ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Corso di Laurea Triennale in Informatica

OTTIMIZZAZIONE DELLA SCANSIONE WIFI
Responsiveness

Tesi di Laurea in Architettura degli Elaboratori

Tesi di laurea di:
Mattia Guberti

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Sessione III
Anno Accademico 2011/2012

Indice

1. Introduzione
2. Le Reti Wireless
    2.1 Tecnologia di trasmissione: Access Point
    2.2 Reti Ad-Hoc
    2.3 Wi-Fi
        2.3.1 Struttura del frame MAC 802.11
        2.3.2 Beacon Frame
    2.4 Scelta del miglior Access Point
3. Modello di scansione Wi-Fi
    3.1 Scansione Attiva
        3.1.1 Probe Diretto
        3.1.2 Probe in Broadcast
    3.2 Scansione Passiva
    3.3 Rapporto fra scansione e traffico di dati
        3.3.1 Scansione in BackGround
        3.3.2 Scansione con traffico
    3.4 Caso di studio
        3.4.1 Possibilità di scandire un solo canale
        3.4.2 Check dei dati latenti al termine di ogni canale scandito
        3.4.3 Scansione canale per canale con controllo del traffico
    3.5 Conclusioni e considerazioni

4. Driver e Kernel
    4.1 L'interfaccia di rete
        4.1.1 Broadcom wireless
    4.2 Gestione della scansione nei moduli del kernel
        4.2.1 scan.c
        4.2.2 offchannel.c
5. Sleep Mode
    5.1 Null Frame Data
    5.2 Beacon Interval e Tim window (Atim window)
        5.2.1 Struttura dell'elemento TIM
    5.3 Power Save Mode con connessione di tipo Infrastructure
    5.4 Power Save Mode con connessione Ad-Hoc
6. Test e valutazioni
    6.1 Test
    6.2 Monitor Mode
7. Considerazioni e sviluppi
    7.1 Considerazioni
    7.2 Funzionalità del sistema
    7.3 Linee guida per l'implementazione
    7.4 Tips
    7.5 Pro e contro
8. Conclusioni

Bibliografia

Capitolo 1
Introduzione

Al giorno d'oggi, in seguito ad una decisa espansione della tecnologica, sono aumentati sensibilmente le reti wireless disponibili nelle nostre città.
D'altro canto, abbiamo assistito alla larga diffusione di nuovi dispositivi in grado di collegarsi ad una rete wireless come smartphone, tablet o PC portatili.
I comuni di alcune grandi città forniscono un servizio che permette di accedere liberamente, attraverso i propri dispositivi, ad una rete pubblica, accessibile in un'area più o meno estesa.
Gli utenti hanno quindi la possibilità di effettuare traffico di dati, non solo in maniera statica, ma anche muovendosi all'interno di vaste aree.
Si pone quindi la necessità di permettere la selezione di una rete, basandosi per esempio su parametri quali potenza di segnale o quantità di traffico.
Tutto questo è semplice quando il device in questione è fisso in un punto, ma diventa più complicato nel momento in cui questo si muove.
Infatti i parametri delle reti wireless captate possono cambiare notevolmente: in base alla distanza dall'Access Point (AP), se ci avviciniamo ovviamente migliorano, mentre in caso contrario si indeboliscono, in base al numero di ostacoli presenti fra il device e l'AP, o ancora in base alla quantità di reti wireless che sfruttano la stessa frequenza o frequenze molto simili.
Per valutare correttamente questi parametri, è necessaria una scansione periodica dell'area da parte del device.
Questa scansione potrebbe essere utile anche nel caso in cui il device sia già associato ad un

AP e stia già effettuando traffico, al fine di migliorare le prestazioni della trasmissione dei dati. Come si può intuire, una scansione, in quest'ultimo caso, potrebbe essere di intralcio per le operazioni di comunicazione fra device e AP, causando un eccessivo delay nella trasmissione o ricezione dei pacchetti di dati, o nella peggiore delle ipotesi, la completa perdita di tali pacchetti.
Ci siamo perciò proposti di studiare un meccanismo che riduca al minimo il ritardo o la perdita di pacchetti, garantendo comunque la possibilità di scandire in modo dinamico le reti, per permettere l'ottimizzazione delle prestazioni del traffico di dati.
Ci eravamo posti come obiettivo finale quello di progettare un sistema in grado di effettuare il tipo di scansione appena descritto, memorizzando la posizione dei nuovi AP incontrati, mentre si è in movimento all'interno di un'area.

