'implementazione dei driver per i dispositivi Atheros [19], ovvero per i dispositivi che utilizzano i driver MadWifi [20].

**Sviluppo del framework** In questa categoria rientrano tutte quelle modifiche ed estensioni al framework in modo da renderlo completamente conforme alle specifiche. Ancora una volta, secondo quanto riportato nel sito del progetto ULLA [17], il framework non è ancora stato sviluppato al 100%. Serve quindi un'attività atta ad implementare le funzionalità mancanti. In questa categoria, rientrano inoltre tutte quelle attività atte ad implementare nuove features o ad estendere le API esistenti.

Nella tesi, vengono svolte attività collocabili su tutte e tre le categorie annunciate: l'integrazione dei dispositivi CalRadio, l'estensione delle funzionalità del framework per consentire la raccolta di nuovi tipi di statistiche e lo sviluppo di alcune funzionalità non completamente implementate e la creazione di un'interfaccia grafica (come applicazione) per visualizzare le statistiche raccolte.

## 2.7 Architettura

L'architettura modulare di ULLA, si può suddividere in tre gruppi principali:

- **LU** Link User application. Consiste nell'insieme delle applicazioni che usano il framework ULLA mediante le API disponibili al fine di raccogliere statistiche ed impostare parametri operativi.
- **ULLA-core** É il motore del framework. Offre le API verso le applicazioni, gestisce le temporizzazioni, interpreta i comandi,...
- **LLA** É lo strato inferiore dell'architettura. Rappresenta l'astrazione dei dispositivi fisici.

In Figura 2.5 è mostrata l'architettura del framework ULLA. Si possono

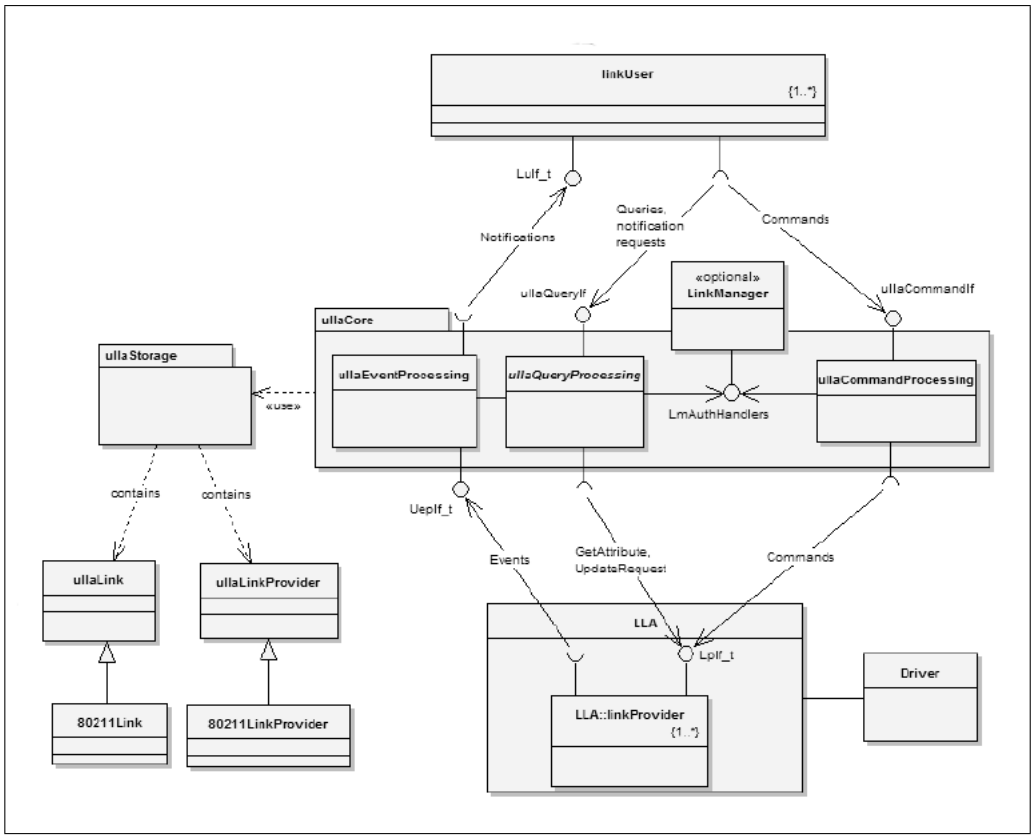



Figura 2.5: Architettura del framework ULLA

facilmente distinguere i moduli descritti precedentemente e i collegamenti che identificano le modalità e le direzioni della comunicazione tra questi. In

aggiunta vanno segnalati il modulo per il salvataggio delle statistiche (ULLA-Storage) e il collegamento con il dispositivo fisico (Driver). La comunicazione tramite le applicazioni (Link User) e l'ULLA-core, come quella tra ULLA-core e LLA (link layer adapter) avviene tramite API. Nei paragrafi successivi verranno descritte le API usate per la comunicazione e successivamente verrà data una descrizione del funzionamento dei singoli moduli. Successivamente verranno presentati i moduli di ULLA a supporto dei dispositivi CalRadio e come questi interagiscono con l'intera architettura.

### UQL

ULLA Query Language, è una grammatica derivata dal più generico linguaggio SQL (Structured Query Language) solitamente usato nella programmazione di DataBase. La sintassi UQL è molto simile a quella SQL e in particolare le clausole di **SELECT** e di **UPDATE** permettono rispettivamente di leggere e modificare il valore di un parametro. Come per SQL, sono inoltre definiti degli operatori quali la media (AVG), il massimo (MAX), etc. che possono aggregare i risultati ottenuti da un'interrogazione.

Rispetto al generico SQL, UQL non permette di eseguire query annidate, join e l'utilizzo di variabili temporanee. É invece ammesso l'uso del quantificatore *.

La sintassi della grammatica UQL aderisce alla Backus Naur Form (BNF). BNF è una meta-sintassi usate per descrivere grammatiche libere dal contesto, ovvero è un modo formale per descrivere linguaggi formali.UQL può pertanto essere interpretata da YACC [1] Per una trattazione più dettagliata di questo linguaggio, si rimanda all'Appendice A.1

## 2.7.1 APIs

Le API offerte dal core di ULLA, possono essere suddivise in due categorie:

- Link User API
- Link Provider API

Le prime sono rivolte alle applicazioni che stanno "sopra" ad ULLA, ovvero quelle applicazioni definite ed usate dagli utenti (Link User).

[1] **YACC (Yet Another Compiler Compiler)** è un generatore di parser nel linguaggio di programmazione C. YACC è simile a JavaCC poichè genera un parser per una grammatica fornita nella notazione BNF, ma il codice in output è in C.

Le Link Provider API, invece, sono quelle rivolte ai moduli "sotto" il core di ULLA. I Link Provider (LP) sono un'astrazione delle *Network Interface Card* (NIC). Per ogni LP viene definita una tabella di descrittori, dove vengono specificati gli attributi e i comandi disponibili per interagire con l'hardware specifico.

### Link User API

Le API verso i link users, permettono di controllare i device fisici collegati ad ULLA. Le API si dividono in tre principali categorie: API per visualizzare/-modificare il valore dei parametri, API dedicate all'esecuzione di comandi e API atte a conoscere i parametri e i comandi disponibili per una specifica NIC.

**ullaRequestReflection** Queste API permettono di conoscere i parametri e i comandi che una specifica NIC espone. L'ullaRequestReflection API è cosi definita:

```
ullaResultCode_t ullaRequestReflection(const ullaString_t
    uqlQuery, ullaResultId_t *urId);
```

`urId` è un puntatore alla locazione di memoria dove sono memorizzati i risultati.
`uqlQuery` indica la query specificata nel linguaggio UQL (ULLA Query Language).
La chiamata a questa funzione ritorna "ULLA_OK" in caso l'operazione sia andata a buon fine, altrimenti un codice d'errore. (vedi Appendice A.2).

Ad ogni NIC sono assegnate delle tabelle che descrivono i parametri e i comandi disponibili. Solitamente sono divise per tipologia di dato; ad esempio una tabella può contenere tutti i parametri di livello fisico, mentre un'altra le statistiche di livello MAC. Alcuni esempi di richieste sono:

- Richiedere l'elenco delle tabelle (descrittive) per un dato NIC.
- Richiesta di tutti gli attributi disponibili per una data tabella. Con questa interrogazione, viene restituito l'elenco degli attributi disponibili, compreso il tipo di dato restituito ed un breve descrizione del significato di ogni parametro.
- Richiedere i comandi eseguibili da un determinato device.

Per alcuni esempi di sintassi e dei valori restituiti da queste interrogazioni, si veda l'appendice A.3.

**ullaRequestAttribute** Questa è una delle API principali. Permette di richiedere, tramite una query UQL, il valore di uno o più parametri. Per i parametri modificabili, sono ammesse anche le operazioni di UPDATE, tramite le quali si possono impostare i parametri funzionali del device. La definizione è:

```
ullaResultCode_t ullaRequestAttribute(const ullaString_t
    uqlStatement, ullaResultId_t *urId, const ullaTime_t
    validity);
```

`urId` è un puntatore alla locazione di memoria dove sono memorizzati i risultati.

`uqlQuery` indica la query specificata nel linguaggio UQL (ULLA Query Language).

`validity` è un numero intero che indica la *freshness* di un attributo, espressa in ms.

La chiamata a questa funzione ritorna "ULLA_OK" in caso l'operazione sia andata a buon fine, altrimenti un codice d'errore. (vedi Appendice A.2).

In base al tipo di clausola specificata nella query UQL (select o update), la chiamata dell'API svolge due funzioni differenti. Nel caso vena eseguita una query di SELECT, l'API restituisce semplicemente il valore dell'attributo richiesto (eventualmente più di uno). Invece, se viene somministrata una query di UPDATE, l'API cambia il valore del parametro interessato (in questo caso ogni query modificherà un solo parametro)e successivamente restituirà il valore corrente. Per alcuni esempi di sintassi e dei valori restituiti da queste interrogazioni, si veda l'appendice A.4.

**ullaRequestNotification** Le API di notifica permettono di creare un processo di osservazione su un dato parametro.

```
ullaResultCode_t ullaRequestNotification(
    ullaNotificationRequestId_t *rnId, const ullaString_t
    uqlQuery, const ullaTime_t validity, const unsigned int count
    );
```

```
ullaResultCode_t ullaReceiveNotification(const
    ullaNotificationRequestId_t rnId, ullaResultId_t *urId);
```

```
ullaResultCode_t ullaCancelNotification(const
    ullaNotificationRequestId_t rnId);
```

```
ullaResultCode_t ullaNotificationRequestStatus(const
    ullaNotificationRequestId_t rnId);
```

`urId` è un puntatore alla locazione di memoria dove sono memorizzati i risultati.

`rnId` è un identificativo della notifica corrente: ulla supporta notifiche multiple, pertanto tramite questo puntatore, si distinguono le varie notifiche. `uqlQuery` indica la query specificata nel linguaggio UQL (ULLA Query Language).

`validity` è un numero intero che indica la *freshness* di un attributo, espressa in ms.

`count` indica il numero di notifiche da ricevere. Dopo aver ricevuto `count` notifiche, viene invocata la procedura di cancellazione della richiesta.

`ullaRequestNotification` serve per inizializzare una richiesta di notifica su un parametro. `ullaReceiveNotification` è un evento che viene generato ogni qualvolta una notifica diviene disponibile e restituisce il valore richiesto. `ullaCancelNotification` permette di terminare la richiesta di notifiche per un dato parametro. Infine, `ullaNotificationRequestStatus` restituisce lo stato attuale del meccanismo di notifica, permettendo di capire se sono disponibili delle notifiche, se il processo è stato arrestato, etc.

Il meccanismo di notifica implementato da ULLA, si basa sull'utilizzo dei thread. in particolare, ogni volta si richiede una notifica, viene inizializzato un thread che si occupa di restituire i risultati attesi ogni qualvolta questi siano disponibili. Nel richiedere una notifica, è necessario fornire alcuni parametri aggiuntivi, come il numero totale di notifiche che si intende ricevere. Le notifiche sono un'utilità molto potente: queste difatti consentono di ricevere statistiche relative a uno o più parametri, qualora un parametro ben preciso, supera una soglia prefissata. Pertanto questo metodo evita alle applicazioni di eseguire continue query che appesantirebbero la comunicazione con ULLA, ricevendo i dati richiesti, solo al verificarsi di determinati eventi. Nel caso in cui, i LU vogliano ricevere informazioni ad intervalli periodici, senza necessariamente specificare soglie o quant'altro, potranno utilizzare l'ullaRequestSample API.

**ullaRequestSample** Questa API consente ad un'applicazione di richiedere aggiornamenti periodici per uno o più parametri. anche in questo caso si vede come l'onere di formulare continue richieste viene demandato al core di ULLA, e non alle applicazioni.

La definizione di questa API, prevede le seguenti chiamate:

```
ullaResultCode_t ullaRequestSample(ullaSampleRequestId_t *rnId,
    const ullaString_t uqlQuery, const ullaTime_t validity,
    const unsigned int count, const unsigned int period);
```

```
ullaResultCode_t ullaReceiveSample(const ullaSampleRequestId_t
    rnId, ullaResultId_t *urId);
```

```
ullaResultCode_t ullaCancelSample(const ullaSampleRequestId_t
    rnId);
```

```
ullaResultCode_t ullaSampleRequestStatus(const
    ullaSampleRequestId_t rnId);
```

`urId` è un puntatore alla locazione di memoria dove sono memorizzati i risultati.
`rnId` è un identificativo della notifica corrente: ulla supporta notifiche multiple, pertanto tramite questo puntatore, si distinguono le varie notifiche.
`uqlQuery` indica la query specificata nel linguaggio UQL (ULLA Query Language).
`validity` è un numero intero che indica la *freshness* di un attributo, espressa in ms.
`count` indica il numero di notifiche da ricevere. Dopo aver ricevuto `count` notifiche, viene invocata la procedura di cancellazione della richiesta.
`period` indica la periodicità delle richieste, espressa in ms.

L'utilizzo di queste chiamate è del tutto analogo al caso delle API di notifica.

**ullaRequestCommand** ULLA permette di inviare comandi verso i dispositivi fisici, come richieste di reset delle statistiche, piuttosto che attivazione o disattivazione di alcune funzionalità...

Le API per svolgere queste operazioni, sono le seguenti:

```
ullaResultCode_t ullaRequestCommand(const ullaString_t uqlQuery,
    ullaResultId_t *urId, const ullaTime_t timeout);
```

`urId` è un puntatore alla locazione di memoria dove sono memorizzati i risultati.

`uqlQuery` indica la query specificata nel linguaggio UQL (ULLA Query Language).
`timeout` Indica un periodo in ms, oltre il quale, se il comando non è stato eseguito, viene abortito. É un parametro opzionale e nel caso non venga specificato, allora non viene utilizzato alcun meccanismo di timeout.

Nel caso il comando venga eseguito correttamente, il valore di ritorno sarà "ULLA_OK", altrimenti un codice d'errore.

### Link Provider API

Le Link Provider API, sono metodi d'accesso da e verso il modulo LLA (Link Layer Adapter) per l'ULLA-core.

Vengono usate dal core di ULLA per inizializzare o rimuovere dei componenti LP, per registrare nuovi componenti e per gestire gli eventi generati dai vari Link Provider. Queste API devono essere esposte da tutti quei driver che intendono interfacciarsi con ULLA.

Ogni dispositivo di rete, viene virtualizzato dal modulo LLA. In ULLA, vengono distinti due tipi di moduli LLA: Link e LinkProvider. I moduli LinkProvider, astraggono le funzionalità di livello MAC e PHY dei dispositivi di rete virtualizzati. I moduli Link, invece, si occupano di raccogliere le statistiche a livello di singolo link di comunicazione, dove ogni link è identificato in maniera univoca da una coppia di indirizzi MAC sorgente-destinazione.

Per quanto riguarda i moduli LinkProvider, ogni device, deve implementare la seguente interfaccia:

```
typedef struct ulla_lp {
  ulla_data_pointer_t class_ids     __attribute__ ((packed));
  ulla_data_length_t class_ids_len __attribute__ ((packed));
  ulla_lla_if_t *interface          __attribute__ ((packed));
} __attribute__ ((packed)) ulla_lp_t;
```

Dove:

```
typedef struct ulla_class {
    ulla_name_t name                        __attribute__ ((
        packed));
    ulla_name_length_t name_len             __attribute__ ((
        packed));
    ulla_version_t version                  __attribute__ ((
        packed));
    ulla_version_length_t version_len       __attribute__ ((
        packed));
    ulla_string_t description               __attribute__ ((
        packed));
    ulla_string_length_t description_len    __attribute__ ((
        packed));
```

```
    ulla_data_pointer_t attributes        __attribute__ ((
        packed));
    ulla_data_length_t attributes_len     __attribute__ ((
        packed));
    ulla_data_pointer_t commands          __attribute__ ((
        packed));
    ulla_data_length_t commands_len       __attribute__ ((
        packed));
  } __attribute__ ((packed)) ulla_class_t;
```

e

```
  typedef struct ulla_lla_if {
    ulla_GetAttribute_t GetAttribute;
    ulla_RequestUpdate_t RequestUpdate;
    ulla_CancelUpdate_t CancelUpdate;
    ulla_ExecCmd_t ExecCmd;
    ulla_SetAttribute_t SetAttribute;
  } ulla_lla_if_t;
```

In sostanza la classe indica una descrizione ed un'identificazione di ogni dispositivo astratto.

Invece l'interfaccia, indica quelle funzioni che devono essere implementate nei driver dei dispositivi, al fine di poter interagire con il framework ULLA per poter soddisfare le richieste provenienti dall'ULLA-core. Mentre la definizione della classe consiste in una semplice compilazione di campi descrittivi, la parte dell'interfaccia consiste nell'implementazione delle funzioni descritte. Più in dettaglio:

```
typedef int (*ulla_GetAttribute_t) (const ulla_id_t objId, const
    ulla_id_t classId, const ulla_id_t attributeId,
    ulla_data_pointer_t data, ulla_data_length_t *length,
    ulla_timestamp_t *timestamp);
```

```
typedef int (*ulla_SetAttribute_t) (const ulla_id_t objId, const
    ulla_id_t classId, const ulla_id_t attributeId, const
    ulla_data_pointer_t data, const ulla_data_length_t length,
    ulla_timestamp_t *timestamp);
```

Queste due funzioni servono rispettivamente per leggere il valore di un parametro e modificare il parametro.

```
typedef int (*ulla_RequestUpdate_t) (const ulla_id_t objId,
    const ulla_id_t classId, const ulla_id_t attributeId,
    ulla_id_t *ruId);
```

```
typedef void (*ulla_CancelUpdate_t) (const ulla_id_t objId,
    const ulla_id_t ruId);
```

Queste, invece, vengono utilizzate per le richieste periodiche. In riferimento alle API verso il livello LU, queste sono la controparte atta a fornire le notifiche e i campioni (samples).
`objId` è un identificativo univo del device verso il quale viene inoltrata la richiesta.
`classId` indica la classe descrittiva del device.
`attributeId` indica l'attributo per il quale soddisfare la richiesta.
`data` indica un puntatore all'area di memoria nella quale inserire i risultati.
`length` indica la dimensione, in byte, dei risultati.
`timestamp` è una struttura dati per memorizzare il tempo trascorso per l'elaborazione dei risultati.
`ruId` viene usato nelle notifiche e serve per identificare in modo univoco una richiesta periodica.
Il valore ritornato da queste funzioni, è "ULLA_OK" in caso l'operazione sia andata a buon fine, altrimenti un codice d'errore.

Viene inoltre definita un'API che serve per l'esecuzione di comandi da parte del device:

```
typedef int (*ulla_ExecCmd_t) (const ulla_id_t objId, const
    ulla_id_t classId, const ulla_id_t commandId, const
    ulla_time_t timeout, ulla_timestamp_t *timestamp);
```

`commandId` è un identificativo univoco del comando richiesto. `timeout` è un valore opzionale, che indica un tempo limite entro il quale il comando deve essere eseguito. É il valore fornito nell'invocazione della medesima API a livello LU.

Per poter utilizzare un nuovo LP, questo deve essere in grado di registrarsi presso il livello LLA del framework ULLA. La procedura di registrazione, consente di segnalare ad ULLA che è disponibile un nuovo device (in fase di de-registrazione verrà segnalato il contrario) e quindi di conoscere il suo stato e quali comandi e parametri sono da esso implementati. Solitamente, alla fase di registrazione, segue una fase di inizializzazione. La definizione delle API atte alla registrazione è:

```
int ulla_registerLP (const ulla_lp_t *lp, ulla_id_t *lpId);
```

```
void ulla_unregisterLP (const ulla_id_t lpId);
```

`lp` è un parametro definito nel LP che si vuole registrare, e rappresenta la sua struttura.
`lpId` è un identificativo univoco assegnato da ULLA.

Adesso che sono state definite le API a supporto del core di ULLA, vediamo come vengono definiti gli altri componenti dell'architettura.

### 2.7.2 LU

Il livello Link User, è composto dalle applicazioni che usano il framework ULLA. Tali applicazioni interagiscono con ULLA tramite le API specificate precedentemente. Le interrogazioni, formulate con il linguaggio UQL, vengono interpretate da ULLA tramite un parser (YACC) e successivamente viene inviata una richiesta al livello LLA.

Secondo quanto definito nelle specifiche del progetto GOLLUM, quando una nuova applicazione vuole usare ULLA, questa deve prima registrarsi e rispettivamente de-registrarsi non appena termina di interagire con ULLA. Questo meccanismo, tuttavia non viene gestito da ULLA.

Invece, quando un'applicazione vuole usare ULLA, semplicemente richiama l'API più adatta per interagire con il framework.

I link user, possono avvalersi delle API per richiedere attributi in modo asincrono, oppure in modo sincrono tramite il campionamento. Possono inoltre sorvegliare uno o più parametri, demandando tutto il carico di lavoro al core di ULLA, tramite il meccanismo delle notifiche.

Il framework ULLA presenta di default delle utility dimostrative, atte a mostrare il funzionamento delle API (Link User API). Questi programmi consentono, tramite un'interfaccia testuale, di interrogare ULLA e di impartire comandi. In particolare, la suite messa a disposizione, comprende:

Richiesta asincrona di attributi singoli o multipli:

```
UllaAttribute "SELECT|UPDATE <attribute>{,<attribute1>{...}}
    FROM {ullaLinkProvider|ullaLink} [WHERE <attribute> {<|=|>} <
    value>] ;"
```

Richiesta periodica di uno o più attributi specificando eventualmente il numero di campioni e l'intervallo di campionamento:

```
UllaSample "SELECT <attribute>{,<attribute1>{...}} FROM {
    ullaLinkProvider|ullaLink} [WHERE <attribute> {<|=|>} <value
    >] ;" <PERIOD> <COUNT>
```

Richiesta di notifiche su uno o più parametri in base al soddisfacimento di una data proprietà; possibilità di inserire il numero massimo di notifiche che si desidera ricevere:

```
UllaNotification "SELECT <attribute>{,<attribute1>{...}} FROM {
    ullaLinkProvider|ullaLink} [WHERE <attribute> {<|=|>} <value
    >] ;" <COUNT>
```

Esecuzione di un comando presso uno specifico LP. L'LP designato viene selezionato dalla clausola WHERE:

```
UllaCommand "SELECT <attribute>{,<attribute1>{...}} FROM {
    ullaLinkProvider|ullaLink} [WHERE <attribute> {<|=|>} <value
    >] ;"
```

Query di reflection che riportano dati sommari circa i parametri e i comandi disponibili:

```
1 UllaReflection "SHOW TABLES;"
2 UllaReflection "SHOW ATTRIBUTES FROM <table>;"
3 UllaReflection "SHOW COMMANDS FROM <table>;"
```

Per degli esempi dell'utilizzo di questi tools, si faccia riferimento all'Appendice A.

Inoltre, nell'ambito della tesi, è stato sviluppato un LU dimostrativo, composto di un'interfaccia grafica, per la visualizzazione di alcuni parametri di livello MAC e PHY. Di questo viene parlato nell'Appendice B

### 2.7.3 ULLA core

ULLA core, come suggerisce il nome, è il modulo centrale del framework. Si occupa del parsing delle richieste da parte dei LU, della sincronizzazione dei thread, della registrazione delle interfacce di rete, etc.

Il modulo, non ha una struttura monolitica, viceversa è composto di tre moduli secondari che cooperano tra loro:

**UQP** ULLA Query Processing.

Questo modulo si occupa della gestione delle query provenienti dai LU. Tali query possono essere delle selezioni sugli attributi, oppure degli update.

**UCP** ULLA Command Processing.

É il modulo responsabile della gestione dei comandi impartiti dai LU.

**UEP** ULLA Event Processing.

Questo modulo prende a carico tutti gli eventi provenienti dagli LP, come aggiornamenti di attributi, registrazioni di nuovi Link Provider, etc.

### 2.7.4 ULLA Storage

A supporto del core, vi è inoltre il modulo **ULLAStorage**.Questo modulo volge un duplice ruolo: da una parte si interpone nelle comunicazioni con i driver, quando vengono generate richieste su degli attributi. L'altra funzionalità svolta da questo modulo, è lo storage dei dati, servendosi di un DataBase esterno a cui collega.

Mentre questa seconda funzionalità è opzionale, la prima non lo è. L'intermediazione tra i driver e il gestore delle richieste, avviene in una sorta di modalità buffer, per cui i dati non vendono mai prelevati direttamente dal driver del singolo dispositivo, ma dal modulo ULLAStorage.

## 2.8 LLA

L'ultimo modulo da considerare, è il Link Layer Adapter. Questo livello dell'architettura, svolge un ruolo cruciale nel funzionamento del framework.

Esso infatti si occupa dell'astrazione dei dispositivi fisici ad esso collegati.

In particolare, dato un messaggio (query) generica, questo modulo è in grado di diversificarlo per l'hardware specifico. In altre parole, una richiesta formulata da un LU, per richiedere il valore di un specifico parametro, viene modellata dal livello LLA, in modo da interfacciarsi con ogni tecnologia sottostante, ammesso che questo parametro sia disponibile sui dispositivi collegati. Ad esempio, una richiesta circa la dimensione della Congestion Window, viene rielaborata dal modulo LLA in modo da interrogare tutti i dispositivi hardware connessi, siano questi dispositivi Atheros, piuttosto che Intel o CalRadio, rendendo quindi la piattaforma Hardware independent.

## 2.9 Contributo

Il contributo della tesi, si concentra principalmente nello sviluppo di un modulo LP per i dispositivi CalRadio. Inoltre, al fine di implementare tutte le funzionalità previste da ULLA, sono stati modificati o espansi alcuni moduli del core di ULLA.

Un altro aspetto chiave della tesi, nell'ottica del progetto ARAGORN, è stata la definizioni di alcuni parametri funzionali aggiuntivi al set standard di ULLA, in modo da consentire altri tipi si statistiche. Questi nuovi tipi di parametro vengono supportati da una nuova struttura dati che permette di rendere ULLA oltre che hardware independent, anche *technology independent*.

Nello sviluppare queste nuove funzionalità, ci si è imbattuti in alcuni bug e lacune del framework, che sono stati risolti con successo. Per quanto l'architettura di ULLA sia modulare, spesso le operazioni di debug devono essere svolte in modo verticale sull'intero framework.

Va inoltre sottolineato che ULLA è stato sviluppato per operare con una determinata versione del kernel Linux (2.6.23), pertanto alcune strutture dati e metodi, usati nell'implementazione dei moduli del kernel o nella gestione delle interazioni con il sistema operativo, sono un pò superati. L'operazione di porting di ULLA verso l'utilizzo di un kernel Linux più recente, non è un'operazione semplice, dato l'uso di molti costrutti deprecati. Tale operazione necessiterebbe di una rivalutazione dell'intero framework.

Nei paragrafi successivi, presenteremo le strutture dati utilizzate per implementare sia il driver per CalRadio, sia i nuovi attributi supportati. Parleremo inoltre dell'architettura di comunicazione tra i device CalRadio e il framework ULLA.

### 2.9.1 Strutture dati

Nel descrivere l'implementazione del driver per i device CalRadio, e quindi per dare le linee guida per lo sviluppo di un nuovo driver che sia compatibile con ULLA, iniziamo con il descrivere come ULLA identifica i vari parametri.

Ogni parametro di livello fisico o di Link, viene specificato in un file di configurazione di ULLA. Questo file altro non fa che definire questi parametri in modo univoco, dando una breve descrizione del loro significato, descrivendo il tipo di dato che li rappresenta ed infine specificando le restrizioni sull'utilizzo di tali attributi (attributi di sola lettura, oppure completamente modificabili).

Ogni dispositivo hardware espone un'interfaccia, nella quale vengono specificati quali parametri di libello MAC e PHY sono supportati. Questi attributi, altro non sono che numeri o stringhe, come ad esempio la dimensione della congestion window, oppure il nome associato alla rete di comunicazione(ESSID). ULLA definisce dei tipi di dato per immagazzinare tali valori. In particolare definisce il tipo intero come ULLA_TYPE_INTEGER, le stringhe come ULLA_TYPE_STRING e tutti gli altri tipi di dati rientrano nella generica categoria detta ULLA_TYPE_UNDEFINED. Va sottolineato che i dati dell'ultima categoria non vengono gestiti direttamente da ULLA, ma vengono ignorati.

Inoltre, ULLA non dispone del tipo di dato "double", ovvero a virgola mobile. Le ragioni di questa lacuna, possono essere trovati nel fatto che alcuni moduli dell'architettura sono implementati in kernel space e pertanto il tipo "double" non esiste.

Vediamo ora la definizione di un attributo:

```
{ id                : ULLA_LP_ATTRIBUTE_TXCWMIN,
  name              : "txCwMin",
  type              : ULLA_TYPE_INTEGER,
  modifier          : ULLA_MODIFIER_READWRITE,
  unit              : "slot(s)",
  min               : 0,
  max               : 32767,
  value_set         :   "1, 3, 7, 15, 31, 63, 127, 255, 511,
     1023, 2047, 4095, 8191, 16383, 32767",
  description       : "Congestion window bottom limit.", }
```

In questo esempio, viene definito l'attributo txCwMin.
`id` identifica in modo univoco tale attributo all'interno del framework.
`name` è il nome che identifica tale parametro a livello di LU. Pertanto nelle query UQL, sarà questo il nome del parametro specificato.
`type` indica il tipo di dato.
`modifier` indica le restrizioni su quel parametro: permessi di sola lettura o anche di scrittura.
`unit` É una stringa che descrive l'unità di misura usata.
`min,max` Indicano le soglie di minimo e massimo valore consentite per questo parametro.
`value_set` É una stringa che descrive i valori consentiti.
`description` É una stringa che descrive brevemente il significato del parametro.

Le descrizioni dei parametri sono contenuti del file *llaIf.c*. Esistono degli array contenenti queste strutture dati e corrispondo alle "tabelle" indagate da query di tipo Reflection. Difatti, una richiesta sugli attributi di una data tabelle, restituisce esattamente le informazioni contenute in tale struttura dati.

Secondo le potenzialità dei dispositivi CalRadio, e secondo le richieste del progetto ARAGORN, i tipi di dato supportati da ULLA, non bastano.

Ad esempio, CalRadio permette di restituire informazioni circa lo stato del CCA (Clear Channel Assesment) per tutti e tredici i canali wireless in modo contemporaneo. Utilizzando i tipi di dato standard, questa operazione avrebbe richiesto tredici iterazioni, per ottenere la visone completa dello stato dello spettro.

É quindi stata definita una nuovo tipo dati (ulla_data_t), siffatto:

```
typedef struct {
 ulla_integer_t attributeId;
 ulla_integer_t dev_technology;
 ulla_integer_t channels[ULLA_DEV_WIFI];
 ulla_integer_t results[ULLA_DEV_WIFI];
 ulla_timestamp_t timestamps[ULLA_DEV_WIFI];
```

```
  ulla_integer_t error;
} ulla_data_t;
```

Tale struttura dati viene riconosciuta da ULLA, come di tipo ULLA_TYPE_UNDEFINED. Una successiva modifica ai moduli che identificano il tipo dati, consente di trattare in modo corretto questa nuova struttura.

Come già accennato, lo scopo del tipo ulla_data_t è di collezionare in modo contemporaneo dati relativi a più canali. Vediamo nel dettaglio come è stata definita e successivamente, come viene usata.
`attributeId` identifica il parametro per il quale richiedere il valore al device.
`dev_technology` Indica la tecnologie usata: WiFi, ZigBee, Bluetooth...
`channels` É un array che identifica i canali, in base alla tecnologia adottata; grazie a questo campo, si possono selezionare i canali per i quali ricevere dati.
`results` É un array dove, in accordo con la tecnologia scelta, ogni posizione indica un canale. Questo array viene riempito con i dati ricevuti dal device.
`timestamps` É un array come i precedenti e mostra l'istante temporale a cui si riferiscono i dati raccolti.
`error` É un campo che assumerà il valore "ULLA_OK" in caso i dati ricevuti siano corretti, altrimenti un codice d'errore esplicativo del problema.

La struttura dati proposta, permette ad ULLA di essere technology independent, in quanto, specificando la tecnologia usata nel campo `dev_technology`, si può interagire con uno svariato numero di device diversi. Ad esempio, impostando `dev_technology` su WiFi, la struttura dati potrà contenere le statistiche contemporanee dei tredici canali della tecnologia 802.11. Se `dev_technology` viene impostato sullo standard ZigBee, quindi per rilevare dati da una WSN, sarà possibile ricevere statistiche relative a ventisei canali.

### 2.9.2 Parametri per il sensing dello spettro

Grazie alla nuova struttura dati `ulla_data_t` e alle potenzialità dell'hardware utilizzato (*CalRadio*), sono stati definiti dei nuovi parametri, in aggiunta a quelli già a disposizione del framework ULLA.

In particola modo, al fine di eseguire operazioni di sensing sullo spettro radio, sono state definiti i seguenti parametri:

1. **cca_mean** questo parametro permette di rilevare la media del CCA (Clear Channel Assesment) sui canali specificati.

2. **cca_burstiness** questo parametro permette di misurare la durata del burst più lungo di rilevazioni di CCA attive.

3. **avg_busytime** questo parametro permette di rilevare la media del CCA sui canali specificati, usando un algoritmo per il calcolo della media diverso dal quello del punto 1.

4. **var_busytime** questo parametro permette di rilevare la varianza dei dati calcolati al punto 3.

5. **busy_burstiness** questo parametro permette di rilevare la durata del burst, in termini di CCA, più lungo sui canali specificati, in base alle rilevazioni del punto 3.

Per capire meglio il significato dei parametri descritti, spieghiamo innanzitutto che misura viene restituita quando si parla di CCA.

Il CCA (Clear Channel Assesment), quando la radio è posizionata su una determinata frequenza, misura l'energia presente nel canale radio. Se l'energia supera una determinata soglia, il CCA assume valore positivo ed indica che nel canale è presente un segnale, con una potenza superiore alla soglia prefissata.

Senza addentrarci troppo nei dettagli di come *CalRadio* misuri il CCA, in quanto tale meccanismo verrà descritto nel capitolo successivo, spieghiamo come avviene la scansione. La radio scorre i tredici canali dello standard 802.11, rimane "in ascolto" per un tempo prefissato su ognuno di questi e misura il CCA. Il dato risultante, darà una stima media dell'occupazione del canale, durante il periodo di osservazione. La misurazione del burst, altro non è che il più lungo intervallo temporale, nel periodo di osservazione, per cui il CCA viene visto come attivo, ovvero nel quale viene rilevata potenza nel canale radio.

Le misurazioni 3,4, e 5, altro non sono che un diverso tipo di media, dove cioè i la scansione dei canali non viene fatta solo una volta, ma viene ripetuta un numero prestabilito di volte. Avendo a disposizione più dati per ogni canale, è pertanto possibile fornire anche una misurazione della varianza.

I metodi per estrarre questi parametri, verranno spiegati a tempo debito, nella trattazione di *CalRadio*.

Per conoscere tutti i parametri esportati a ULLA si rimanda all'Appendice A.

### 2.9.3 LP per CalRadio

Una volta definite le strutture dai, è stato sviluppato un modulo Link Provider per permettere la raccolta delle statistiche dai dispositivi CalRadio.

Tale modulo (calradio_LLA) implementa l'interfaccia LLA descritta precedentemente, tramite alcune strutture dati a supporto.

In particolar modo vengono implementate le API di registrazione del modulo presso il livello LLA, e le API di gestione dati.

L'associazione tra le API e le funzioni del modulo calradio_LLA, sono cosi definite:

```
static struct ulla_lla_if template_lla_lp_if = {
  GetAttribute : getAttribute,
  CancelUpdate : cancelUpdate,
  RequestUpdate : requestUpdate,
  ExecCmd : execCmd,
  SetAttribute : setAttribute
};
```

### getAttribute

Tale funzione, viene "agganciata" all'API omonima. Permette di richiedere informazioni circa un parametro di livello MAC o PHY.

La sua dichiarazione è:

```
typedef int (*ulla_GetAttribute_t) (const ulla_id_t objId, const
    ulla_id_t classId, const ulla_id_t attributeId,
   ulla_data_pointer_t data, ulla_data_length_t *length,
   ulla_timestamp_t *timestamp);
```

Tale funzione, in primis controlla che l'attributo richiesto, specificato nel parametro `attributeId`, sia tra quelli esposti dal device CalRadio.

Successivamente si fa carico di interrogare il driver del dispositivo CalRadio, mediante una *ioctl*. Una volta ottenuto il dato, la funzione ritorna i risultati, comprensivi di un'indicazione temporale (timestamp).

In caso l'operazione non sia andata a buon fine, la funzione ritornerà un codice d'errore identificativo del problema riscontrato.

Nel caso il parametro richiesto non sia disponibile, tra quelli di CalRadio, la funzione ritornerà subito il codice d'errore "ULLA_NOT_SUPPORTED".

In questo caso, è interessante notare come ULLA utilizzi i codici d'errore, sia per capire se le operazioni sono andare a buon fine, sia come feedback. É il caso dell'allocazione della memoria per contenere il risultato. ULLA, quando invia una richiesta al livello LLA, prealloca un'area di memoria di dimensione minima, atta ad esempio a contenere un valore intero. Qualora questa area di memoria non si sufficiente per immagazzinare i risultati, il modulo LP, restituirà "ULLA_NO_MEMORY". A questo punto il core di ULLA, allocherà uno spazio maggiore. Questo procedimento viene iterato fintantochè la memoria allocata non sarà sufficiente per gestire i risultati.

### setAttribute

Questa funzione, permette di modificare i parametri di CalRadio. La sua dichiarazione è la seguente:

```
typedef int (*ulla_SetAttribute_t) (const ulla_id_t objId, const
    ulla_id_t classId, const ulla_id_t attributeId, const
    ulla_data_pointer_t data, const ulla_data_length_t length,
    ulla_timestamp_t *timestamp);
```

Qualora il parametro che si vuole modificare non sia tra quelli disponibili o si cerchi di modificare un parametro di sola lettura, il modulo LP segnalerà al core che tale operazione non è permessa.

Viceversa, se l'operazione va a buon fine, ovvero se all'attributo specificato da `attributeId` prende il valore indicato nel parametro `data`, la funzione ritorna "ULLA_OK".

### RequestUpdate e CancelUpdate

Queste due funzioni servono per richiedere notifiche al driver CalRadio. La loro definizione è la seguente:

```
typedef int (*ulla_RequestUpdate_t) (const ulla_id_t objId,
    const ulla_id_t classId, const ulla_id_t attributeId,
    ulla_id_t *ruId);
```

```
typedef void (*ulla_CancelUpdate_t) (const ulla_id_t objId,
    const ulla_id_t ruId);
```

Come suggeriscono i nomi, la prima viene usata per iniziare la procedura di notifica, mentre la seconda per arrestarla.

Queste funzioni non erano completamente supportate da ULLA, sia dal punto di vista implementativo che documentativo.

In particolar modo, non era specificato quali attività fossero a carico del core e quali del modulo LP, nella richiesta delle notifiche.

Una volta definito come il meccanismo di notifica debba essere gestito ed implementato, si è dovuto rivisitare in parte il modulo core di ULLA.

Il meccanismo di notifica funziona nel seguente modo: quando un LU manda una richiesta di notifica, il core prende a carico tale richiesta. Supponendo che la sintassi e il contenuto della query siano corretti, il core avvia un thread che si occupa in primo luogo di schedulare la richiesta verso il modulo LLA, e successivamente si pone in ascolto per eventuali segnalazioni da parte dell'LP. Quando riceve un aggiornamento circa lo stato del parametro di guardia, questo applica la regola di confronto specificata nella query iniziale e in caso sia soddisfatta, manda i dati richiesti all'applicazione che

aveva generato la richiesta. In caso l'attributo guarda non soddisfi i vincoli, il thread semplicemente ignora i risultati.

Compito del thread è inoltre quello di tener traccia del numero di notifiche segnalate all'applicazione LU, in modo tale da interrompere la sua esecuzione, nel caso fornisca in numero massimo di notifiche richieste.(vedi 2.7.2 a pagina 53).

Ancora, quando un'applicazione chiede di terminare l'attività di ricezione delle notifiche, il thread in questione si prenderà carico di tale richiesta, richiamando a libello LP la funzione `CancelUpdate`.

Pertanto, il thread svolge un ruolo di supervisione sull'intero meccanismo di notifica. La maggior parte del lavoro viene demandata al livello LU.

Nel caso di CalRadio, quando viene invocata la funzione `RequestUpdate`, viene schedulata una funzione periodica che interroga, ad intervalli regolari di tempo, il parametro richiesto. In sostanza, questa altro non fa che richiamare la funzione `RequesAttribute`.

Al fine di gestire notifiche multiple, è stato dichiarato un array di elementi di notifica:

```
//notification data structures
struct ullaobj_update {
  struct ullaobj* obj;
  ulla_id_t objId;
  ulla_id_t atrid;
  ulla_id_t updid;
  ulla_id_t ruid;
  ulla_id_t classId;
  struct timer_list timer;
  u32 usec;
};

struct notification_list{
  struct ullaobj_update data;
  struct notification_list* next;
  struct notification_list* prev;
};

struct notification_list* nl;
```

`nl` indica la lista delle notifiche.
La struttura `ullaobj_update` tiene traccia dei parametri relativi ad ogni notifica richiesta, come il parametro richiesto, l'identificativo della richiesta, etc. e corrispondono ai parametri della funzione `RequestUpdate`.

Inoltre, vi è il campo `timer` che è una struttura offerta dal Kernel Linux per la schedulazione periodica di funzioni.

Quando viene invocata la funzione `CancelUpdate`, la notifica selezionata viene tolta dalla lista e il suo timer viene fermato.

#### ExecCmd

Questa funzione permette di inviare comandi alla CalRadio. La sua definizione è la seguente:

```
typedef int (*ulla_ExecCmd_t) (const ulla_id_t objId, const
    ulla_id_t classId, const ulla_id_t commandId, const
    ulla_time_t timeout, ulla_timestamp_t *timestamp);
```

L'implementazione di questa funzione è relativamente semplice, in quanto deve solamente invocare una *ioctl* [2] verso CalRadio.