Dal momento che questo progetto è stato svolto da due persone, si è cercato di suddividerlo in due sottoprogetti:

1. Studio delle tecniche di scansione
2. Analisi dei meccanismi di Power Save Mode (PSM) per la comunicazione con l'AP

La sottoparte di cui mi sono occupato maggiormente riguarda l'analisi delle tecniche adottate dal kernel Linux (versione 3.6.8) per effettuare la scansione, distinguendo scansione attiva e passiva, e prestando particolare attenzione a ciò che avviene nel modulo *scan.c*. Nell'analizzare questo modulo il mio intento è stato quello di capire nel dettaglio secondo quali politiche venga cambiato il canale corrente da scandire, che controlli vengano effettuati prima di iniziare la scansione di ogni canale e l'analisi del beacon che l'AP invia ai device periodicamente. La parte di testing invece è stata effettuata completamente in collaborazione con Federico Baldini.

Passeremo ora a spiegare in generale che cosa verrà trattato nei capitoli seguenti.
Nel capitolo 2 verrà descritto l'ambiente nel quale andremo a lavorare, in particolare le tecniche di comunicazione wireless, le tecnologie e le strutture dati utilizzate.
Nel capitolo 3 verranno descritti i metodi di scansione delle reti WiFi, passaggio fondamentale per l'introduzione del modello di scansione da noi proposto.

Nel capitolo 4 verranno descritte le differenze fra i diversi tipi di driver utilizzati dalle interfacce di rete, con particolare interesse per i moduli del kernel che si occupano della scansione e del Power Save Mode (PSM).
Nel capitolo 5 spiegheremo nel dettaglio il meccanismo di PSM, illustrando le tecniche risveglio dei device nelle reti di tipo Infrastructure e Ad-Hoc.
Nel capitolo 6 verranno descritti i test effettuati al fine di comprendere e confermare quanto studiato precedentemente a livello teorico.
Nel capitolo 7 infine presenteremo le nostre considerazioni riguardo la possibilità di realizzare in futuro un'applicazione che implementi i meccanismi di scansione da noi suggeriti, valutandone eventuali pro e contro e fornendo suggerimenti.

Capitolo 2
Le reti Wireless

Per comunicazione wireless si intende il trasferimento di informazioni lungo una certa distanza senza l'utilizzo di cavi. La distanza interessata può variare da pochi metri (come nel caso di un telecomando per una TV) a migliaia di chilometri (come nel caso delle comunicazioni radio). Le comunicazioni wireless sono generalmente considerate come un ramo delle telecomunicazioni. Questo tipo di tecnologia è utilizzata in svariati ambiti tra cui telefonia cellulare, dispositivi PDA e comunicazioni di rete tra computer.

Le reti wireless, quindi, possono avere diversi utilizzi, ma l'uso più comune è quello di permettere la connessione di utenti di computer che si spostano da un posto all'altro. Le possibili situazioni, che giustificano l'utilizzo di tale tecnologia in questo ambito, sono le seguenti:

- espandere la distanza di una tipica rete cablata
- fornire una comunicazione di riserva nel caso di un malfunzionamento della rete
- collegare postazioni portatili o temporanee
- coprire situazioni dove il normale cablaggio è difficile o troppo costoso
- connettere a distanza utenti o reti mobili

2.1 Tecnologia di trasmissione: Access Point

Un Access Point (AP) solitamente può comunicare con poche decine di client situati in un raggio di un centinaio di metri. Tuttavia questa caratteristica varia a seconda di diversi fattori quali: il posizionamento, la presenza di ostacoli, l'interferenza con altri campi elettromagnetici e la potenza dei dispositivi. E` possibile però aumentare la distanza di trasmissione mediante l'utilizzo di ripetitori che amplificano i segnali radio.

Le reti Wi-Fi utilizzano onde radio per la trasmissione dei dati, in modo da permettere la realizzazione di reti in ambienti eterogenei dove spesso sono presenti pareti tra le postazioni, rendendo di fatto possibile la creazione di intere zone metropolitane completamente coperte dalla rete.