**UllaSample** Si noti come il meccanismo di campionamento **UllaSample**, non sia implementato a livello di LP.

Difatti ULLA, demanda l'implementazione del campionamento, direttamente all' *ULLA-core*. Difatti, il meccanismo di campionamento, per quanto sia simile al procedimento di notifica, è sostanzialmente diverso: nel caso delle notifiche è 'LP che deve, in maniera autonoma, scandire lo spettro continuamente e qualora sia verificata una data condizione, segnalare il risultato al livello LU. Viceversa, il meccanismo di campionamento, chiede un aggiornamento circa lo stato di un parametro (o più parametri) ad istanti di tempo ben precisi e da lui definiti. Pertanto, ULLA implementa nel core un thread specifico per gestire il meccanismo UllaSample, che itererà la chiamata dell'API GetAttribute, secondo le temporizzazioni specificate dalle applicazioni (LU).

### 2.9.4 Modulo per statistiche di link

Secondo le specifiche di ULLA, i driver dei device fisici, devono collezionare statistiche di livello MAC e PHY. In particolare, ULLA definisce due moduli per collezionare le statistiche: **LinkProvider** e **Link**.

Nell'implementazione per CalRadio, questi due moduli assumono rispettivamente i nomi **calradio_LLA_lp** e **calradio_LLA_link**. Il primo di questi è stato presentato nella sezione precedente.

[2] **ioctl** è l'abbreviazione di input/output control. É parte integrante dell'interfaccia di comunicazione tra user-space e kernel-space in un sistema operativo e viene tipicamente usata per permettere ad applicazioni in user space di comunicare con hardware device o componenti del kernel

Il modulo calradio_LLA_link, si occupa di collezionare statistiche per singolo link. Un link è una coppia sorgente destinazione, identificata da due indirizzi MAC. *CalRadio* è in grado di fornire statistiche circa i pacchetti ricevuti nella rete, fornendo per l'appunto queste statistiche e in particolare:

- MAC address di sorgente (SRC) e destinazione (DST).
- dimensione del pacchetto ricevuto (in byte).
- dimensione del payload (in byte).
- rate trasmissivo.
- potenza di ricezione del pacchetto (in dBm).

Pertanto, al fine di collezionare le statistiche circa le comunicazioni rilevate da *CalRadio*, è stata definita una struttura dati come quella riportata in Figura 2.6.

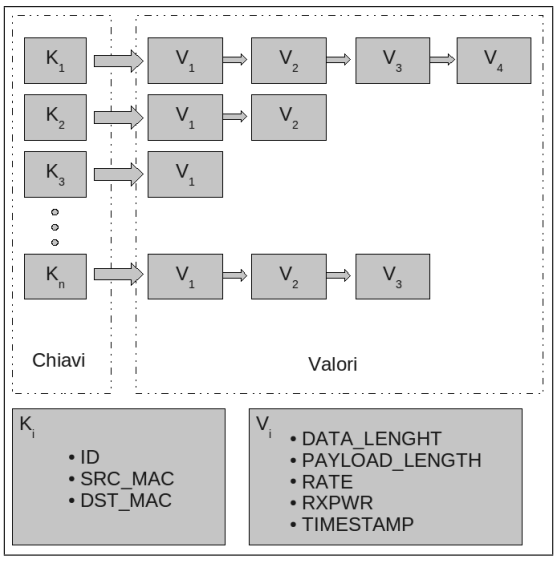



Figura 2.6: Struttura dati definita per immagazzinare le statistiche a livello di singolo link

La struttura dati vuole imitare il comportamento di un DataBase: vi è una lista di chiavi che identificano in modo univoco una comunicazione. Ogni chiave K, è un oggetto composto di tre campi: un identificativo univoco,e una coppia di indirizzi MAC, relativi al nodo sorgente e destinazione della comunicazione rilevata.

Ogni chiave identifica in modo univoco una comunicazione. Quando CalRadio comunica un nuovo dato, si cerca nell'elenco delle chiavi se esiste già una comunicazione tra sorgente e destinazione (riportati dal nuovo dato); se esiste, si crea un nuovo elemento "valore" V, contenente i dati descritti precedentemente, e viene aggiunto in testa alla lista "agganciata" all'elemento chiave.

Nel caso una chiave non esista, questa viene creata, assieme ad una nuova lista, contenente il nuovo elemento valore.

In questo modo, si riescono a collezionare dati relativi a tutte le comunicazioni, mantenendo uno storico, mediante le liste di valori. In questo modo, si possono calcolare altre statistiche relative ad ogni singola comunicazione, come ad esempio il throughput.

Questo meccanismo necessita della gestione sia della lista delle chiavi, sia delle liste valori. n particolare si devono fissare dei limiti per il numero di chiavi, ovvero per il numero di comunicazioni da tracciare. Si deve inoltre fissare un limite massimo di dati storici, per ogni comunicazione.

Se tali limiti non venissero fissati, le liste crescerebbero a dismisura, producendo un degrado dell'intero sistema.

La politica adottata per la conservazione dei dati, si basa sulla freshness dei dati: quando una lista valori raggiunge il limite massimo di elementi, si eliminano quelli meno recenti, ovvero la coda della lista. Per quanto riguarda le chiavi, invece, si controlla il primo elemento valore associato ad ogni chiave e si elimina la chiave (e la lista valori associata) dove il campo timestamp riporta il valore meno recente.

Lo sviluppo dell'esportazione delle statistiche di livello link, non è ancora completamente supportato dal modulo **calradio_LLA_link**. Il suo completamento, viene lasciato come sviluppo futuro.

### 2.9.5 Comunicazione

Abbiamo visto come è strutturato il framework ULLA e come sono stati implementati i moduli a supporto del livello LLA per la SDR *CalRadio*.

Vediamo ora, come si presenta l'architettura di ULLA, arricchita del modulo calradio_LLA e come avviene la comunicazione con il device *CalRadio*.

La premessa, essenziale per comprendere l'architettura di Figura 2.7, è che il device *CalRadio*, non è un dispositivo hardware integrato nel pc, come potrebbe essere una scheda Atheros PCI.

Come verrà specificato meglio nel prossimo capitolo, *CalRadio* è una radio stand-alone, dotata di un proprio sistema operativo. Si potrebbe pensare

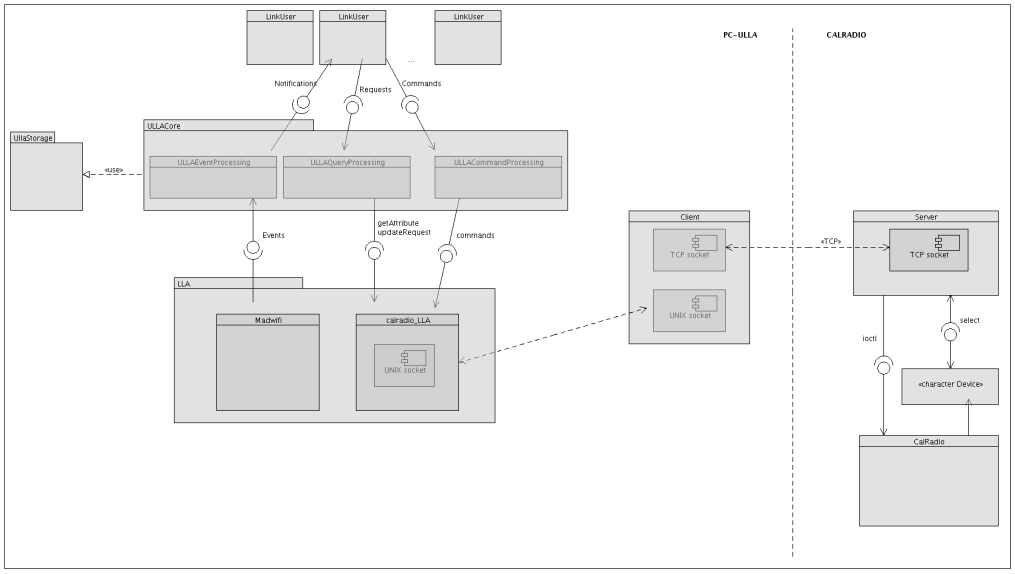



Figura 2.7: Architettura del framework ULLA con il modulo calradio_LLA

quindi di utilizzare il framework ULLA, direttamente su *CalRadio*, ma vedremo che questo non è possibile a causa dell'hardware e del kernel adottati da *CalRadio*. Quello che viene fatto, è collegare mediante un'interfaccia ethernet, *CalRadio* con il pc che fa hosting al framework ULLA.

Viene quindi a delinearsi una comunicazione clien-server tra ULLA e *CalRadio*, per lo scambio di messaggi.

Tutto questo non limita le potenzialità di *CalRadio*, anzi da un certo punto di vista le estende: basti pensare alla possibilità di posizionare il device ovunque si voglia, grazie alle dimensioni contenute del dispositivo (molto inferiori a quelle di un pc). Un altro vantaggio offerto da quest'architettura, è la possibilità di utilizzare *CalRadio* su qualunque piattaforma, con il solo limite che questa abbia installato il framework ULLA. Invece, se *CalRadio* fosse un dispositivo integrato in un pc, bisognerebbe garantire la compatibilità con l'hardware e i driver del dispositivo. Ancora si può vedere che *CalRadio* non può essere collegata direttamente al pc con ULLA ma allo stesso modo potrebbe essere collegata ad un qualunque pc, in qualunque luogo; si pensi ad esempio ad una rete di medie dimensioni: se calradio fosse collegata direttamente al pc con ULLA servirebbero n-pc per monitorare n-punti di questa rete. Viceversa, con questa architettura, basterà un unico pc con installato ULLA e n-device *CalRadio* disseminati nella rete, che comunicano tutti con il medesimo nodo. In questa maniera, si centralizza il framework e si distribuiscono i dispositivi osservatori, semplificando così

l'aggregazione delle informazioni, senza contare la semplificazione in termini di gestione.

Pertanto quest'architettura, accresce la flessibilità e la modularità dell'intero environment.

Nell'ambito della tesi, ULLA era in esecuzione su un pc di fascia media, con sistema operativo Linux Gentoo, con kernel alla versione 2.6.23.17. Il fatto di utilizzare un pc per l'esecuzione del framework ULLA, è dovuto in parte alle limitazioni dell'hardware *CalRadio* e in parte per poter supportare in modo più efficiente lo storage e l'elaborazione delle statistiche rilevate, tramite un motore cognitivo, come può essere il CRM di ARAGORN. Difatti, il salvataggio delle statistiche può avvenire avvalendosi di un DataBase, come ad esempio MySQL, il quale necessita sia di spazio fisico, sia di performance elaborative.

Premesso tutto questo, vediamo nel dettaglio come funziona la comunicazione tra il modulo di ULLA e il device *CalRadio*.
Una volta che una richiesta da parte di un'applicazione arriva al livello LLA dell'architettura, tale modulo si occupa di specializzare la richiesta generica al device astratto corretto, nel nostro caso al LP calradio_LLA. A sua volta, il modulo decide se inoltrare la richiesta verso il modulo calradio_LLA_lp oppure verso calradio_LLA_link. Per semplicità nell'architettura di Figura 2.7, si parla del solo modulo calradio_LLA.

Quindi il modulo prende a carico la richiesta, e l'inoltra, in forma di *ioctl* all'applicazione client. Questa, semplicemente creerà un pacchetto UDP e lo invierà ad una corrispondente applicazione server su *CalRadio* la quale a sua volta decodificherà il pacchetto e invierà un'*ioctl* verso il kernel di *CalRadio* per eseguire l'operazione richiesta. Una volta che i dati richiesti sono disponibili, questi verranno inviati tramite l'applicazione server al client (lato ULLA), che elaborerà i risultati ottenuti. Per quanto riguarda l'applicazione server e la gestione da parte di *CalRadio* delle *ioctl*, si rimanda al capitolo successivo.

L'applicazione client, che opera in user space, deve comunicare con il modulo del kernel calradio_LLA per il trasferimento dei dati. Questa comunicazione tra i due software, avviene con l'ausilio di un character device [3] per la ricezione dei risultati, mentre per l'invio delle richieste, vengono viene usato il meccanismo di *ioctl*.

L'implementazione, prevede che l'applicazione client debba solamente occuparsi dell'invio e della ricezione dei dati. Viceversa, il modulo calra-

[3]Un character device è un file di Linux che permette la comunicazione tra un processi. Quello I device in Linux, sono distinti in tre categorie: character, network e block.Per maggiori informazioni sull'implementazione e l'utilizzo dei device e della comunicazione tra processi nel sistema Linux, si faccia riferimento a [21]

dio_LLA, deve tener conto delle temporizzazioni necessarie per cui *CalRadio* elabori le richieste e restituisca i risultati...

Visto che il modulo di ULLA opera in kernel space, è impensabile che questo resti in attesa di una risposta finchè *CalRadio* non ha elaborato i dati. Quello che viene fatto nella pratica è di schedulare l'esecuzione del modulo, dopo un istante temporale pari al tempo stimato per l'elaborazione dei dati da parte di *CalRadio* e la comunicazione. Per rendere questa tecnica più consistente, il meccanismo funziona nel seguente modo: il modulo viene schedulato dopo il tempo designato; se il dato richiesto è disponibile presso il character device, questo viene letto. Altrimenti, il modulo viene schedulato altre n volte ad istanti temporali ravvicinati. Questo viene fatto per essere tolleranti ad eventuali ritardi nella comunicazione client-server. Nel caso che, dopo le n-retry i risultati non siano ancora disponibili, il modulo segnala un errore e smette di interrogare il character device.

### 2.9.6 Lavori futuri

Lo sviluppo del modulo calradio_LLA può dirsi completata per l'esportazione di statistiche di livello MAC e PHY.

Grazie alla struttura dati per l'esportazione delle statiche, può facilmente essere preso come modello per lo sviluppo di moduli per altri device o per altre tecnologie. Quel che resta da sistemare, è lo sviluppo del modulo che si occupa delle statistiche dei link, implementando le funzionalità per la ricerca e la popolazione della struttura dati presentata precedentemente. Vanno inoltre definite delle funzioni per ricavare delle misure dai dati raccolti, come ad esempio il throughput di un link, oppure statistiche circa quale sia il link che utilizza maggiormente il canale, etc.

Attualmente, le statistiche relative ai link, sono gestite solo fino all'applicazione cliente. Ovvero, *CalRadio*, qualora riceve un pacchetto, ne estrapola le caratteristiche richieste ed invia tale informazioni ad un client (lato ULLA), il quale si serve di un apposito character device per immagazzinare i risultati.

Per quanto riguarda l'architettura in generale, questa si è dimostrata stabile durante i test eseguiti. La stabilità è un requisito essenziale, sopratutto in un sistema così complesso, composto da diversi "point of failure". Difatti, possono presentarsi problemi nella *CalRadio* come nella comunicazione client-server o ancora applicazioni (LU). L'importante è che il core di ULLA rimanga stabile, anche in presenza di guasti nei sistemi correlati. Se questo non fosse, un crash del framework provocherebbe un arresto dell'intera macchina che fa hosting di ULLA in seguito ad un KERNEL PANIC.

Inoltre, per poter gestire questi errori, è importante inserire dei meccanismi di notifica degli errori. Ad esempio, le statistiche provenienti da *CalRadio*, contengono un campo apposito per la segnalazione degli errori. In questo modo si riesce a capire quando il device non funziona correttamente, oppure se la comunicazione è interrotta. Risulta quindi importante, nelle applicazioni future, continuare in quest'ottica per mantenere il sistema il più resistivo possibile agli errori delle applicazioni correlate.

### Kernel Space vs User Space

I moduli appena descritti sono implementati su un sistema Linux. Questi quindi possono essere visti come un set di applicazioni che cooperano tra loro.

Uno dei concetti fondamentali su cui si basa l'architettura dei sistemi Unix è quello della distinzione fra il cosiddetto user space, che contraddistingue l'ambiente in cui vengono eseguiti i programmi, e il kernel space, che è l'ambiente in cui viene eseguito il kernel.

Ogni programma vede se stesso come se avesse la piena disponibilità della CPU e della memoria ed è completamente ignaro del fatto che altri programmi possono essere messi in esecuzione dal kernel, salvo nel caso utilizzi i meccanismi di comunicazione previsti dall'architettura.

Grazie a questa separazione non è possibile che singolo programma disturbi l'esecuzione di un altro programma o del sistema e questo è il principale motivo della stabilità di un sistema unix-like nei confronti di altri sistemi in cui i processi non hanno di questi limiti, o che vengono eseguiti al livello del kernel.

Pertanto deve essere chiaro a chi programma in Unix che l'accesso diretto all'hardware non può avvenire se non all'interno del kernel; al di fuori del kernel, il programmatore deve usare le opportune interfacce che quest'ultimo espone verso lo user space.

In ULLA il software che opera in user space è costituito dalle applicazioni LU e dall'applicazione client per la comunicazione. Oltre a questi, vi sono i thread generati per gestione delle statistiche e dei campionamenti. Il core di ULLA come i moduli LP, operano invece in spazio kernel.

In questo capitolo abbiamo mostrato l'architettura del framework ULLA sia nel contesto più ampio del progetto ARAGORN sia come applicazione stand alone nell'ambito della tesi. Per ULLA è stato sviluppato un modulo per estrapolare statistiche di livello MAC e PHY dai device *CalRadio*.

Inoltre sono state definite le strutture dati a supporto di tali moduli e utilizzabili per lo sviluppo di altri LP. Queste strutture dati, sono state modellate per essere utilizzabili anche con tecnologie diversi dallo standar 802.11 per le reti WiFi.

Sono inoltre stati definiti dei parametri avanzati, rispetto a quelli standard usati da ULLA, in modo da supportare in sensing dello spettro radio, esportando contemporaneamente statistiche riguardanti il CCA per tredici canali. Per una lista completa dei parametri disponibili, si veda l'Appendice A.

Vedremo nel capitolo successivo, la parte mancante della descrizione dell'architettura in Figura 2.7, ovvero parleremo dei dispositivi *CalRadio*.

# Capitolo 3

# CalRadio

*CalRadio* è un device sviluppato alla UCSD (UC San Diego) nel progetto Calit2 [22].

La piattaforma è dotata di un transceiver che opera secondo lo standard 802.11b e che permette di implementare via software il livello MAC. Questa proprieà rende a tutti gli effetti *CalRadio* una SDR (software defined radio).

*CalRadio* è dotata di un chip DSP della Texas Instruments, il quale viene programmato in linguaggio C, al fine di implementare il livello MAC.

La piattaforma è inoltre dotata di un processore ARM che permette l'esecuzione di un sistema operativo Linux.

*CalRadio* è quindi una dispositivo stand alone, che implementa la tecnologia wireless 802.11b. Nella tesi il dispositivo non viene utilizzato per operare in maniera autonoma, ma viene collegato al framework ULLA per estrarre statistiche di livello MAC e PHY.

## 3.1 Specifiche tecniche

*CalRadio* è composta di:

- TI TMS320VC5471 ARM+ DSP (ARM7TDMI + 5410 style DSP)
- 8-bit software controlled internal status LEDs
- Power On/Off key, 2-GP keys, and reset control key
- Power management
- Operation from WallWart (Plug Transformer) (9 - 14 volt) via power connector
- Size 4 x 6 x 3

- Four external LED indicators
- External reset pushbuttons
- RJ45 Ethernet jack
- Dismountable antennas - SMA connector
- 10/100 Ethernet Phy
- 4 channel ADC 100KSPS (TLV2541)
- 4 channel DAC 100KSPS (TLV5636)
- Jumpers on all power connections to measure current draw
- Serial port connector (external) Mini-DIN
- EMIF pins to mezzanine connector + Vcc
- 2 Mbytes static RAM - ARM side, 512KB - DSP
- 16 Mbytes SDRAM - ARM side
- 4 Mbytes Flash ROM - ARM side
- DSP Expansion connector

## Software

Il software caricato e gestito dal processore ARM, è composto di due elementi principali: is sistema operativo UCLinux e un bootloader.

Il bootloader di *CalRadio* è chiamato "crload". Quando il processore viene avviato, o in seguito ad un reset, esegue il bootloader, il quale si occupa di inizializzare tutti i registri, caricare la mappa della memoria, impostare la modalità operativa, avviare il sistema operativo ed infine, presenta un menu dal quale possono essere avviate operazioni di debug, caricamento, avvio del sistema Linux, etc.

Il bootloader viene eseguito dal codice Assembly, descritto dal file head_c5471.ASM e le operazioni svolte sono:

1. Inserire le tabelle dei vettori in memoria
2. Impostare vari parametri della memoria
3. Impostare la frequenza del clock

4. Abilitare il chip UART per l'I/O da console
5. Eseguire un check della memoria
6. Inizializzare il DSP
7. Eseguire il bootstrap del DSP
8. Spostare il bootloader dalla flash alla RAM
9. Avviare la GUI dei comandi

L'inizializzazione del DSP consiste dei seguenti passi:

1. Impostare tutti i registri del DSP alla configurazione di default
2. Impostare la mailbox per la comunicazione con l'ARM
3. Caricare i registri per i chip HFA3863 e MAX2821
4. Istanziare l'handler delle interrupt
5. Inizializzare il ciclo principale di gestione del DSP

All'arrivo di una interrupt che segnala la ricezione di un pacchetto (generata dal transceiver), il DSP:

1. Imposta il DMA per leggere il pacchetto
2. Calcola il CRC
3. Manda un ACK
4. Sposta il pacchetto in un buffer, per trasferirlo all'ARM

Il ciclo principale di gestione del DSP esegue le seguenti operazioni:

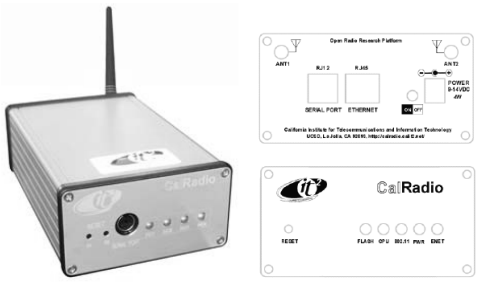
1. Controlla continuamente lo stato della mailbox (poll) e qualora sia presente un pacchetto proveniente dall'ARM. In caso sia presente, gestisce la spedizione.
2. In caso sia presente un messaggio, ricevuto dal transceiver, diretto all'ARM, questo chiama invoca una interrupt verso l'ARM.
3. *Nel caso il DSP debba mostrare dei messaggi dei debug, viene segnalata all'ARM la presenza di questi.*[1]

[1] Questa operazione è opzionale.

4. *Nel caso siano presenti delle statistiche di livello MAC o PHY, viene segnalata all'ARM la presenza di queste.*[2]

## 3.2 Architettura

L'hardware che compone *CalRadio* consiste di un processore ARM interconnesso mediante una memoria condivisa ad un DSP che implementa via software il livello MAC 802.11b. Nelle Figure 3.2 e 3.1, viene mostrata *CalRadio* e uno schema della sua architettura.




(a) *CalRadio* (b) *CalRadio*

Figura 3.1: *CalRadio*

Nel pannello frontale della radio, si vedono i led programmabili, atti a mostrare le fasi di comunicazione, o utili per segnalare l'esecuzione di determinate operazioni. Nella parte posteriore, vi è un tap RJ45 per il collegamento ethernet, l'adattatore per l'antenna e l'ingresso per l'alimentazione, oltre ad una porta di comunicazione seriale.

L'architettura mostrata in Figura 3.2, mostra l'interazione tra il microprocessore ARM, il DSP e il trasmettitore RF.

Vediamo ora in dettaglio queste componenti.

### 3.2.1 ARM

Il microprocessore ARM, prodotto dalla Texas Instruments (modello TI TMS320VC5471), consente l'esecuzione del sistema operativo UCLinux [23] con il kernel 2.4.38. Questo sistema operativo è stato concepito per l'utilizzo su dispositivi embedded, essendo strutturato con un kernel "leggero".

[2]Queste operazioni sono opzionali e vengono eseguite solo se il meccanismo di raccolta delle statistiche è stato attivato.

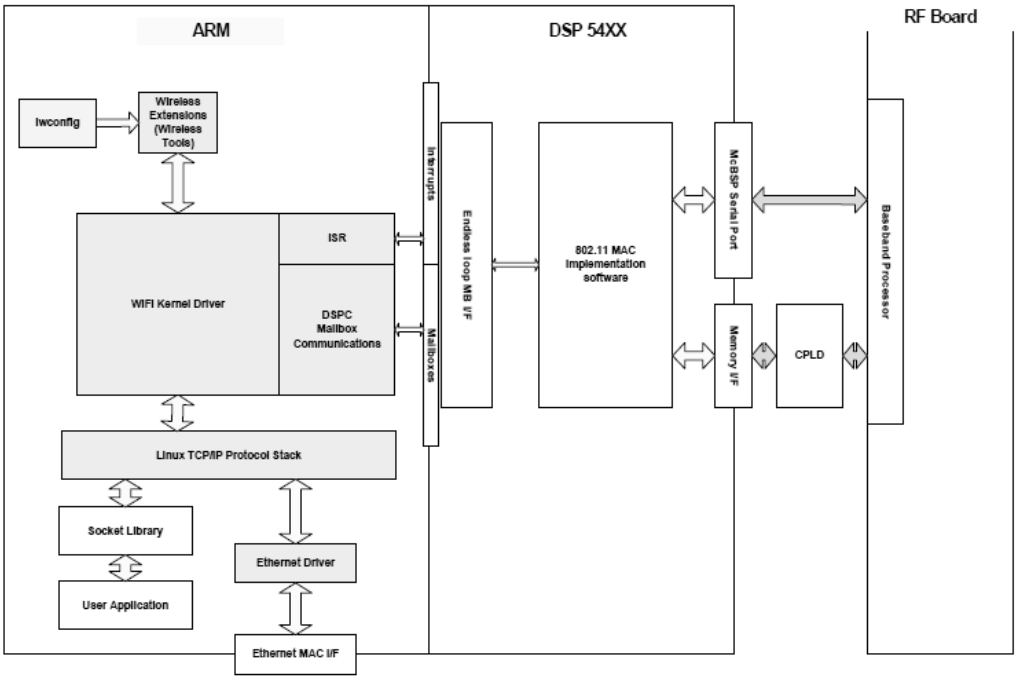



Figura 3.2: Architettura della board di *CalRadio*

Abbinato a UCLinux, troviamo la suite BusyBox[3], che permette di interagire con la maggior parte delle funzionalità di un sistema operativo Linux standard. In particolar modo vengono offerte le funzionalità delle Wireless Extensions e i relativi Wireless tools, oltre ai comandi standard per la configurazione della comunicazione ethernet.

La comunicazione tra ARM e DSP, avviene nelle due direzioni mediante la gestione delle **interrupt** e l'utilizzo di una **mailbox**. Come si può vedere in Figura 3.2, oltre al gestore delle interrupt, vi è una memoria condivisa (mailbox) tra il microprocessore e il DSP. Le operazioni di I/O su quest'area di memoria, permettono lo scambio dei dati tra i due dispositivi, previa opportuna conversione dati. La segnalazione della presenza dati sulla mailbox, viene gestita tramite le interrupt.

Il compito del processore ARM, nell'integrazione con il framework ULLA è di raccogliere le statistiche ottenute tramite il DSP ed inviarle ad ULLA.

Per fare questo è stato creato un character device, che si occupa di im-

[3]BusyBox è un software libero, rilasciato sotto la GNU General Public License, che combina diverse applicazioni standard di Unix in un piccolo eseguibile. BusyBox può fornire la maggior parte delle utility menzionate nel Single Unix Specification ed in aggiunta altre che un utente si aspetterebbe di vedere su un sistema GNU/Linux.[24]

magazzinare le informazioni, mentre un modulo server, si occuperà di inoltrare le richieste provenienti da ULLA verso il DSP e successivamente, di restituire i dati raccolti.

In particolar modo, è stato modificato il modulo di *CalRadio* che si occupa della gestione delle *ioctl* per i comandi di manipolazione delle interfacce di rete, in modo da gestire un nuovo insieme di comandi, detti `ULLA_ATTRIB_IOCTL`. Questi verranno descritti meglio nei paragrafi successivi. Quello che si vuole sottolineare ora, è come l'ARM gestisce questi eventi: all'arrivo di una chiamata *ioctl*, proveniente dalle Wireless Extensions, piuttosto che da ULLA l'ARM inoltra tale richiesta verso il DSP, scrivendo nel buffer condiviso. Quando poi, il DSP ha elaborato la richiesta, segnalerà mediante una interrupt software che l'evento è stato processato ed eventualmente che nella mailbox sono disponibili dei dati.

Un'ultima considerazione doverosa sul funzionamento dell'ARM è che questo opera con una tecnologia a 32 bit, viceversa il DSP è a 16 bit, pertanto la comunicazione tramite la mailbox, deve avvenire tramite una conversione del formato dati e, sopratutto, l'ARM non può utilizzare (verso il DSP) tipi di dato superiori ai 16 bit.

### 3.2.2 DSP

Il DSP (Digital Signal Processor), è un processore integrato nel chip TI TMS320VC5471. Questo è programmabile in linguaggio C e permette di implementare via software il livello MAC del protocollo di comunicazione 802.11b.

Questo modulo di occupa dell'invio e ricezione di pacchetti 802.11, della codifica, dell'interpretazione del preambolo in fase di ricezione, del calcolo del CRC, etc. Insomma, il DSP implementa tutte le funzionalità di livello MAC. Unica lacuna dell'implementazione corrente, è la mancanza del meccanismo di beaconing. Più in dettaglio, il DSP riesce a riconosce i beacon, ma non svolge l'operazione di beaconing.

Per rendere CalRadio una radio cognitiva a tutti gli effetti, dal DSP si devono estrarre le informazioni di livello MAC circa la comunicazione. Per fare questo,sono stati creati dei punti di osservazione nel codice, che raccolgono statistiche. Le statistiche esportate verranno specificate in seguito. Spieghiamo solo come avviene il loro collezionamento: quando arriva un messaggio al DPS, proveniente dal modulo RF, nelle varie fasi di decodifica e riconoscimento del pacchetto, vengono aggiornate le relative statistiche.

Il DSP, mette a disposizione una porzione di memoria, mostrata in Figura 3.3. Questa viene usata come mailbox nella comunicazione con l'ARM e per utilizzo interno del DSP, come ad esempio per collezionare le statistiche.

La comunicazione con l'ARM avviene mediante la combinazione di interrupt

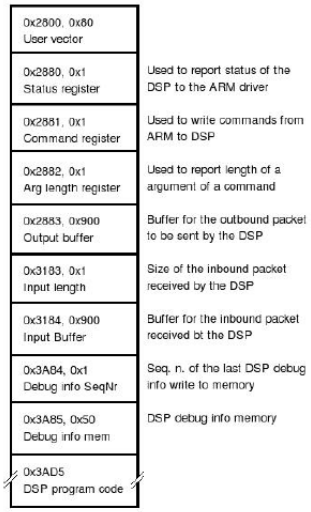



Figura 3.3: Mappa della memoria del DSP

e mailbox. Invece, la comunicazione con il chip RF, avviene con l'ausilio di un registro, sul quale vengono inseriti i messaggi da inviare e, viceversa, quelli ricevuti.

### 3.2.3 RF

Il chip RF, è composto dal chip Intersil HFA3863 e dal transceiver Maxim MAX2820, operante a 2.4GHz, progettato per la tecnologia 802.11b. Il chip Intersil è un processore in banda base per produrre sequenze DSSS (Direct Sequence Spread Spectrum), con modulazioni DBPSK,DQPSK e CCK per ottenere rate trasmissivi di 1, 2, 5 e 11MBps. In Figura 3.4, vediamo lo schema di funzionamento del transceiver. Le caratteristiche tecniche dei chip HFA3863 e MAX2820, si trovano nei datasheet [25, 26]. Per completezza, riportiamo lo spettro del segnale in uscita dal transceiver e in Figura 3.2.3 il filtro in ricezione.

Da questi grafici si può vedere come il segnale trasmesso, abbia una banda di circa 22MHz che il ricevitore applichi un filtro passa banda della stessa ampiezza. Questo fenomeno fa si che, quando la radio è posizionata su un determinato canale, riesca a "vedere" le comunicazioni anche delle frequenze adiacenti.

Questo fenomeno, investigato da Fuxjager et Al. in [27]. In questo articolo, viene studiato come anche nei cosiddetti "non-overlapping channels" vi

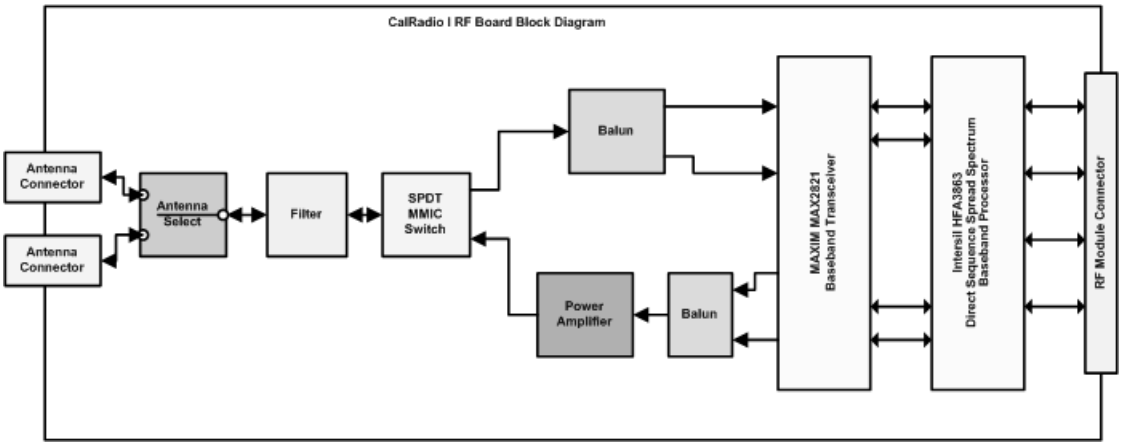
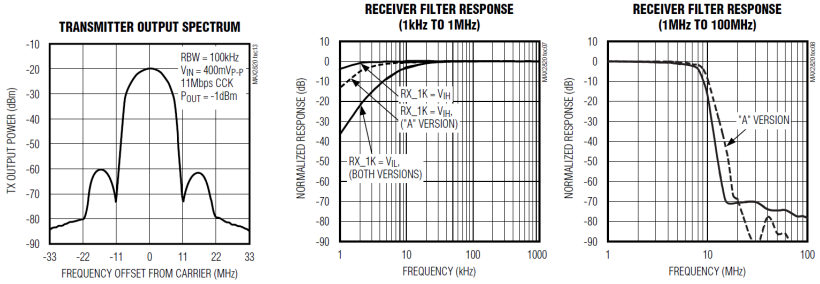



Figura 3.4: Transceiver




(a) Spettro del segnale in uscita

(b) Risposta del filtro in ricezione

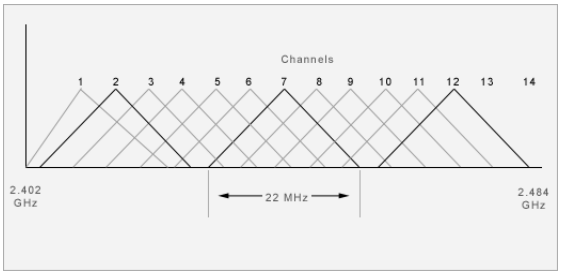



Figura 3.5: IEEE 802.11b Overlapping channels

sia interferenza. In particolar modo, si conclude che l'interferenza sui canali "non-overlapping" (1,6,11) è minima nel caso i dispositivi che trasmettono su queste frequenze siano distanti, mentre l'interferenza diventa più significativa nelle reti a mesh, ovvero con dispositivi disposti a distanza di qualche metro. É anche il caso delle reti domestiche, dove più dispositivi, ravvicinati trasmettono.

Questo fenomeno è stato visto anche nell'utilizzo di *CalRadio*. Ad esempio, mettendo in comunicazione wireless due dispositivi *CalRadio* operanti su frequenze diverse e a distanza ravvicinata (meno di un metro), la comunicazione avveniva comunque in modo corretto. Nel capitolo successivo, verranno presentati alcuni risultati, dove si potrà vedere ancora una volta il fenomeno dell'overlapping.

Il transceiver permette di esportare verso il DSP diverse statistiche di livello PHY, tra cui:

- La potenza del segnale ricevuto
- Il rapporto segnale rumore (SNR)
- L'RSSI (Received Signal Strength Indication)
- La misura del CCA (Clear Channel Assesment)

In particolar modo, quest'ultimo parametro risulta particolarmente utile per indagare l'occupazione dello spettro.

## 3.3 CCA

La maggior parte degli standard per WLAN e WPAN, adottano CSMA (Carrier Sense Multiple Access) come protocollo *de-facto* per l'accesso al mezzo trasmissivo.

CSMA prevede che i nodi della rete rilevino la presenza di altre trasmissioni nel canale, prima di trasmettere. Se viene rilevata una trasmissione, il nodo ritarda la trasmissione fintantochè non trova il canale libero.

Il meccanismo CSMA è implementato dal livello MAC del protocollo e si basa sull'operazione di sensing del canale che avviene al livello PHY, mediante la lettura del CCA.

Lo standard 802.11 prevede tre diversi meccanismi per la rilevazione del CCA:

**CCA Mode 1** Viene rilevata la sola presenza di energia nel mezzo trasmissivo. La misura del CCA, darà esito positivo (canale occupato) qualora l'energia rilevata superi una certa soglia.

**CCA Mode 2** Viene rilevata la presenza di un segnale DSSS. La misura del CCA sarà positiva in presenza di tale segnale, ignorando la soglia di energia.

**CCA Mode 3** É una combinazione degli algoritmi precedenti: il mezzo sarà rilevato come occupato, se viene rilevata la presenza di un segnale DSSS la cui energia supera una determinata soglia.

Il chip HFA3863, può essere programmato per rilevare il CCA in funzione dell'RSSI, ovvero dell'energia rilevata nel canale, oppure in funzione di CS1 o SQ1.

Il CS1 diviene attivo qualora venga rilevato un segnale abbastanza potente da supportare una comunicazione a 11Mbps, ma non viene attivato se ad esempio la potenza è sufficiente per trasportare un segnale a 1 o 2 Mbps.

Il segnale SQ1, invece diviene attivo quando viene rilevata un segnale a banda larga, mediante un'operazione di correlazione sui picchi del segnale. Quest'ultimo modo di misurare il CCA, è particolarmente adatto a rilevare i soli segnali 802.11. Tuttavia, questa misura può non essere sufficiente per determinare se il canale sia effettivamente in uno stato di Idle.

*CalRadio* permette un uso combinato di questi metodi di rilevazione, semplicemente cambiando le impostazioni nei registri del chip HFA3863, secondo quanto riportato in Tabella 3.1.
L'ultima impostazione, permette di ottenere misure del CCA in modo asincrono o sincrono. Nel caso sincrono, il CCA viene aggiornato una volta per simbolo e sono valide le letture ogni 15,8 $\mu$s o 18,7 $\mu$s.

per quanto riguarda la soglia dell'ED (Energy Detection), solitamente la soglia è impostata nel range [-70dBm:-80dBm]. Per maggiori dettagli circa la lettura del CCA, si veda il datasheet del chip HFA3863 [25].

## 3.4 Contributo

*CalRadio* è in grado di eseguire le comuni operazioni di comunicazione wireless, sia in modalità ad-hoc, sia infra-strutturata.

Grazie all'implementazione software del livello MAC, è possibile esportare statistiche di livello MAC e PHY, mantenendo il normale funzionamento della radio. Questo vuol dire che *CalRadio* può essere usata come mezzo trasmissivo anche durante le operazioni di rilevamento delle statistiche, ad eccezion fatta per la rilevazione del CCA per un periodo di tempo arbitrario.

| REGISTRO | BIT | SIGNIFICATO |
|---|---|---|
| CR1 (0X02) | 2 | CCA attivo alto/basso |
| CR9 (0X12) | 7 | Campionamento:<br>0 = 18,7 $\mu$s<br>1 = 15,8 $\mu$s |
| CR9 (0X12) | 6:5 | 00 => CCA = ED<br>01 => CCA = (CS1 OR SQ1)<br>10 => CCA = (ED AND (CS1 OR SQ1))<br>10 => CCA = (ED OR CS1 OR SQ1) |
| CR11 (0X16) | 3 | CCA mode<br>0 = normale: CCA risponde in modo istantaneo ad un cambiamento di ED,CS1 o SQ1<br>1 = campionato: Il CCA viene aggiornato ogni 20 $\mu$s e sarà valido al per il tempo espresso nel registro CR9 |

Tabella 3.1: Configurazione dei registri per la misura del CCA
ED = Energy Detection

Questo limite è imposto dall'hardware, per cui fintantochè il modulo RF sta misurando il CCA, la radio è occupata in questa operazione e vengono impedite le normali operazioni di invio e ricezione dei pacchetti.

Vediamo ora in dettaglio quali statistiche vengono esportate e in che modo.

### 3.4.1 Esportazione di statistiche verso il processore ARM

Come detto in precedenza, all'interno del DSP vengono collezionati dei dati statistici ogni qual volta viene processato un pacchetto, sia che questo sia in fase di trasmissione che di ricezione.

Per poter mostrare questi valori all'esterno del DSP, quindi sia tramite i Wireless Tools, sia mediante l'interazione con ULLA bisogna trasferire l'informazione dal DSP al processore ARM. Questo poichè il DSP è usato per le sole operazioni di livello MAC e non può interagire direttamente con il sistema UCLinux (ovvero non riesce a fornire nessun output diretto).

L'unico modo per far comunicare ARM e DSP è attraverso l'area di memoria da essi condivisa (mailbox).

L'esportazione delle statistiche avviene sia in modalità asincrona dal pun-

to di vista dell'ARM, in quanto gli viene segnalata la presenza di nuove informazioni nella mailbox, mediante un'interrupt software.

Su *CalRadio* possono essere esportati due tipi di statistiche, in accordo con quanto definito da ULLA:

- statistiche a livello MAC e PHY (LinkProvider)
- statistiche per singolo Link

Le prime vengono sempre collezionate sul canale in cui si trova la radio, ma vengono esportate solo su esplicita richiesta proveniente dall'ARM. In altre parole, il DSP continua a raccogliere dati e ad immagazzinarli nella propria memoria interna e solo quando arriva una richiesta "dall'esterno" per questi valori, il DSP copia il dato richiesto nella mailbox e lancia un'interrupt software di segnalazione.

Per quanto riguarda le statiche sul singolo link, queste possono essere abilitate o meno. Nel caso lo siano, ogni qualvolta il DSP riesce a decodificare correttamente un pacchetto, questo scrive subito nella mailbox i valori estrapolati e segnala tale evento all'ARM.

Nel sistema operativo in esecuzione sull'ARM, sarà caricato un modulo opportuno per gestire tali statiche.

Dal lato del sistema operativo, i dati vengono immagazzinati con l'ausilio di due character device, uno per le statistiche del primo tipo e uno per quelle relative ai singoli link.

### 3.4.2 Raccolta di statistiche

In Tabella 3.2, vengono riportate le statistiche collezionate.

In particola modo, poniamo il focus, circa le statistiche sulla misura del CCA. Diversamente dalle altre informazioni, la raccolta di queste avviene in modo esclusivo.