Un qualsiasi dispositivo Wi-Fi (come un personal computer, una console per video giochi, un telefonino, ecc) può accedere a Internet quando è nel raggio d'azione di una rete wireless collegata. La presenza di uno o più AP, chiamati "hotspot", è in grado di creare zone completamente connesse con una dimensione che va da poche stanze fino a un'intera città.

2.2 Reti Ad-Hoc

Wi-Fi permette anche la comunicazione diretta tra due dispositivi senza l'utilizzo di AP. Questa modalità è chiamata "Ad-Hoc". Le reti Ad-Hoc permettono quindi a due apparecchi di entrare in comunicazione tra di loro senza l'utilizzo di una rete esistente e senza nessun cavo. Questo tipo di utilizzo è spesso usato da console per videogiochi per creare una modalità multigiocatore.

2.3 Wi-Fi

Lo standard IEEE che definisce le specifiche di una rete wireless, detta anche WLAN, è IEEE 802.11. La diffusione delle reti di questo tipo ha reso necessario la fondazione di una sorta di certificazione che attesti che un dispositivo è in grado di operare con una rete WLAN. Per questo nel 1999 la Wi-Fi Alliance, un'associazione non-profit di

aziende che promuovono la tecnologia WLAN e certificano i prodotti conformi a certi standard di interoperabilità, ha fondato il marchio "Wi-Fi", abbreviazione di "WIreless FIdelity". Un dispositivo Wi-Fi è in grado di operare in una qualsiasi rete basata sulle specifiche IEEE 802.11. Per ottenere il logo Wi-Fi però, non è sufficiente rispettare le linee guida dello standard, ma è necessario superare una serie di procedure di certificazione stabilite dal consorzio Wi-Fi Alliance, denominate "Wireless Ethernet Compatibility Alliance". A causa dei costi di questo processo di certificazione, non tutte le aziende sono in grado di ottenere il marchio Wi-Fi, quindi l'assenza del logo non implica necessariamente che un dispositivo non sia in grado di operare in una rete certificata Wi-Fi.

2.3.1 Struttura del frame MAC 802.11

Ogni frame è composto da:

- MAC header
- frame body: di lunghezza variabile, contiene informazioni specifiche al frame (tipo e sottotipo);
- FCS: che contiene un CRC di 32 bit.

Il MAC header è lungo 32 bit e comprende frame control, duration, address, sequence control e, per i frame QoS4, QoS control. Il formato del frame è il seguente:

Byte	2	2	6	6	6	2	6	2	0 - 23124	4
	Frame Control	Durata	Indirizzo 1	Indirizzo 2	Indirizzo 3	Sequence Control	Indirizzo 4	QoS Control	Body	FCS

Table: *Il frame IEEE 802.11*

b0 b1	b2 b3	b4 - b7	b8	b9	b10	b11	b12	b13	b14	b15
Protocol Version	Type	Subtype	ToDS	FromDS	More Fragments	Retry	Power M.	More Data	Protected Frame	Order

Table: *Frame Control*

Immagine 1: Il frame MAC IEEE 802.11

I primi tre campi e l’ultimo sono la base minimale e fanno parte di tutti i tipi di frame, mentre i restanti sono presenti solo in alcuni. Il primo campo detto frame control è composto da undici campi secondari.

2.3.2 Beacon Frame

Il beacon frame è uno dei frame di gestione delle reti WLAN.

Questi frame sono trasmessi periodicamente per annunciare la presenza di una rete wireless e vengono trasmessi dagli AP.

Il beacon frame è composto da un MAC header, un frame body e FCS. Elencheremo ora alcuni dei suoi campi:

- Timestamp: dopo la ricezione di un beacon frame, tutti i device impostano il loro local clock a questo valore
- Beacon interval: questo è l'intervallo di tempo che intercorre tra la trasmissione di due beacon
- Capability information: questo campo ha una dimensione di 16 bits e contien informazioni riguardanti le capacità del device/network. Il tipo di network come ad esempio Ad-Hoc o infrastruttura è segnalato in questo campo.
- SSID
- Supported-rates
- Frequency-hopping (FH)
- Direct-sequence (DS)
- Contention-free (CF)
- IBSS
- Traffic Indication Map (TIM)

I campi TIM e Beacon interval verranno spiegati approfonditamente nel capitolo 5.