Quando viene eseguita la scansione del CCA, l'hardware (RF) viene completamente impegnato da questa operazione, pertanto *CalRadio* non riuscirà ad eseguire nessun altra operazione di invio/ricezione dati tramite la connessione 802.11.

Viceversa, le altre statistiche vengono collezionate in contemporanea con l'elaborazione dei pacchetti in invio/ricezione.

### 3.4.3 Media del CCA

Con questo parametro, si vuole valutare l'andamento medio dell'occupazione dello spettro, per un dato periodo di tempo.

| **STATISTICHE DI LIVELLO MAC** | |
|---|---|
| **STATISTICHE RX** | **STATISTICHE TX** |
| # totale pacchetti ricevuti | # totale pacchetti inviati |
| # pacchetti ricevuti con preambolo corretto | # pacchetti data inviati |
| # pacchetti ricevuti con preambolo corretto e CRC corretto | # pacchetti data inviati e seguiti da un ack |
| # pacchetti scartati | # pacchetti data inviati e seguiti da un busy channel |
| # pacchetti beacon ricevuti | |
| # pacchetti rts ricevuti | # pacchetti rts inviati |
| # pacchetti cts ricevuti | # pacchetti cts inviati |
| # pacchetti ack ricevuti | # pacchetti ack inviati |
| # pacchetti unicast ricevuti | # pacchetti unicast inviati |
| # pacchetti broadcast ricevuti | # pacchetti broadcast inviati |
| **STATISITCHE DI LIVELLO PHY** | |
| rate del singolo pacchetto | |
| dimensione del pacchetto | |
| RSSI medio | |
| media del CCA | |
| durata del burst massimo del CCA | |
| media del CCA con scansione ripetuta | |
| varianza della media del CCA | |
| durata del burst massimo del CCA (nel caso di scansione ripetuta) | |

Tabella 3.2: Statistiche di livello MAC e PHY collezionate da *CalRadio*

*CalRadio* consente di decidere se analizzare l'intero spettro, ovvero tutti e tredici i canali 802.11, oppure solo un sottoinsieme. Una volta definite le frequenze da analizzare, la scansione avviene nel seguente modo:

1. La radio viene sintonizzata sul primo canale (2412 MHz)
2. per un tempo (`t_period_ms`), vengono eseguite delle letture del CCA (ogni 0.2$\mu$s)
3. trascorsi `t_period_ms` millisecondi, ci si sposta sul canale successivo e poi si reitera al punto 2.

Ogni qualvolta il CCA viene rilevato come attivo (su un canale), viene incrementata una variabile contatore; al termine della scansione, il rapporto

tra il numero di letture attive di CCA e il numero di letture effettuate, ci dice la percentuale di tempo nella quale è stata rilevata energia sul canale.

### 3.4.4 Durata del burst massimo del CCA

Questo parametro viene calcolato contemporaneamente al precedente e tiene conto del periodo più lungo per cui viene rilevata potenza nel canale di comunicazione, durante il periodo di scansione. Questo parametro, ci dà un'idea dell'occupazione massima, in termini di tempo, del canale nel periodo di scansione.

### 3.4.5 Media del CCA con scansione ripetuta

Il calcolo della media presentato precedentemente, soffre del fatto che per scandire tredici canali, ogniuno per n-millisecondi, ci si sposta da un canale al successivo in modo sequenziale. Pertanto, i dati rilevati sul primo canale, saranno $12 \times n$ millisecondi più "vecchi" dei dati rilevati sul tredicesimo canale.

Per limitare questo comportamento è stata introdotta un nuovo tipo ti media, che scandisce i canali un numero prefissato di volte. Su tutte le scansioni eseguite, viene poi restituita la media e la **varianza** di tali scansioni e la **durata del burst massimo**.

### 3.4.6 RSSI medio

La ricezione di un pacchetto 802.11, permette di ottenere informazioni circa l'RSSI.

Il parametro che viene misurato, è la media dell'RSSI rilevato sul canale. La funzione di media, è una media media mobile tarata su due parametri $\alpha$ e $\beta$, che rispettivamente pesano i nuovi valori e i dati meno recenti.

$$\mu_t = \alpha \times x_t + \beta \times \mu_{t-1}, \text{ dove } \beta = (1 - \alpha)$$

L'utilità di questo parametro, può essere ricercata in quanto enunciato nella Sezione 1.8 del Capitolo 1, dove sono stati annunciati i paradigmi secondo cui operano le radio cognitive. In particolar modo, il paradigma underlay, prevede che le radio cognitive siano a conoscenza dell'interferenza rilevata presso le altre radio. Presupponendo che *CalRadio* venga posizionata abbastanza vicino ad una radio legacy, si può presupporre che il livello di RSSI rilevato da *CalRadio* sia equivalente a quello ricevuto dalla radio non cognitiva e da questo si può quindi ottenere una stima dell'interferenza.

## 3.5 Strutture dati

Per supportare la raccolta delle statistiche sono state create delle opportune strutture dati.

Nell'ottica di mantenere le statistiche sui tredici canali, mantenendo uno storico anche quando si cambia canale, tutte le statiche elencate in Tabella 3.2, vengono mantenute all'interno di array di tredici celle, una per ogni canale. In questo modo, anche se la frequenza di osservazione cambia, resta comunque uno storico delle misurazioni precedenti. Un esempio dell'utilità di questo modo di mantenere i dati è il seguente: supponiamo che *CalRadio* stia operando sul canale undici.
Un processo cognitivo, in base a determinate statistiche circa le performance attuali, decide di spostarsi su di un altro canale. Nella scelta del canale da occupare, si può valutare l'occupazione dello spettro (attraverso le misurazioni del CCA). Se lo spettro risulta essere "equamente libero" su altri cinque canali, la scelta su quale sia il miglior canale su cui spostarsi, può essere dettata da una lettura delle statistiche precedenti, circa le trasmissioni su questi cinque canali candidati. Ovviamente tutto questo può funzionare si si hanno a disposizione delle statistiche per questi cinque canali.

Per quanto riguarda la raccolta di informazioni sui singoli link, è stata creata le seguente struttura dati:

```
typedef enum ulla_dev_tech_t {ULLA_DEV_WIFI = 13 ,
    ULLA_DEV_ZIGBEE = 26} ulla_dev_tech_t;

typedef unsigned short ulla_integer_t;
typedef timeval ulla_timestamp_t;

#define MAC_SIZE 6
typedef struct {
  char srcmac[MAC_SIZE];      //source mac
  char dstmac[MAC_SIZE];      //destination mac
  ulla_integer_t length;      //Rx packet length
  ulla_integer_t Datalength; //Rx packet Data length
  ulla_integer_t rate;        //Rx packet PHY rate
  ulla_integer_t Datarate;    //Rx packet data rate
  ulla_integer_t rxPwr;       //Received packet power
  timeval rxTimestamp;        //Rx start timestamp
} ulla_link_t;

///Structure to manage timings
typedef struct{
  ///milliseconds
  unsigned long ms;
  ///with 100MHz clock 1 tick = 0.1us
```

```
  unsigned short ticks;
} timeval;
```

Quando *CalRadio* riceve un nuovo pacchetto, estrapola i valori per riempire i campi della struttura `ulla_link_t`. Il tipo di dato timeval, serve per conoscere l'istante temporale a cui si riferiscono i valori ed la sua definizione è mostrata nel precedente listato di codice. Per quanto riguarda il tipo `ulla_integer_t`, questo è uno short senza segno.

## 3.6 Comunicazione

Perchè ULLA possa ottenere le statistiche di *CalRadio* sono stati sviluppati un modulo client (lato ULLA) e un modulo server (lato *CalRadio*). Lo scopo di questi due moduli è di istanziare una comunicazione UDP (si ricorda che *CalRadio* comunica con il framework ULLA mediante una connessione di rete ethernet).

Il modulo server, si occupa inoltre di invocare le *ioctl* sul modulo che regola il funzionamento di *CalRadio*. Queste richieste provengono da ULLA e la richiesta è già formattata in modo che sia comprensibile da *CalRadio*.

Visto che ULLA può sia richiedere statistiche, sia impostare parametri, sia eseguire comandi, le *ioctl* sono state suddivise nelle seguenti tre classi:

```
//IOCTL defines
#define ULLA_ATTRIB_IOCTL       (SIOCIWFIRSTPRIV + 10)
#define ULLA_CMD_IOCTL          (SIOCIWFIRSTPRIV + 11)
#define ULLA_UPD_IOCTL          (SIOCIWFIRSTPRIV + 12)
```

La funzione del modulo di *CalRadio* che si occupa di gestire gli *ioctl*, farà un primo controllo sul tipo di *ioctl*. Una volta distinto il tipo di richiesta , ovvero dopo aver determinato se si tratta di un comando, piuttosto che della richiesta di un parametro, piuttosto che della modifica di un valore, tale funzione andrà ad indagare il contenuto dell'*ioctl*.

Tale contenuto, altro non è che la richiesta proveniente da ULLA (nel nostro caso). Se tale richiesta viene riconosciuta come valida, allora viene inoltrata al DSP, altrimenti viene ritornato un messaggio di errore.

*CalRadio* per garantire la stabilità del device, impone dei limiti di tempo nell'esecuzione dei comandi verso il DSP. In particolar modo, viene impostato un timeout entro il quale il DSP può svolgere le operazioni richieste. Se si supera tale limite, il modulo che gestisce *CalRadio* decide che l'operazione non è andata a buon fine e dichiara un fallimento.

Nella richiesta dei parametri, questo tempo non è costante, ovvero ogni operazione richiesta chiede un tempo variabile a seconda della comp-

lessità dell'operazione. Ad esempio, una scansione del CCA sui tredici canali richiede un tempo nettamente superiore che non la richiesta del canale su cui si sta operando.

Serve quindi che il modulo di gestione delle *ioctl*, sia in grado di stimare (upper bound) il tempo richiesto per le varie operazioni ed in seguito imposti correttamente il valore del timeout.

### Comandi

Per *CalRadio* sono stati definiti i seguenti comandi:

1. ULLA_LP_COMMAND_UP
2. ULLA_LP_COMMAND_DOWN
3. ULLA_LP_COMMAND_RTS_CTS_ON
4. ULLA_LP_COMMAND_RTS_CTS_OFF
5. ULLA_LP_COMMAND_RESET_STATISTICS
6. ULLA_LP_COMMAND_SHOW_CHANNEL
7. ULLA_LP_COMMAND_SHOW_FREQUENCY

I Comandi 1 e 2 servono rispettivamente per attivare/disattivare l'interfaccia WiFi di *CalRadio*.

I comandi 3 e 4 abilitano e disabilitano il meccanismo di rts/cts, mentre il comando 5 permette di resettare tutte le statistiche salvate.

Gli ultimi due comandi, influiscono sul funzionamento dei Wireless Tools, in quanto permettono di mostrare il canale corrente in termini di frequenza (7) oppure come numero di canale scelto (6).

Va sottolineato come il comando 1, nel caso l'interfaccia WiFi sia già attiva, permette di eseguire una sorta di reboot dell'interfaccia, con le impostazioni iniziali.

Invece non esiste nessun comando che permette di riavviare l'intera *CalRadio*. L'unico modo per eseguire questa operazione, è intervenire manualmente sul pulsante di reset del device.

# Capitolo 4

# Tests, Analisi dei risultati e Conclusioni

In questo capitolo presenteremo alcuni test eseguiti con l'ausilio del framework ULLA e di *CalRadio*.

I test sono atti a dimostrare il corretto funzionamento del framework, quindi la sua stabilità.

Verranno successivamente presentati dei risultati ottenuti con delle misurazioni del CCA, per stimare l'occupazione dello spettro.

## 4.1 Stabilità del framework

Uno degli aspetti cruciali, nello sviluppo del framework ULLA e nell'implementazione di *CalRadio* è la stabilità dell'intero sistema. Si è cercato di garantire che il sistema funzioni correttamente per periodi di tempo estesi (in linea teorica il sistema non dovrebbe mai andare in crash), anche in condizioni di forte stress dell'architettura.

Per fare questo, è stata data priorità agli aspetti di fault-tollerance del software,sia nell'implementazione di ULLA, che di *CalRadio*.

**ULLA** Il framework ULLA è da considerarsi il punto cardine per la stabilità dell'intero sistema. Nel caso ULLA non operi correttamente, possono verificarsi problemi non solo nel sistema di raccolta dati fornito dal framework, ma anche su tutte le applicazioni che operano nella stessa macchina che esegue ULLA.

Difatti, ULLA è composto di programmi in user space e altri in kernel space. Qualora un modulo del kernel non risponda più ai comandi, o peggio

ancora provochi una *segment fault*, l'intero sistema rischia di andare in crash (kernel panic). Questo comportamento, ovviamente, va evitato.

Ancora, se i moduli che compongono ULLA non vengono progettati in modo corretto, dato che operano in kernel space e quindi con permessi maggiori, si corre il rischio che queste applicazioni vadano ad intaccare il funzionamento dell'intero sistema operativo che supporta ULLA. basti pensare ad un'allocazione errata della memoria, piuttosto che una scrittura dei dati in una posizione errata...

Pertanto, la programmazione del framework è stata testata approfonditamente per scovare eventuali bug di programmazione.

Questa minuziosità nel controllare il corretto funzionamento di tutti gli elementi costitutivi di ULLA, ha portato per l'appunto a trovare e risolvere alcuni bug presenti nell'architettura iniziale di ULLA. In termini di tempo, attualmente ULLA è risultato funzionare correttamente per diverse settimane, sia con un uso intensivo della piattaforma, sia alternando i diversi meccanismi offerti dal framework.

**CalRadio** Per quanto riguarda *CalRadio* la politica seguita è stata la stessa che per ULLA. Tuttavia, le operazioni di debug su questa piattaforma si sono rivelate essere difficili da implementare. Queste difficoltà sono dovute al fatto che la maggior parte del codice per il collezionamento delle statistiche e le altre funzionalità operative di *CalRadio*, sono implementate nel modulo DSP. Il DSP non fornisce un output diretto. L'unico modo di comunicare "verso l'esterno" è fornito dai led di *CalRadio* e dal meccanismo di mailbox.

I led possono tornare utili fino ad un certo punto per le operazioni di debug, mentre la mailbox può essere sfruttata meglio, scrivendoci ad esempio delle stringhe di debug. Anche questo sistema, però impone dei limiti: il DSP deve comunicare "normalmente" con l'ARM e questo avviene tramite il buffer condiviso. Qualora questo buffer venga impiegato anche per le operazioni di debug, vengono a presentarsi delle problematiche sulle temporizzazioni di comunicazione, in quanto il buffer ha una dimensione limitata, come è limitato il rate di trasferimento dati.

Al di là di questi dettagli, si è cercato di implementare le funzionalità di *CalRadio* in modo che questa si il più fault-tollerant possibile, introducendo meccanismi di timeout per l'esecuzione delle operazioni e altri meccanismi di autocorrezione.

I test sulla stabilità svolti su *CalRadio*, l'hanno vista operare per svariati giorni in modo continuo, ovvero collezionando statistiche del CCA con il rate massimo.

Altri test sono stati svolti collezionando alti parametri, quali l'RSSI o

statistiche circa il numero di pacchetti ricevuti e tutti i test hanno portato a risultati soddisfacenti in termini di stabilità.

Per quanto riguarda l'esportazione delle statistiche sui singoli link, anche questo meccanismo è risultato essere stabile.

Va inoltre sottolineato come *CalRadio* essendo un dispositivo sperimentale, possa avere dei problemi intrinseci di stabilità. Difatti, caricando lo stesso software su due diversi dispositivi *CalRadio* si vede come il comportamento di un device sia più stabile dell'altro. Questo non avviene solo nella raccolta delle statistiche, ma anche con semplici operazioni di I/O tramite telnet.

## 4.2 Raccolta di statistiche

Una volta appurata la stabilità dell'intero sistema, sono stati svolti dei test per raccogliere dati statistici. In particolar modo, è stata analizzata l'occupazione spettrale, tramite misure continuative del CCA.

I dati sono stati raccolti con l'ausilio di un'applicazione grafica Java, che opera da LU per il framework ULLA. Tale applicazione viene presentata nell'Appendice B.

In tutti i test presentati successivamente, salvo diverse indicazione, si assume che:

1. La scansione del CCA avviene su tutti e tredici i canali 802.11.
2. Il tempo di scansione su ogni canale è pari a 100 ms.
3. Il tempo totale di scansione dello spettro è di 1.3 s.
4. *CalRadio* opera sempre nello stesso ambiente (stesso posizionamento, stessa antenna, etc.)

Premesso tutto questo, presentiamo ora i risultati ottenuti.

### 4.2.1 Interazione con dispositivi 802.11

Essendo *CalRadio* un dispositivo 802.11b, il primo test svolto, consiste nell'analizzare l'occupazione del canale wireless in presenza di altri dispositivi WiFi.

Per una prima analisi, non è servito predisporre un testbed, in quanto *CalRadio* è posizionata all'interno di un laboratorio del Dipartimento di Ingegneria dell'Informazione, nel quale sono presenti svariati computer collegati alla rete dipartimentale via wireless.

Quando nessun utente utilizza la rete, ad esempio nei giorni di chiusura del dipartimento, si vede come *CalRadio* sia in grado di rilevare del traffico nei canali dall'11 al 13, riconducibile alle operazioni di beaconing svolte dagli Access Point del dipartimento.

Invece, in presenza di utilizzatori, la disponibilità di questi canali diminuisce a causa delle trasmissioni degli utenti.

Dalle analisi svolte, si vede come lo spettro disponibile per la comunicazione non sia sfruttato nel modo più consono, in quanto le prime frequenze dello spettro non vengono utilizzate.

La Figura 4.1, mostra l'occupazione dello spettro secondo una scala colori: i colori caldi implicano una maggiore occupazione del canale; viceversa il colore blu, indica che il canale non è utilizzato. Il grafico, indica i valori percentuali di occupazione temporale di ogni singolo canale.

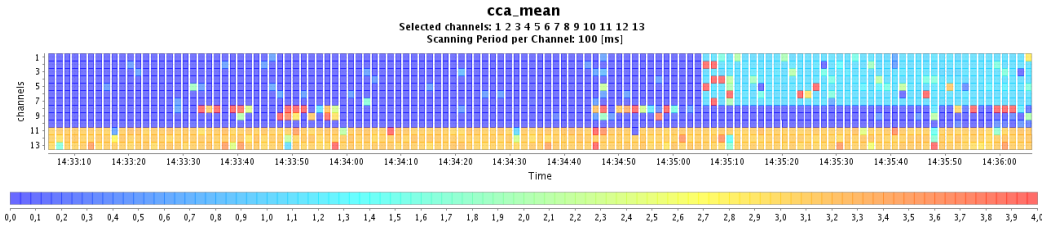



Figura 4.1: Rilevazione dello spettro mediante CCA

Nella prima parte de grafico (circa fino alle 14:35), vediamo che lo spettro del segnale mostra come occupati, i soli canali dall'11 al 13. Questa occupazione, pari circa al 3.2% del tempo di scansione, è relativa ai beacons inviati dagli Access Point del dipartimento.

Negli istanti di tempo successivi, è stata attivata una scheda Atheros, posizionandola sul canale numero 4 (2426 MHz). Tale interfaccia, non effettua alcun tipo di comunicazione, ad esclusione dell'operazione di beaconing. Vediamo come, l'apporto di energia nello spettro fornito da una sola interfaccia, sia inferiore (circa l'1.5%) rispetto all'energia degli altri Access Point dipartimentali.

Nella figura, si vedono inoltre delle occupazioni di canale aperiodiche. Queste possono riferirsi a comunicazioni di altri dispositivi, dei quali non si conosce l'entità. Invece, il grafico di Figura 4.2, mostra la larghezza della banda rilevata da *CalRadio*.

In particolar modo, è stata attivata un'interfaccia Atheros, sul canale 6. Vediamo come i beacons inviati da questo dispositivo, siano rilevabili in termini di energia, in un range di canali che vanno dal 3 al 9, avvalorando così la tesi di [27], sui canali "non-overlapping".

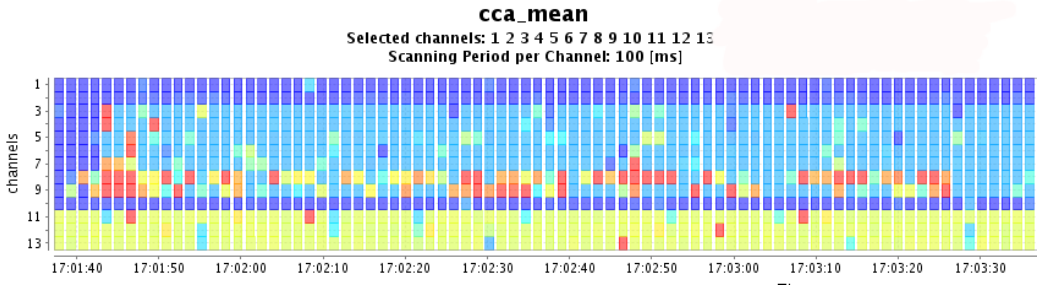



Figura 4.2: Larghezza di banda rilevata da *CalRadio*

Durante lo studio delle rilevazioni del CCA, si è visto come *CalRadio* riesca a rilevare il traffico generato anche da altri dispositivi, anch'essi operanti nella banda ISM dei 2.4 GHz. In particolare, si è in grado di rilevare l'interferenza prodotta dai dispositivi che operano secondo lo standard Bluetooth e ZigBee (802.15.4).

## 4.3 Interazione con dispositivi 802.15.4

Lo scopo dello standard 802.15.4 è di fornire una base per le reti i cui dispositivi sono caratterizzati da una minima complessità, costo contenuto, limitato consumo energetico e connettività wireless a basso data rate. Lo standard definisce due livelli per questa tecnologia: il livello fisico (PHY) e il livello di accesso al canale (MAC).