2.4 Scelta del miglior Access Point

La politica di selezione basata solo sulla potenza di segnale è inadeguata in quanto non assicura che il nodo scelto sia in grado di fornire un servizio migliore rispetto agli altri AP presenti nella zona. Per scegliere un buon AP al quale associarsi è opportuno valutare anche altri fattori. A tale scopo il nostro esistono software che stimano anche un indice di qualità (Quality Index), basandosi su tutto quello che può influire sulla qualità del collegamento.

Capitolo 3
Modello di scansione Wi-Fi

La ricerca di nuove reti WiFi necessita un meccanismo di scansione, gestito dal kernel, che ispeziona i canali disponibili, restituendo le informazioni degli Access Point (AP) accessibili attraverso una rete WiFi.
La scansione dei canali si differenzia in due diversi tipi:

- scansione attiva
- scansione passiva

3.1 Scansione Attiva

La scansione attiva si verifica quando il client modifica la propria frequenza (dello standard IEEE802.11) per spostarsi sul canale che deve scandire, invia quindi un probe di richiesta in broadcast ed attende di ricevere dei probe di risposta (o di beacon periodici) dagli AP su quel canale (che abbiano un SSID corrispondente a quello inserito nel probe).
Lo standard IEEE 802.11 non specifica quanto a lungo il client debba attendere, ma il periodo tipico è di 10 ms. Il probe di richiesta utilizzato in una scansione attiva è uno dei due seguenti:

3.1.1 Probe diretto

Il client invia un probe di richiesta con un SSID di destinazione specifico; soltanto gli AP con il SSID corrispondente risponderà con il probe di risposta.

3.1.2 Probe in Broadcast

Il client invia un SSID broadcast (un SSID impostato a null) nel probe di richiesta; tutti gli AP che ricevono tale probe risponderanno con un probe di risposta per ogni SSID che supportano.

3.2 Scansione Passiva

La scansione passiva viene eseguita semplicemente cambiando la frequenza (dello standard IEEE 802.11) del client sul canale che si sta scandendo ed attendendo un beacon periodico da qualsiasi AP presente su quel canale. Di default, gli AP inviano un beacon ogni 100 ms.

Poiché potrebbero dover attendere 100 ms per ricevere un beacon periodico in broadcast, la maggior parte dei client predilige la scansione attiva.

3.3 Rapporto fra scansione e traffico di dati

Un problema riscontrabile durante la scansione di un canale è che il client non è in grado di trasmettere o ricevere dati.

Ci sono differenti approcci, da parte del client, per minimizzare questo impatto sul traffico:

- Scansione in Background
- Scansione con traffico

3.3.1 Scansione in Background

Il client potrebbe fare la scansione dei canali disponibili prima ancora di essere associato ad un AP.

Questo permette di creare un ambiente che sia a conoscenza degli AP, in tal modo il client ha la possibilità, se necessario, di avere uno scambio di dati più rapido.

L'impatto sul traffico dati del client può essere minimizzato ad esempio facendo la scansione quando il client non sta trasmettendo attivamente dei dati, oppure facendo scansioni periodiche sui singoli canali in una sola volta (infatti la scansione di un singolo canale alla volta causa una minima perdita di dati).

3.3.2 Scansione con traffico

In alternativa a quella in background, la scansione con traffico attivo avviene quando il device è già collegato ad un AP e di conseguenza vi è già trasmissione di pacchetti.
Ogni casa produttrice per ogni singolo device, potrebbe implementare i propri algoritmi per minimizzare la latenza di dati e l'impatto della scansione sul traffico di dati; per esempio, alcuni client potrebbero scandire soltanto i canali non sovrapposti [0].

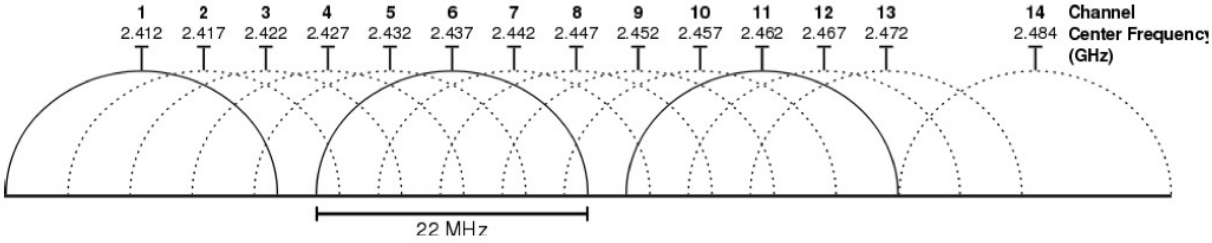

Immagine 2: Frequenza dei canali wireless

3.4 Caso di studio

Il caso di cui ci siamo occupati è quello in cui il nostro device sia già connesso ad un AP, ma richieda una scansione delle reti WiFi disponibili.
I motivi possono essere diversi: ricerca di una rete con un segnale migliore, ricerca di una rete con un traffico di dati inferiore[1], o ancora, memorizzare una sorta di mappa degli AP incontrati.