I dispositivi 802.15.4 operano sulle frequenze ISM 2.4 GHz, 915 MHz e 868 MHz.

Quando si parla di reti 802.15.4, si parla in modo implicito di **ZigBee**.

ZigBee è una specifica per un insieme di protocolli di comunicazione ad alto livello che utilizzano piccoli dispositivi radio a bassa potenza, basati sullo standard 802.15.4, per implementare delle wireless personal area networks (WPANs).

I dispositivi ZigBee, hanno 16 canali nella banda dei 2.4 GHz, con banda di 3 MHz ciascuno. La massima potenza in trasmissione è di 1mW (0 dBm) con un rate massimo di 250 Kbps.La codifica è una OQPSK nella banda dei 2.4 GHZ, mentre viene usata una BPSK per i 915 e gli 868 MHz.

Lo standard 802.15.4 precede un meccanismo di carrier sense per accedere al canale ed in particolar modo viene adottato il CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance).

### 4.3.1 Tmote Sky

Nei test effettuati per indagare l'interferenza tra i dispositivi 802.11 e quelli 802.15.4, sono stati usati i Tmote Sky [28]. Questi dispositivi, sono dei device dotati di una radio 802.15.4 e di alcuni sensori (luce, umidità, temperatura) e vengono utilizzati per creare reti di sensori wireless (WSN).

Le radio che equipaggiano i Tmote Sky, sono le CC2420, prodotte da Chipcom. Queste permettono di trasmettere a sette livelli di potenza, da 0 dBm a -15 dBm e la loro antenna integrata, permette di coprire 25 m in ambiente indoor e ben 125 m all'aperto.

### 4.3.2 Esecuzione dei test

I test sono stati svolti nel seguente modo: i Tmote Sky sono stati programmati per spedire messaggi in broadcast, al rate massimo consentito loro, nella banda dei 2.4 GHz (rispettivamente i canali dal numero 16 al numero 26 dello standard 802.15.4). *CalRadio* nel frattempo eseguiva la scansione dello spettro.

Mediante la pressione di un tasto nel dispositivo Tmote Sky, il device cambiava canale, spostandosi alla frequenza successiva. Questo fenomeno viene completamente rilevato da *CalRadio*. In Figura 4.3 vediamo appunto come *CalRadio* riesca a distinguere il cambio di canale.

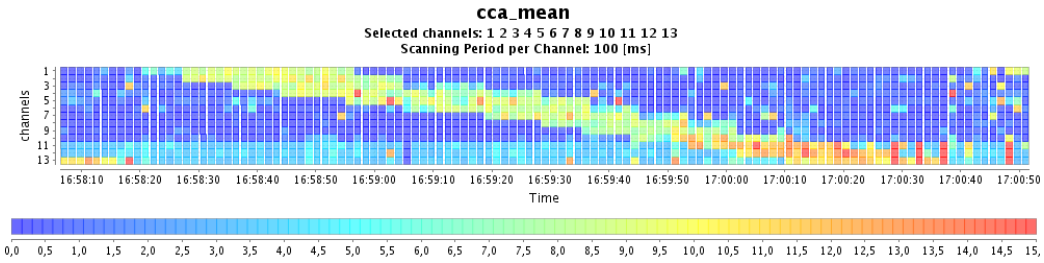



Figura 4.3: Rilevamento delle frequenze 802.15.4

In Figura si vede come ZigBee abbia un'occupazione, in termini di banda, inferiore a quella del segnale 802.11.

Pertanto si possono elaborare delle tecniche per distinguere le comunicazioni 802.11 da quello 802.15.4, semplicemente analizzando la banda rilevata dal CCA.

In [29], viene svolta un'analisi della probabilità di collisione tra dispositivi operanti nello standard 802.15.4 e 802.11b.

Il risultati di questa analisi, mostrano come la probabilità di collisione

tra le due tecnologie cresce in modo lineare, all'aumentare del numero di dispositivi 802.15.4.

In Figura 4.4, vengono mostrate le sovrapposizioni tra i canali dei due standard.

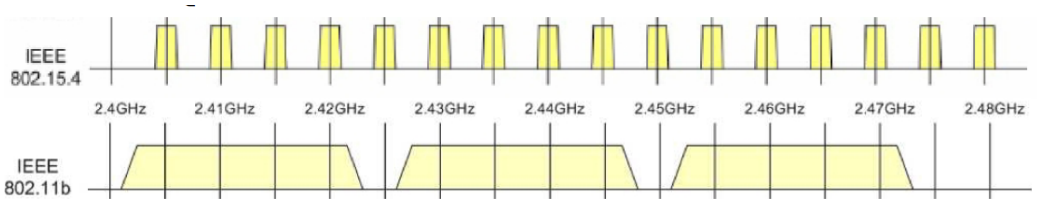



Figura 4.4: Allocazione dei canali IEEE 802.15.4 e IEEE 802.11b

Le frequenze di centro banda per gli standard IEEE 802.15.4 e IEEE 802.11b, vengono calcolate con le seguenti:

$$f_{802.15.4} = 2405 + 5(k - 11); k = 11..26 \tag{4.1}$$

$$f_{802.11b} = 2412 + 5k; k = 0..13 \tag{4.2}$$

In Figura 4.5 viene mostrata l'interferenza di tre segnali ZigBee. Si vede come l'occupazione del CCA aumenti notevolmente e di come questa vada a sovrapporsi alla traccia 802.11 già esistente.

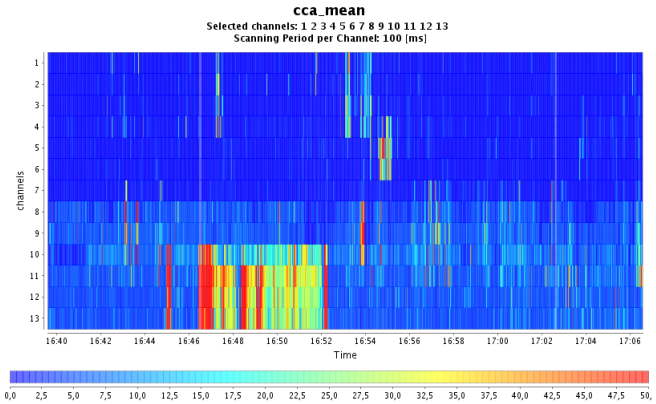



Figura 4.5: Interferenza dovuta a 3 dispositivi IEEE 802.15.4 comunicanti sullo stesso canale

## 4.4 Interazione con dispositivi Bluetooth

Un'altra tecnologie che condivide le bande ISM, è il Bluetooth (BT).

Bluetooth fornisce un metodo standard, economico e sicuro per scambiare informazioni tra diversi dispositivi, entro un raggio di qualche decina di metri. La specifica Bluetooth è stata sviluppata da Ericsson e in seguito formalizzata dalla Bluetooth Special Interest Group (SIG). SIG è un'associazione formata da Sony Ericsson, IBM, Intel, Toshiba, Nokia e altre società che si sono aggiunte come associate o come membri aggiunti.

L'obiettivo principale, che ha guidato lo sviluppo di questa tecnologia, è lo sviluppo di dispositivi economici, sia dal punto di vista della produzione, sia dei consumi.

I dispositivi BT, collegati tra loro formano una piconet. Le topologie supportate sono:

- punto - punto
- punto - multi punto
- scatternet

Ogni piconet è formata da un unico nodo master e diversi (fino a 7) nodi slave. L'unione di più piconet, porta ad una scatternet.

Un dispositivo Bluetooth può partecipare sequenzialmente a diverse piconet come slave attraverso luso di tecniche TDM (Time Division Multiplexing), ma può essere master solo in una di queste. Un dispositivo Bluetooth si può trovare essenzialmente in due stati: in quello di connessione o in quello di standby. L'unità si trova nello stato di connessione se è connessa ad un altro dispositivo ed è coinvolta con esso in una comunicazione. Se il dispositivo non è connesso, o non è coinvolto nelle attività della piconet, allora esso si trova nello stato di standby. Questo stato è stato concepito come un modo per far risparmiare energia ai dispositivi. Quando un'unità si trova in standby, ascolta il canale ogni 1,28 secondi per eventuali messaggi dal master.

Quando un nodo è connesso, può operare nelle seguenti modalità:

- **Active mode** L'unità partecipa attivamente alla piconet, sia in ricezione che in trasmissione, ed è sincronizzata al clock del master. Il master trasmette regolarmente per mantenere la sincronizzazione del sistema.

- **Hold mode** Il master può mettere i dispositivi slave nello stato di Hold per un tempo determinato. Durante questo periodo nessun pacchetto può essere trasmesso dal master anche se il dispositivo mantiene la sincronizzazione con il master. Questa modalità operativa è utilizzata nel momento in cui non si devono inviare pacchetti ad un dispositivo per un periodo relativamente lungo in modo da permettere al dispositivo slave di risparmiare energia.

L'Hold mode può essere utilizzata anche nel caso in cui un'unità vuole scoprire o essere scoperta da altri dispositivi BT o vuole partecipare ad altre piconet.

- **Sniff mode** Questo stato indica la modalità di risparmio energetico. Per entrare in sniff mode, master e slave devono negoziare due parametri: uno `sniff interval` ed uno `sniff offset`. Con il primo si fissano gli slot di sniff, mentre con il secondo si determina listante del primo slot di sniff. Quando il collegamento entra in sniff mode, il master può inviare pacchetti solamente all'interno degli sniff slot.

- **Park mode** Un dispositivo in park mode, è considerato inattivo. Questa modalità viene utilizzata nella piconet per estendere il numero di dispositivi collegati ad essa, da 7 a 255 (di cui solo 7 possono essere contemporaneamente attivi oltre al master). Questa modalità prevede che i nodi slave rimangano comunque sincronizzati con il master.

### 4.4.1 Protocollo di comunicazione

La comunicazione tra due nodi BT, è regolata da un TDD (Time Division Duplex) basato sui pacchetti, per cui si ha un tempo di slot di 625 $\mu$s, nel quale avviene una comunicazione da parte di un nodo. La sincronizzazione, tra master e slave, prevede che un dispositivo master comunichi negli slot pari, mentre allo slave vengono assegnati quelli dispari.

Il meccanismo di comunicazione, è inoltre regolato da un politica di hopping sui 79 canali Bluetooth, per cui ogni 625 $\mu$s, viene cambiato il canale.

La massima frequenza di hopping è di 1600 hop/s quando i dispositivi sono nello stato "connesso", mentre di 3200 hop/s quando i dispositivo si trova nello stato di "inquiry".

Vengono definiti quattro possibili canali di comunicazione:

- **BASIC PICONET PHY CHANNEL** Questa modalità è caratterizzata da una sequenza di hopping pseudo-random su tutti i 79 canali. La sequenza di hopping è determinata dal clock e dall'indirizzo del nodo master.

- **ADAPTED PICONET PHY CHANNEL** Questa modalità si differenzia dalla precedente, poichè non utilizza l'intero spettro per eseguire l'hopping, ma un sottoinsieme dei canali disponibili (con un minimo di 20 canali).

- **PAGE SCAN PHY CHANNEL** Questa modalità comporta una frequenza di hopping inferiore ai casi precedenti e viene determinata dal clock del dispositivo che esegue lo scan e dal suo indirizzo..

- **INQUIRY SCAN PHY CHANNEL** Come nel caso precedente, la frequenza di hopping è inferiore rispetto ai canali piconet. La temporizzazione viene gestita dal clock del dispositivo che esegue lo scan, mentre la sequenza di hopping viene determinata dall'*inquiry access code*.

La comunicazione, regolata dalla sequenza di hopping e dai clock di master e slave sincronizzati, può operare su singolo slot o in modalità multi slot. In Figura 4.6 viene mostrata la comunicazione, utilizzando 1, 3 o 5 slot.

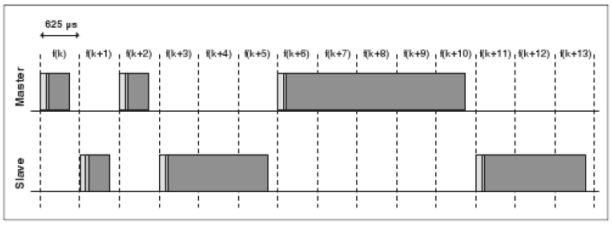



Figura 4.6: Comunicazione single e multi-slot Bluetooth

### 4.4.2 Hopping

Il meccanismo di hopping di Bluetooth, segue un algoritmo pseudo-random. Lo standard [30] definisce sei differenti sequenze di hopping: cinque per un sistema di hopping base e una per un hopping adattato, usando l'AFH (Adapted Frequency Hopping). Le 6 sequenze di hopping sono:

1. Page hopping sequence

2. Page response hopping sequence

3. Inquiry hopping sequence

4. Inquiry response hopping sequence

5. Basic channel hopping sequence

6. Adapted channel hopping sequence

Lo schema di selezione è composto di due fasi: per primo, si sceglie la frequenza di hopping. Successivamente, si deve mappare tale sequenza nelle frequenze di hopping. Lo schema di Figura 4.7, mostra questo procedimento.

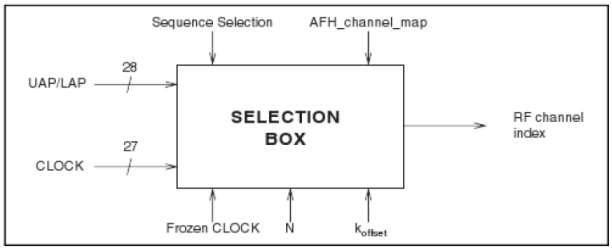



Figura 4.7: Selezione della frequenza di hopping

### 4.4.3 Esecuzione dei test

Una volta capito come opera Bluetooth, è stata creata una piconet tra due dispositivi BT e si è operato un trasferimento dati. I dispositivi in questione si trovavano in un raggio di 10 metri da *CalRadio* e la distanza tra loro variava da pochi centimetri a qualche metro.

La scansione dello spettro, mostra chiaramente come l'hopping di Bluetooth vada ad impegnare tutte le frequenze rilevate dal *CalRadio* in un modo uniforme.

In Figura 4.8, si vede la traccia del segnale Bluetooth.

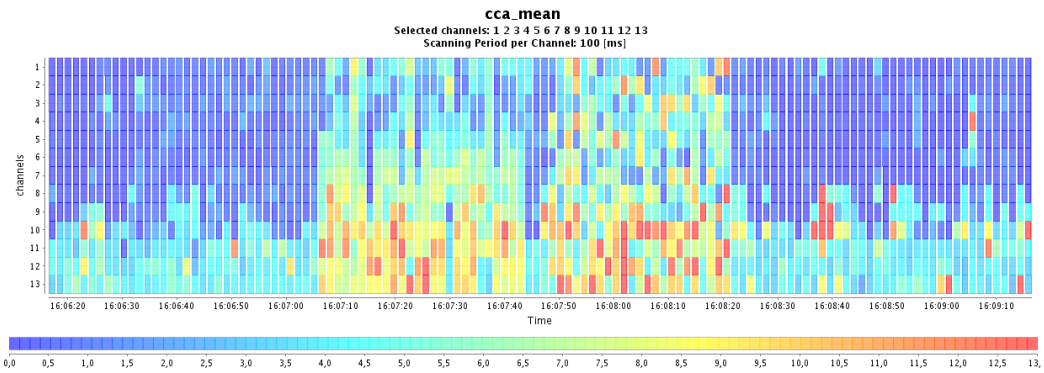



Figura 4.8: Traccia del segnale Bluetooth

Nella parte centrale di Figura 4.8, si può notare la trasmissione BT. In questo caso, si riesce a distinguere una fase di comunicazione iniziale, nella

quale vi è la ricerca dei dispositivi Bluetooth e la loro sincronizzazione (da 16:07:05 a 16:07:40), seguita dalla fase di trasmissione vera e propria (da 16:07:50 a 16:08:20).

Durante i test si è cercato di trovare una periodicità nel pattern della traccia Bluetooth, tuttavia questo non è stato possibile rilevarla, a causa di limitazioni hardware: Bluetooth fa hopping ogni 625 $\mu$s, mentre la risoluzione minima offerta da *CalRadio* è di circa 10 ms, con un tempo di 7 $\mu$s per cambiare canale. Pertanto è impensabile di riuscire a rilevare tutti gli hop della trasmissione Bluetooth.

Dopo una ricerca nella letteratura, ed in seguito ad alcuni esperimenti, è stato svolto il seguente studio sull'interazione tra la tecnologia Bluetooth e 802.11, al fine di poter rilevare una comunicazione Bluetooth, tramite l'osservazione del CCA.

### 4.4.4 Rilevamento di comunicazioni Bluetooth

La comunicazione Bluetooth è basata su un meccanismo sincronizzato di tipo master-slave. Il trasferimento dati, utilizza delle tecniche ARQ, in modo da aumentare l'affidabilità della della comunicazione. Questo avviene per gli stream dati, mentre per il trasferimento del traffico voce, non viene usato ARQ, rendendo quindi la comunicazione non affidabile.

Viceversa, le comunicazioni tra i nodi 802.11, non sono sincronizzate, ma vengono coordinate dal meccanismo di Backoff a livello MAC. Tale meccanismo, si basa su metodi statistici per evitare collisioni nella trasmissione, in seguito al sensing del CCA.

Nel caso si verifichino delle collisioni, il traffico Bluetooth viene corrotto, ma questo viene rilevato solo dal livello PHY, che non riesce a demodulare correttamente i pacchetti. Il protocollo MAC, invece non viene affetto da tali interferenze.

Al contrario, le comunicazioni WiFi, che si basano su schemi DSSS, sono in genere più robuste. Difatti tali trasmissioni sono resistenti alle interferenze dei segnali Narrow-Band, come nel caso del Bluetooth. Tuttavia, il segnale Bluetooth viene facilmente rilevato dal correlatore nei transceiver WiFi e questo va ad influenzare il meccanismo di trasmissione a livello MAC.

In seguito a queste considerazioni, possiamo dire che, quando le trasmissioni Bluetooth si sovrappongono alle trasmissioni 802.11, l'energia media rilevata dal CCA, viene affetta dalla sovrapposizione dei seguenti effetti:

- Durante la trasmissione di un frame Bluetooth, quando il nodo WiFi è in fase di Backoff, la trasmissione WiFi viene rimandata di 625 $\mu$s $\times$ (1,3,5)

slots temporali. In questa fase, il contributo alla misura del CCA è data dalle sole comunicazioni Bluetooth.

- Durante la trasmissione di una PDU 802.11, tutte le trasmissioni Bluetooth vengono "mascherate" dalla trasmissione del pacchetto WiFi, pertanto l'influenza del segnale Bluetooth sulla misura del CCA, è dovuta ai soli pacchetti BT che non si sovrappongono alla trasmissione WiFi.

In Figura 4.9, viene schematizzato questo comportamento.

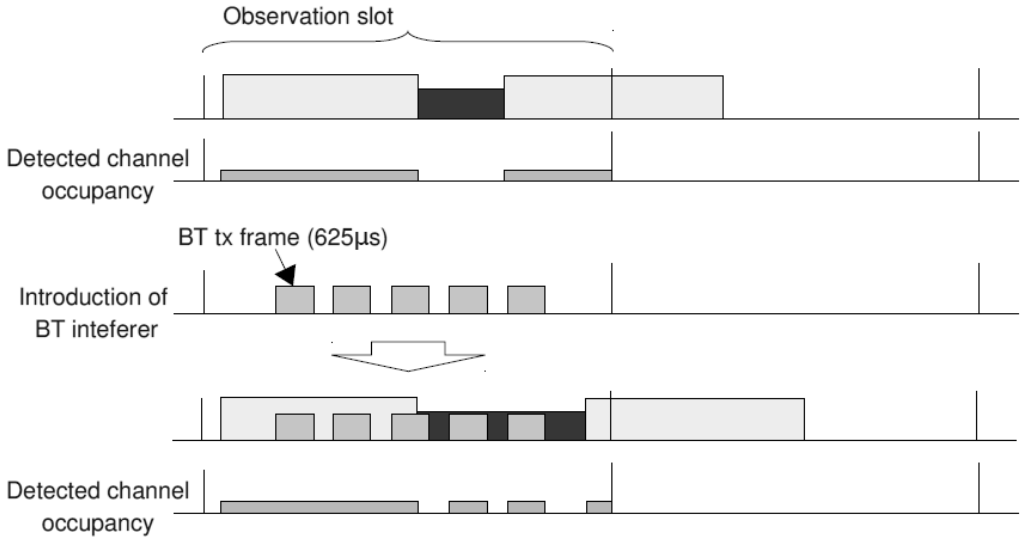



Figura 4.9: Overlapping di comunicazioni 802.11 e Bluetooth

Visto che l'occupazione del CCA dovuta al segnale BT è invariante rispetto all'interferenza WiFi, il punto chiave per analizzare queste situazioni, è l'occupazione dovuta alla trasmissioni 802.11. In Figura 4.9, viene presunto il caso in cui le trasmissioni WiFi siano broadcast, quindi non viene utilizzato il meccanismo degli ACKnoledge.

Nel caso l'interferenza Bluetooth interferisca durante una comunicazione WiFi che occupa la maggior parte dello slot temporale di osservazione, il suo contributo nella misura del CCA non viene rilevato, in quanto è "assorbito" dalla comunicazione WiFi. In questa situazione, è possibile che la comunicazioni BT mandi in Backoff la trasmissione WiFi.

La situazione opposta, avviene quando la comunicazione WiFi deve spedire una quantità inferiore di dati, rispetto alla durata finestra della finestra di osservazione. In questo caso, la comunicazione BT non ritarda le trasmis-

sioni WiFi, in quanto riesce a collocarsi negli istanti temporali in cui la comunicazione WiFi è idle.

Il comportamento di questi due scenari, viene mostrato in Figura 4.10.

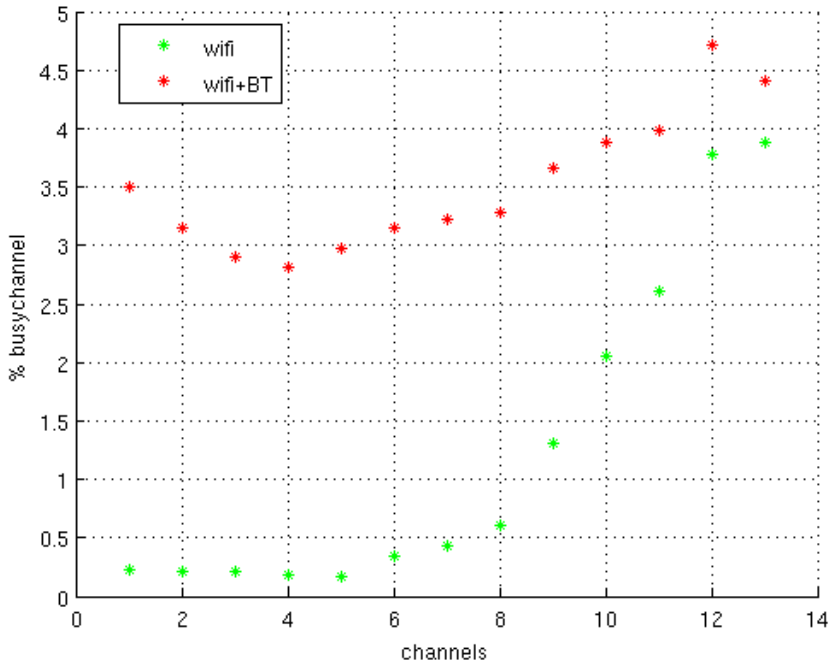



Figura 4.10: Analisi della comunicazione 802.11 e dell'effetto di una comunicazione Bluetooth sovrapposta

Nel grafico, vediamo l'occupazione media di una comunicazione WiFi (verde). In rosso, viene mostrata la sovrapposizione di una comunicazione BT.

Per eseguire questo test, è stato analizzato lo spettro wireless, per un tempo sufficientemente lungo da ottenere un'occupazione media delle frequenze.

Successivamente, nello stesso ambiente, è stata attivata una comunicazione Bluetooth di trasferimento dati.

Nel grafico di Figura 4.10, si vede per l'appunto il comportamento descritto precedentemente:

- I canali 1-7, presentano un'occupazione media, dovuta al solo segnale WiFi circa del 0.3%; al contrario, i successivi canali, sono affetti da comunicazioni WiFi che occupano lo spettro con una percentuale che varia dal 2% al 4 %.

Sommando una trasmissione Bluetooth, la quale spedisce pacchetti senza eseguire il sensing del canale e ignorando quindi se il canale sia già occupato o meno da altre trasmissioni, vediamo come nei canali "scarichi" la misura del CCA sia influenzata dalla trasmissione BT. Questo è il caso in cui il Bluetooth trasmette nei periodi di idle delle comunicazioni WiFi.

- Invece, nei canali 12 e 13, vediamo come una trasmissione già presente (beaconing degli Access Point), vada ad "assorbire" l'effetto delle comunicazioni BT. Il risultato di questa situazione, è che il CCA misurato su tali frequenze, mostra un incremento inferiore rispetto alle rilevazioni sui canali "scarichi".

Nelle Figure 4.11, vediamo l'occupazione media del solo segnale WiFi e della sovrapposizione del segnale Bluetooth.

### 4.4.5 Lavori correlati

In letteratura, si trovano vari studi circa l'influenza del segnale Bluetooth nelle comunicazioni 802.11 e viceversa.

In particolar modo, evidenziamo i seguenti studi:

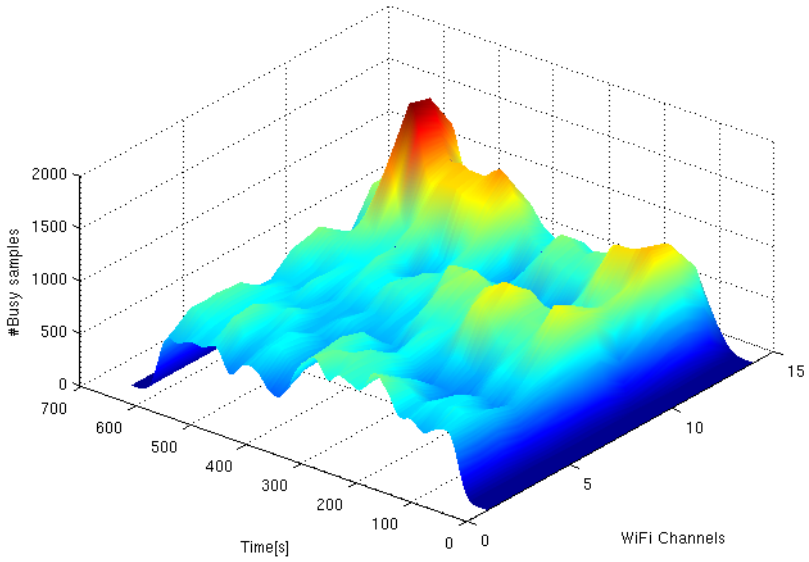
**Experimental Results for Interference between Bluetooth and IEEE 802.11b DSSS Systems** In questo articolo [31], viene misurata in modo sperimentale l'interferenza prodotta da dispositivi Bluetooth sulle comunicazioni 802.11.

Le misure mostrano come la comunicazione 802.11b mantenga delle buone performance in presenza di segnale Bluetooth, in termini di Packet Loss. Invece, in termini di banda, si vede come le performance di WiFi degradano rapidamente.

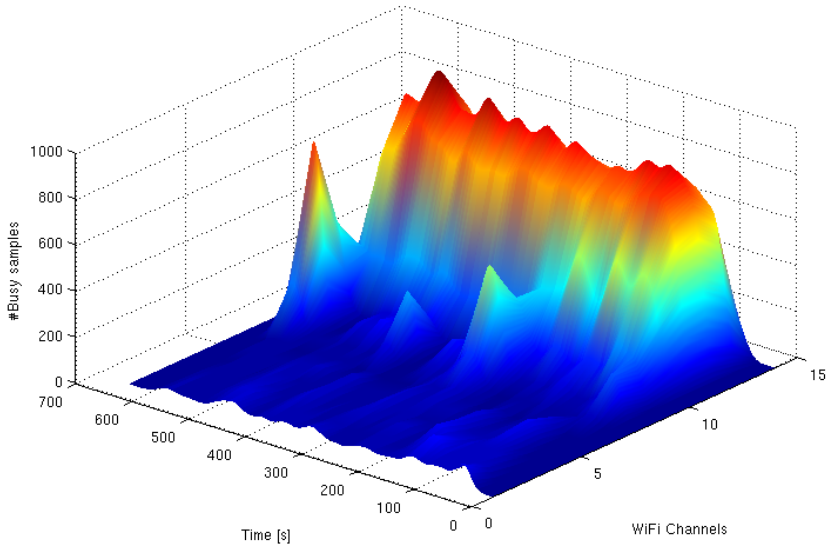
Questi due risultati, confermano le nostre osservazioni, ovvero in presenza di comunicazione BT, le trasmissioni WiFi entrano in Backoff, limitando il fenomeno di packet loss e diminuendo la banda.

L'articolo continua indagando come variano le performance delle comunicazioni Bluetooth. Viene messo in evidenza come le performance del Bluetooth degradino quando l'interferenza del segnale WiFi diviene confrontabile con il segnale BT, aumentando il packet loss piuttosto che la banda, in quanto il protocollo Bluetooth, non effettua carrier sensing.

Anche questa conclusione, è in linea con le nostre osservazioni.




(a) Occupazione del segnale WiFi




(b) Occupazione dei segnali WiFi e Bluetooth

Figura 4.11: Occupazione dei segnali WiFi e Bluetooth

**On the coexistence of Bluetooth and 802.11 technologies in real networking environments** Zeadally et Al. in [32], indagano le performance di Bluetooth in presenza di comunicazioni wireless.

Il loro studio dimostra come le performance di Bluetooth degradino rapidamente in presenza di comunicazioni WiFi, indicando come cause di questo fenomeno l'aumento delle ritrasmissioni di Bluetooth.

Altri articoli che confermano le nostre osservazioni possono essere trovati in bibliografia ([33, 34]).

## 4.5 Confronto delle tre tecnologie

Nei paragrafi precedenti sono state analizzate le interazioni tra tre diversi standard che operano nelle bande ISM dei 2.4 GHz: WiFi, Bluetooth e ZigBee.

Nella Tabella 4.1 vengono riassunte le caratteristiche di queste tecnologie.

| **Standard IEEE** | **802.11b** | **802.15.1** | **802.15.4** |
|---|---|---|---|
| **Frequenza** | 2.4GHz | 2.4GHz | 2.4GHz |
| **Banda** | 22MHz | 1MHz | 2MHz |
| **Numero di canali** | 11 | 79 | 16 |
| **Rate di trasmissione massimo (Mbps)** | 11 | 0.72 | 0.25 |
| **Range (m)** | 100 | 10 | 20 |
| **Applicazioni** | WLAN | WPAN | LR-WPAN |

Tabella 4.1: Caratteristiche delle tecnologie a confronto

Si è visto come *CalRadio* sia in grado di rilevare le tre diverse tecnologie. É facile notare, come i tre diversi standard "colorino" in modo diverso lo spettro radio: Bluetooth ha un pattern caratteristico, distinguibile grazie ad una colorazione pseudo-casuale, dove gli hop effettuati dalla tecnologia BT sono visibili su un solo canale 802.11 per volta.

Il segnale 802.15.4 è distinguibile grazie alla sua occupazione di banda, che "colora" al più quattro canali WiFi. la banda occupata dal segnale WiFi, infine occupa 22MHz, quindi è la più vasta.

Nelle Figure 4.12 e 4.13 vediamo la sovrapposizione dei segnali delle tre tecnologie: WiFi, Bluetooth e 802.15.4.

L'interpretazione della Figura 4.13 è la seguente:

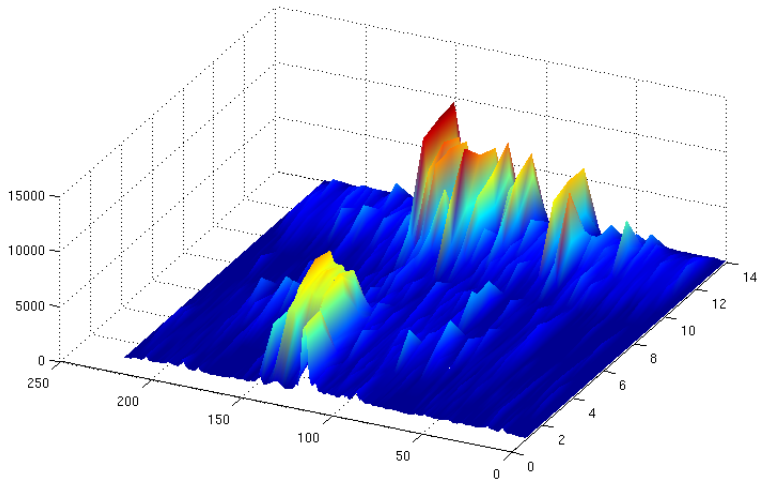



Figura 4.12: Occupazione dei segnali WiFi,Bluetooth e 802.15.4

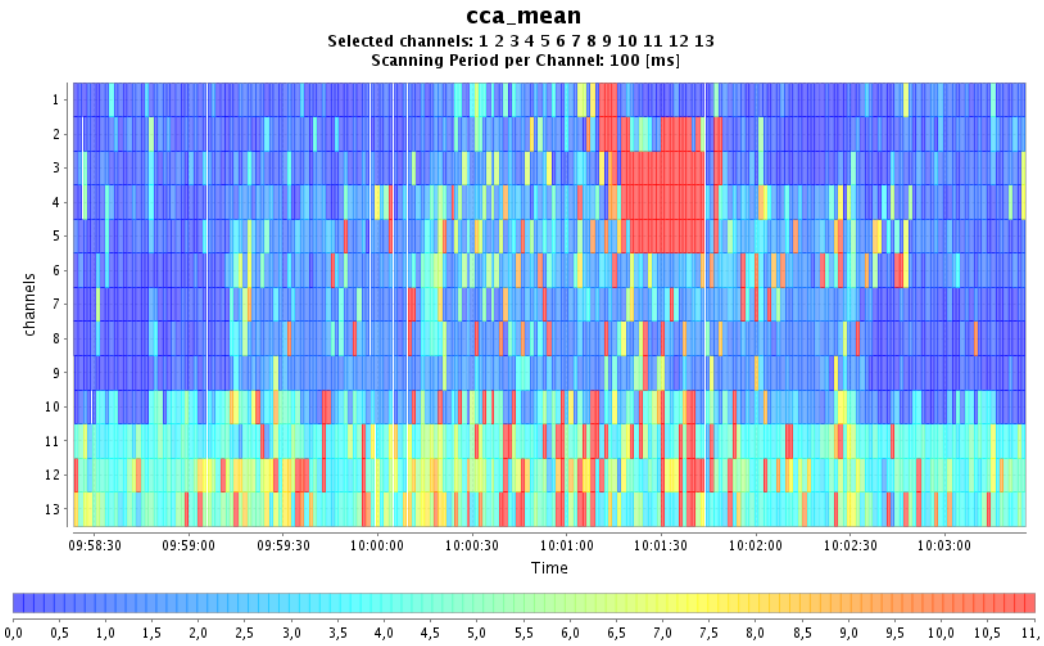



Figura 4.13: Occupazione dei segnali WiFi,Bluetooth e 802.15.4

- Alle 9:59 viene attivata una comunicazione 802.11 sul canale 8. Si vede che questa influisce nella misura del CCA su una banda di 22 MHz circa.

- Alle 10:00 viene attivata un trasferimento dati Bluetooth. Si vede come la presenza ci "celle colorate" su tutto lo spettro.

- Alle 10:01 viene attivato un Tmote Sky che spedisce pacchetti broadcast al rate massimo (con il meccanismo CSMA disabilitato). Questo viene portato a comunicare sul canale WiFi numero 3. Vediamo la tipica occupazione di 4 canali WiFi.

- Alle 10:01:30 viene disattivato il nodo 802.15.4 e rimangono attive le comunicazioni BT e 802.11.

- Alle 10:02 viene disattivata l'interfaccia 802.11. Vediamo come i canali dal 5 al 13 vengano "liberati" (occupazione minore).

- Alle 10:02:30 viene terminata la trasmissione BT e successivamente, lo spettro torna alla sua occupazione "normale".

## 4.6 Conclusioni

Il progetto svolto, ha visto lo sviluppo e l'integrazione del framework ULLA con la piattaforma *CalRadio*.

Sono state definite le API di comunicazione e i protocolli necessari alla raccolta delle statistiche, mediante interrogazioni dirette ad ULLA. In quest'ottica, il framework svolge funzionalità di middleware per vari tipi di dispositivi fisici e, grazie alle strutture dati create, diverse tecnologie.

Gli obiettivi prefissati sono stati raggiunti con successo, creando un sistema stabile e reattivo, con una ricca suite di API per il collezionamento dei parametri, mediante meccanismi sincroni e asincroni di raccolta dati.

Le estensioni apportate al framework ULLA permettono di integrare in maniera trasparente nuovi dispositivi fisici anche operanti con diverse tecnologie all'interno del framework, definendo nuove metriche e algoritmi per risolvere lo stato della rete e definire le ottimizzazioni necessarie.

Il lavoro svolto su *CalRadio* ha permesso di collezionare statistiche di livello MAC e PHY, mantenendo comunque la piena operatività della radio, intesa come dispositivo di comunicazione stand alone, ovvero senza disturbare in alcun modo il flusso della comunicazione.

Particolare rilievo, a livello fisico, ha avuto la misura del CCA (Clear Channel Assessment), che permette di conoscere l'occupazione dello spettro

da parte delle varie tecnologie operanti nelle bande ISM. In particolare, stato possibile rilevare non solo le comunicazioni WiFi, ma anche il traffico generato da tecnologie quali Bluetooth e Zigbee.

In quest'ottica sono stati presentati dei risultati, nei quali viene sottolineato come sia possibile distinguere gli effetti di queste tecnologie sull'occupazione dello spettro radio. Per fare questo, è stata sviluppata un'applicazione grafica per la raccolta e la visualizzare delle statistiche.

Nell'ambito del progetto Europeo ARAGORN in cui si inserisce la tesi, sono state rispettate tutte le specifiche richieste, permettendo cosi l'integrazione del modulo ULLA con la relativa estensione per *CalRadio*, nell'architettura di ARAGORN, permettendo così al Cognitive Resource Manager, modulo centrale dell'architettura sviluppata in ARAGORN, di avere le informazioni necessarie ad effettuare un'ottimizzazione cognitiva delle performance della rete.

### 4.6.1 Sviluppi futuri

Il progetto svolto, offre possibilità di sviluppo lungo diverse direzioni.

- In primo luogo, si dovrebbe completare il meccanismo di esportazione delle statistiche relative alle singole comunicazioni. Attualmente queste statistiche vengo esportate correttamente dal *CalRadio*, ma non vengono ancora gestite da ULLA.

- Vi è la possibilità di creare nuovi moduli LP per integrare diversi dispositivi hardware. In quest'ambito può essere esteso ulteriormente il set dei parametri statistici collezionabili mediante il framework ULLA.

- In un'ottica di analisi e raccolta dei dati, è possibile sviluppare applicazioni ed algoritmi cognitivi per l'ottimizzazione a medio e lungo termine, servendosi delle statistiche raccolte mediante ULLA e *CalRadio*.

- Per quanto riguarda lo studio dell'occupazione dello spettro, si è visto come sia sia possibile distinguere, mediante l'osservazione di un operatore, le comunicazioni di diverse tecnologie. In quest'ottica sarebbe interessante elaborare degli algoritmi di riconoscimento delle tecnologie trasmissive, in base ai pattern rilevati dalle misure di CCA.

# Appendice A

# ULLA

Il framework ULLA mette a disposizione dei tools per l'interrogazione dei dispositivi fisici collegati al framework.

Questi programmi, accettano come parametro una query scritta secondo la grammatica UQL.

Se la richiesta è corretta, ULLA interpreta la richiesta e l'inoltra al device corretto. Questo risponde con i dati rilevati, oppure, nel caso l'operazione non vada a buon fine, restituisce un codice d'errore.

Mostriamo ora le regole della sintassi UQL, i codici d'errore gestiti dal framework ed in seguito alcuni esempi d'utilizzo dei tool.

## A.1 UQL

**Descrizione dei token:**

AVG Average function

MIN Minimum function

MAX Maximum function

SUM Summation

COUNT Number of entries found

FROM Specifies a list of class names

IS Keyword

NOT Negation

NULL Value of and attribute which is not available

AND Logical AND operation

OR Logical OR operation

SELECT Keyword for an UQL select_statement

WHERE Keyword for the where_clause in an UQL select statement

SHOW Reflection function

TABLES Reflection subject is a list of tables

ATTRIBUTES Reflection subject is a list of attributes

COMMANDS Reflection subject is a list of commands

UPDATE Keyword for the UQL update function on attributes values

SET Keyword for the UQL set function on attribute values

* Allquantor

### A.1.1 Select Statement

La sintassi per uno statement di select è la seguente:

```
SELECT select_clause
FROM from_clause
 [WHERE where_clause]
SEMICOLON
```

Esempi di interrogazioni sono:

```
SELECT *
FROM ullaLinkProvider;
```

Questa query, richiede i valori di tutti i parametri afferenti alla tabella `ullaLinkProvider`.

```
SELECT cca_mean, MyChannel, MyChannel_rssi
FROM ullaLinkProvider
WHERE name= 'template_lla';
```

Questa query, richiede i valori del CCA, del canale corrente e del valore medio di RSSI del canale corrente, relativi al device *CalRadio*.

### A.1.2 Update Statement

La sintassi per uno statement di select è la seguente:

```
UPDATE update_clause
SET set_clause
 [WHERE where_clause]
SEMICOLON
```

Un esempio di utilizzo è il seguente, nel quale si vuole impostare il canale in cui opera *CalRadio*sul numero 7:

```
UPDATE ullaLinkProvider
SET MyChannel = 7
WHERE name = 'template_lla';
```

## A.2 Codici d'errore

ULLA definisce i seguenti codici d'errore, il cui significato è intuibile dal nome del codice d'errore.

```
#define ULLA_ERROR_BASE             (0)
#define ULLA_OK
                (-(ULLA_ERROR_BASE+0))
#define ULLA_NO_MEMORY
                (-(ULLA_ERROR_BASE+1))
#define ULLA_INSUFICIENT_MEMORY
                (-(ULLA_ERROR_BASE+2))
#define ULLA_ERROR_IN_VALUE
                (-(ULLA_ERROR_BASE+3))
#define ULLA_ERROR_FAULT
                (-(ULLA_ERROR_BASE+4))
#define ULLA_NOT_SUPPORTED
                (-(ULLA_ERROR_BASE+5))
#define ULLA_ATTRIBUTE_NOT_AVAILABLE
                (-(ULLA_ERROR_BASE+6))
#define ULLA_SEGMENTATION_FAULT
                (-(ULLA_ERROR_BASE+7))
#define ULLA_CORE_NOT_PRESENT
                (-(ULLA_ERROR_BASE+8))
#define ULLA_BUG
                (-(ULLA_ERROR_BASE+42))
#define ULLA_NO_MORE_TUPLES_ERROR
                (-(ULLA_ERROR_BASE+15))
#define ULLA_FIELD_VALUE_ERROR
                (-(ULLA_ERROR_BASE+16))
#define ULLA_ERROR_NOTIFICATION_REQUEST_CANCELED
                (-(ULLA_ERROR_BASE+17))
```

```
#define ULLA_ERROR_SAMPLE_REQUEST_CANCELED
                (-(ULLA_ERROR_BASE+18))
#define ULLA_ERROR_INTERRUPTED
                (-(ULLA_ERROR_BASE+19))
#define ULLA_ERROR_LOCKED
                (-(ULLA_ERROR_BASE+20))
```

## A.3 ULLAReflection

Il tool ULLAReflection serve per ottenere informazioni circa la struttura delle tabelle e i parametri che che le costituiscono.

Per conoscere le tabelle disponibili:

```
user@ulla ~ $ ullaReflection "SHOW TABLES;"

ullaReflection: SHOW TABLES;
ULLA tools: ULLA result

name: ullaLinkProvider
type: structured
description: This class provides an abstraction of the Network
   Interface Card (NIC).
value_set: version 0.1+txcwmin


name: ullaLink
type: structured
description: This class provides an abstraction of common link
   characteristics for all link technologies.
value_set: version 0.1
```

Per ottenere la lista dei parametri disponibili nella tabella `ullaLinkProvider`:

```
user@ulla ~ $ ullaReflection "SHOW ATTRIBUTES FROM
   ullaLinkProvider;"

ullaReflection: SHOW ATTRIBUTES FROM ullaLinkProvider;
ULLA tools: ULLA result

name: name
type: string
description: Name of the Link Provider.
value_set:
unit:

name: description
type: string
```

```
description: Supplier of LLA, the manufacturer of  NIC, etc.
value_set:
unit:

name: version
type: string
description: Version of the Link Layer Adapter (LLA).
value_set:
unit:

name: lpId
type: integer
description: Id of the Link Provider (LP) is unique during
   runtime, however subject to change after re-registration of
   the same LP.
min: 0
max: 2147483647
value_set:
unit:

name: state
type: integer
description:  The state of the Link Provider (LP) indicates the
   operational state of the LLA, such as: Down [1], Registering
   [2], Up [3], Deregistering [4]. The Registering and De-
   registering states can be used on the condition clause of an
   UQL-statement to filter for new LPs registering with the ULLA
    Core.
min: 0
max: 4
value_set: 0, 1, 2, 3, 4
unit:

name: txCwMin
type: integer
description: Contention window bottom limit.
min: 0
max: 32767
unit: slot(s)

name: txCwMax
type: integer
description: Contention window top limit.
min: 0
max: 32767
unit: slot(s)

...
```

Per ottenere la lista dei comandi:

```
user@ulla ~ $ ullaReflection "SHOW COMMANDS FROM
    ullaLinkProvider;"

ullaReflection: SHOW COMMANDS FROM ullaLinkProvider;
ULLA tools: ULLA result

name: up
type: integer
description: switch up/down interface
min: -4
max: 0
parameter: -

name: down
type: integer
description: switch up/down interface
min: -4
max: 0
parameter: -

name: show_channel
type: integer
description: Show channel in iwtools instead of frequency
min: -4
max: 0
parameter: -

name: show_frequency
type: integer
description: Show frequency in iwtools instead of channel
min: -4
max: 0
parameter: -

name: reset_statistics
type: integer
description: Reset statistics for all channels
min: -4
max: 0
parameter: -

name: rts_cts_on
type: integer
description:
min: -4
max: 0
parameter: -
```

```
name: rts_cts_off
type: integer
description:
min: -4
max: 0
parameter: -
```

## A.4 ULLAAttribute

UllaAttribute permette di interrogare ULLA circa il valore di uno o più parametri, ad esempio:

```
user@ulla ~ $ ullaAttribute "SELECT channels,t_period_ms,
    cca_mean FROM ullaLinkProvider WHERE name = 'template_lla';"

ullaAttribute: SELECT channels,t_period_ms,cca_mean FROM
    ullaLinkProvider WHERE name = 'template_lla';
ullaAttribute: urId 2
ULLA tools: ULLA result

name: channels
type: string
value: [1][1][1][1][1][1][1][1][1][1][1][1][1]
last updated: 1473 [ms]

name: t_period_ms
type: integer
value: 100
last updated: 1373 [ms]

name: cca_mean
type: structured
value:
-------------------------------------------------
|channel|select|results|      ms      |     ticks    |
|      1|     1|      0|   723565753|         8379|
|      2|     1|      0|   723564653|         6893|
|      3|     1|      0|   723564753|         7044|
|      4|     1|      0|   723564853|         7174|
|      5|     1|      0|   723564953|         7312|
|      6|     1|      0|   723565053|         7451|
|      7|     1|      0|   723565153|         7588|
|      8|     1|     58|   723565253|         7727|
|      9|     1|      0|   723565353|         7851|
|     10|     1|      0|   723565453|         7994|
|     11|     1|    447|   723565553|         8118|
```

```
|      12|      1|     257|    723565653|         8261|
|      13|      1|     730|    723565753|         8379|
-------------------------------------------------
last updated: 1 [ms]
```

Con questa interrogazione, richiediamo quali sono i canali selezionati per la scansione, il periodo di scansione per ogni canale e la misura del CCA medio secondo i precedenti parametri. Si noti come la misura del CCA riporti anche gli istanti temporali (rilevati da *CalRadio* dei tempi di scansione). Il valore numerico del CCA, riporta ha come unità di misura il tempo. Va poi opportunamente convertito il valore percentuale mediante la moltiplicazione per il fattore 0.00432.

### A.4.1 Update statement

Invece, per impostare un parametro si usa la clausola di update. Ad esempio, per selezionare i canali 1,3,7 e 11 per la scansione:

```
user@ulla ~ $ ullaAttribute "UPDATE ullaLinkProvider SET
    channels = '1 0 1 0 0 0 1 0 0 0 1 0 0 ' WHERE name = '
    template_lla ';"

ullaAttribute: UPDATE ullaLinkProvider SET channels = '1 0 1 0 0
     0 1 0 0 0 1 0 0 ' WHERE name = 'template_lla ';
ullaAttribute: urId 2
ULLA tools: ULLA result

name: channels
type: string
value: 1 0 1 0 0 0 1 0 0 0 1 0 0
last updated: 1 [ms]
```

## A.5 ULLACommand

Per inviare comandi *CalRadio*, si può utilizzare il tool ULLACommand. Ad esempio, per abilitare il meccanismo RTS/CTS:

```
user@ulla ~ $ ullaCommand "SELECT rts_cts_on FROM
    ullaLinkProvider WHERE name = 'template_lla ';"

ullaCmd: SELECT rts_cts_on FROM ullaLinkProvider WHERE name = '
    template_lla ';

name: rts_cts_on
type: integer
```

```
value: 0
last updated: 0 [ms]
```

Il valore restituito è 0, corrispondente a ULLA_OK, ovvero che l'operazione è andata a buon fine.

## A.6 ULLANotification e ULLASample

Il meccanismo di notifica e quello di campionamento permettono di ottenere statistiche circa uno o più parametri in modo continuativo, senza dover ripetere manualmente la query. Per brevità riportiamo solo il caso d'uso per il tool ULLASample.

```
user@ulla ~ $ ullaSamples "SELECT cca_mean FROM ullaLinkProvider
    WHERE name = 'template_lla';" 3 2000

ullaSample: ullaReceiveSample rnId 1
ullaSample: press <Crtl-C> to exit

ULLA tools: ULLA result

name: cca_mean
type: structured
value:
-------------------------------------------------
|channel|select|results|      ms      |     ticks    |
|      1|     1|      0|   724514660|          6560|
|      2|     0|      0|   724514360|          6099|
|      3|     1|      0|   724514460|          6248|
|      4|     0|      0|   724514460|          6248|
|      5|     0|      0|   724514460|          6248|
|      6|     0|      0|   724514460|          6248|
|      7|     1|      0|   724514560|          6421|
|      8|     0|      0|   724514560|          6421|
|      9|     0|      0|   724514560|          6421|
|     10|     0|      0|   724514560|          6421|
|     11|     1|    709|   724514660|          6560|
|     12|     0|      0|   724514660|          6560|
|     13|     0|      0|   724514660|          6560|
-------------------------------------------------
last updated: 0 [ms]


ullaSample: count 1
ULLA tools: ULLA result

name: cca_mean
```

```
type: structured
value:
-------------------------------------------------
|channel|select|results|      ms      |     ticks    |
|      1|     1|      0|   724516657|         7831|
|      2|     0|      0|   724516357|         7408|
|      3|     1|      0|   724516457|         7545|
|      4|     0|      0|   724516457|         7545|
|      5|     0|      0|   724516457|         7545|
|      6|     0|      0|   724516457|         7545|
|      7|     1|      5|   724516557|         7679|
|      8|     0|      0|   724516557|         7679|
|      9|     0|      0|   724516557|         7679|
|     10|     0|      0|   724516557|         7679|
|     11|     1|    720|   724516657|         7831|
|     12|     0|      0|   724516657|         7831|
|     13|     0|      0|   724516657|         7831|
-------------------------------------------------
last updated: 0 [ms]


ullaSample: count 2
ULLA tools: ULLA result

name: cca_mean
type: structured
value:
-------------------------------------------------
|channel|select|results|      ms      |     ticks    |
|      1|     1|      0|   724518657|         5878|
|      2|     0|      0|   724518357|         5463|
|      3|     1|      0|   724518457|         5596|
|      4|     0|      0|   724518457|         5596|
|      5|     0|      0|   724518457|         5596|
|      6|     0|      0|   724518457|         5596|
|      7|     1|      0|   724518557|         5730|
|      8|     0|      0|   724518557|         5730|
|      9|     0|      0|   724518557|         5730|
|     10|     0|      0|   724518557|         5730|
|     11|     1|    718|   724518657|         5878|
|     12|     0|      0|   724518657|         5878|
|     13|     0|      0|   724518657|         5878|
-------------------------------------------------
last updated: 0 [ms]


ullaSample: count 3
ULLA lib: rnId 1 auto-canceled
ULLA tools: [exit]  pid 22839, res 0
```

```
ullaSample: worker that with pid 22839 received 3 samples
ullaSample: [exit] pid 22839: 0
```

In questo caso sono stati chiesti 3 campioni, con periodo di campionamento di 2000 ms. Vediamo come il tool, riporti a ogni iterazione il numero di campioni già ottenuti e al raggiungimento del numero desiderato, il processo di auto-cancelli.

# Appendice B

# Link User - Interfaccia grafica

Nell'ambito della tesi, è stata sviluppata un'interfaccia grafica per la raccolta e la visualizzazione delle statistiche di ULLA. Dal punto di vista di ULLA, questa costituisce un'applicazione LU (link user).

Tale GUI è stata scritta in linguaggio Java.

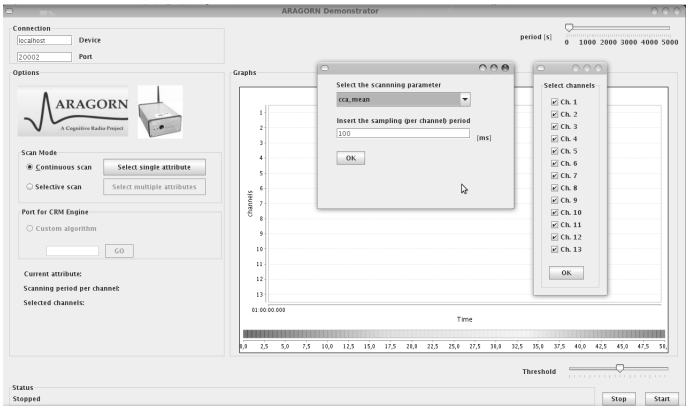
Per poter interagire con il framework ULLA, si è dovuta sviluppare un'applicazione server che permette ad un programma Java di utilizzare un programma scritto in C.

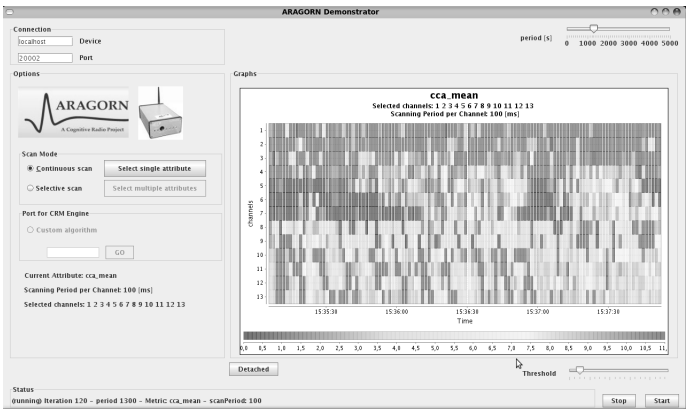
Compito del processo server, che viene eseguito nel pc con il framework ULLA, esegue un parsing dei comandi provenienti dalla GUI, richiamando direttamente le API fornite da ULLA. La comunicazione istanziata è di tipo TCP/IP, quindi affidabile.

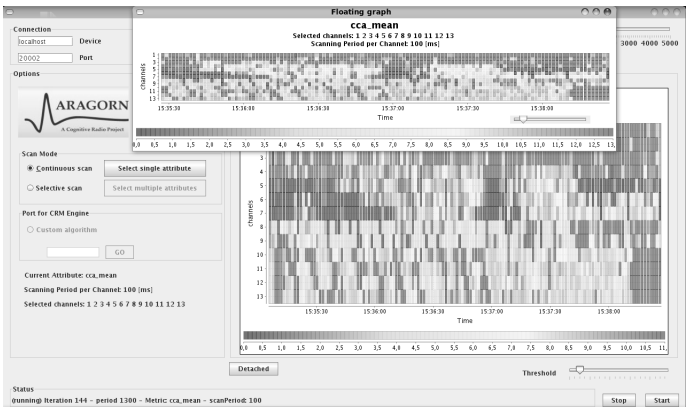
Il fatto di disporre di un'applicazione LU suddivisa in una parte client (GUI) e server, aiuta ad aumentare la modularità dell'intero sistema.Inoltre, grazie alla portabilità del linguaggio Java, è possibile governare ed interrogare il framework ULLA mediante un'interfaccia grafica che può essere eseguita da un qualunque computer che sia connesso alla rete internet.

Per quanto riguarda la visualizzazione grafica dei risultati, si è fatto uso della suite di librerie Open Source JFreeChart.

Le seguenti immagini mostrano la GUI creata e la sua operatività. Si ricorda che i risultati presentati nel Capitolo 4 della tesi, sono ottenuti grazie a tale applicazione.


ARAGORN Demonstrator
Connection
localhost
Device
20002
Port
Options
ARAGORN
A Cognitive Radio Project
Scan Mode
Continuous scan
Selective scan
Select single attribute
Select multiple attributes
Port for CRM Engine
Custom algorithm
GO
Current attribute:
Scanning period per channel:
Selected channels:
Graphs
period [s]
0 1000 2000 3000 4000 5000
Select the scannning parameter
cca_mean
Insert the sampling (per channel) period
100
[ms]
OK
Select channels
Ch. 1
Ch. 2
Ch. 3
Ch. 4
Ch. 5
Ch. 6
Ch. 7
Ch. 8
Ch. 9
Ch. 10
Ch. 11
Ch. 12
Ch. 13
OK
channels
Time
Threshold
Status
Stopped
Stop
Start


ARAGORN Demonstrator
Connection
localhost
Device
20002
Port
Options
ARAGORN
A Cognitive Radio Project
Scan Mode
Continuous scan
Selective scan
Select single attribute
Select multiple attributes
Port for CRM Engine
Custom algorithm
GO
Current Attribute: cca_mean
Scanning Period per Channel: 100 [ms]
Selected channels: 1 2 3 4 5 6 7 8 9 10 11 12 13
Graphs
period [s]
0 1000 2000 3000 4000 5000
cca_mean
Selected channels: 1 2 3 4 5 6 7 8 9 10 11 12 13
Scanning Period per Channel: 100 [ms]
channels
Time
Detached
Threshold
Status
(running) Iteration 120 - period 1300 - Metric: cca_mean - scanPeriod: 100
Stop
Start


Floating graph
cca_mean
Selected channels: 1 2 3 4 5 6 7 8 9 10 11 12 13
Scanning Period per Channel: 100 [ms]
channels
Time
Connection
localhost
Device
20002
Port
Options
ARAGORN
A Cognitive Radio Project
Scan Mode
Continuous scan
Selective scan
Select single attribute
Select multiple attributes
Port for CRM Engine
Custom algorithm
GO
Current Attribute: cca_mean
Scanning Period per Channel: 100 [ms]
Selected channels: 1 2 3 4 5 6 7 8 9 10 11 12 13
Detached
Threshold
Status
(running) Iteration 144 - period 1300 - Metric: cca_mean - scanPeriod: 100
Stop
Start

# Bibliografia


[1] Maravedis Market research and analysis. Clash of the titans – wimax and 4g: The battle for convergence is joined. 2006.

[2] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research.

[3] D. Kliazovich, F. Granelli, G. Pau, and M. Gerla. Apohn: Subnetwork layering to improve tcp performance over heterogeneous paths, April 2006.

[4] J. Mitola. *Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio*. PhD thesis, Royal Institute of Technology, 2000.

[5] Dzmitry Kliazovich, Fabrizio Granelli, and Nelson L.S. da Fonseca. Architectures and cross-layer design for cognitive networks.

[6] Christos H. Papadimitriou. Algorithms, games, and the interne. *Proceeding of STOC 2001*, 2001.

[7] Ryan W. Thomas, Luiz A. DaSilva, and Alle B. MacKenzie. Cognitive networks. IEEE, June 2007.

[8] J. Mitola and G. Q. Maguire. Cognitive radio: Making software radios more personal. IEEE Pers. Commun, vol. 6, pp. 13–18, 1999.

[9] Andrea Goldsmith, Syed Ali Jafar, Ivana Maric, and Sudhin Srinivasa. Breaking spectrum gridlock with cognitive radio: An information theoretic perspective. *Proceeding of the IEEE*, 97(5), May 2009.

[10] Arpita Guha and Viswanath Ganapathy. Power allocation schemes for cognitive radios. 2008.

[11] Friedrich K. Jondral. Software-defined radio-basics and evolution to cognitive radio. *EURASIP Journal on Wireless Communications and Networking*, 3:275–283, February 2005.

[12] Ryan W. Thomas, D. H. Friend, Luiz A. DaSilva, and Alle B. MacKenzie. Cognitive networks: adaptation and learnig to achieve end-to-end performace objectives. *IEEE Communications magazine*, 44(12):51–57, December 2006.

[13] E. Gelenbe, Zhiguang Xu, and E. Seref. Cognitive packets networks. *IEEE International conference on tools with Artificial Intelligence*, pages 47–54, November 1999.

[14] S. M. Lake Sr. Cognitive networking with software programmable intelligent networks for wireless and wireline critical communications. *IEEE Military Communications Conference (MILCOM)*, pages 1693–1699, October 2005.

[15] Microsoft TechNet. New networking features in windows server 2008 and windows vista. http://technet.microsoft.com/enus/library/bb726965.aspx, February 2008.

[16] Progetto gollum. http://www.ist-gollum.org/.

[17] http://ulla.sourceforge.net/.

[18] Udae for wsn. http://ulla.sourceforge.net/UDAE.html.

[19] Atheros. http://www.atheros.com/.

[20] Madwifi driver. http://www.madwifi.org/.

[21] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers, 3rd Edition.* O'Reilly Media, Inc., 2005.

[22] Calit2 - calradio. http://calradio.calit2.net/index.htm.

[23] Uclinux. http://www.uclinux.org/.

[24] busybox. http://busybox.net/.

[25] Intersil. *Direct Sequence Spread Spectrum Baseband Processor with Rake Receiver and Equalizer*, December 2001.

[26] Maxim. *2.4 GHz 802.11b Zero-IF Transceivers*, April 2004.

[27] P. Fuxjager, D. Valerio, and F. Ricciato. The myth of non-overlapping channels: interference measurements in ieee 802.11. pages 1–8, jan. 2007.

[28] Tmote sky. http://www.moteiv.com/.

[29] G.M. Tamiselvan and Dr. A. Shanmugam. Probability analysis of channel collision between ieee 802.15.4 and ieee 802.11b using qualnet simulation for various toppologies. *International Journal of Computer Theory and Engineering*, 1(1), April 2009.

[30] Specification of bluetooth system. http://www.bluetooth.com, 2003.

[31] R.J. Punnoose, R.S. Tseng, and D.D. Stancil. Experimental results for interference between bluetooth and ieee 802.11b dsss systems. volume 1, pages 67–71 vol.1, 2001.

[32] S. Zeadally, A. Banda, and A. Kumar. On the coexistence of bluetooth and ieee 802.11 technologies in real networking environments. pages 269–273, june 2003.

[33] B. Zielinski. Ieee 802.11 network behaviour in the presence of bluetooth network. pages 18–18, aug. 2007.

[34] Tsung-Chuan Huang and Shao-Hsien Chiang. Coexistence mechanisms for bluetooth sco link and ieee 802.11 wlan. volume 2, pages 424–431, nov. 2006.

[35] Riccardo Manfrin. *Calradio Documentation*, January 2009.

[36] I. Ramachandran and S. Roy. Wlc46-2: On the impact of clear channel assessment on mac performance. pages 1 –5, 27 2006-dec. 1 2006.

[37] G. Anastasi, E. Borgia, M. Conti, and E. Gregori. Wi-fi in ad hoc mode: a measurement study. pages 145 – 154, march 2004.

[38] N. Baldo and M. Zorzi. Cognitive network access using fuzzy decision making. pages 6504 –6510, june 2007.

[39] N. Baldo, B.R. Tamma, B.S. Manojt, R. Rao, and M. Zorzi. A neural network based cognitive controller for dynamic channel selection. pages 1 –5, june 2009.