Le politiche del kernel Linux prevedono la scansione di tutti i canali e non consentono una customizzazione della ricerca di questi da parte dell'utente, attraverso applicazioni ad alto livello.

Nel caso il device abbia traffico di rete attivo in uscita o in entrata, una scansione potrebbe, come sottolineato in precedenza, arrecare problemi, causando il ritardo o addirittura la perdita di pacchetti.
Per ovviare a questo problema, è stato creato un sistema di ottimizzazione, chiamato “Sleep

Mode", che notifica all'AP dell'imminente scansione ed introduce una sincronizzazione fra nodo mobile e AP atta appunto a non perdere nessun pacchetto da ricevere.
Ci occuperemo più approfonditamente dello Sleep Mode nei prossimi capitoli.
Nonostante questi accorgimenti permettano di non disperdere i dati, rimane impossibile il controllo del numero di canali scanditi, quindi il delay di trasmissione, considerando un tempo di 100 ms per canale, per 13 canali, rimane nell'ordine del secondo per scansione.

L'idea che ci siamo proposti di studiare è quindi quella di un sistema che permetta la scansione dei canali con customizzazione lato utente, in modo da fornire i tools necessari per garantire una trasmissione più lineare dei dati.
Idee base di implementazione del sistema:

- Possibilità di scandire un solo canale
- Check dei dati latenti al termine di ogni canale scandito
- Scansione canale per canale con controllo del traffico

3.4.1 Possibilità di scandire un solo canale

Seppur non esistano programmi ad alto livello che forniscono questa funzione, abbiamo studiato il codice del kernel Linux e abbiamo potuto constatare che il codice stesso del sistema operativo opera una scansione su ogni canale, incrementando la variabile che rappresenta il canale da scandire al termine della funzione di scansione di un singolo canale. Questo ovviamente sottolinea come sia possibile, implementando una sincronizzazione fra applicazione e kernel, gestire una scansione su un singolo canale, selezionando la frequenza di quest'ultimo.

3.4.2 Check dei dati latenti al termine di ogni canale scandito

Un altra possibilità che fornisce il kernel, è quella di "dialogare" con l'AP per controllare se vi siano pacchetti latenti, bufferizzati durante la scansione. Sarebbe pertanto possibile implementare il nostro sistema in modo da richiedere eventuali pacchetti latenti nei momenti immediatamente precedente e successivo alla scansione del canale.

3.4.3 Scansione canale per canale con controllo del traffico

Fissati i due punti precedenti, è semplice prevedere una scansione canale per canale che esegua il controllo di pacchetti latenti ad ogni incremento del canale.
In questo modo un eventuale scambio di dati avrà delay di minor durata (nell'ordine dei 100 millisecondi) e permetterà di conseguenza un flusso più omogeneo.

3.5 Conclusioni e considerazioni

Le funzionalità del modello presentate sopra sono chiaramente molto utili all'utente, abbiamo perciò sviluppato degli studi di fattibilità ed eseguito test sul codice del kernel per comprendere a pieno il funzionamento e l'implementazione dello Sleep Mode; nel capitolo 5 ne evidenzieremo gli aspetti fondamentali.

Capitolo 4
Driver e Kernel

Nei sistemi operativi spesso i driver delle interfacce di rete implementano algoritmi di gestione della scansione che differiscono da quelli implementati nei moduli del kernel.

Se il kernel è modulare, come quello di Linux con il quale abbiamo lavorato, la maggior parte dei driver per l'hardware della macchina risiedono sull'hard disk e sono disponibili come moduli. All'avvio, udev fa un inventario dell'hardware presente. Lo stesso udev carica quindi i moduli (i driver) per l'hardware corrispondente, e il driver, di contro, permette la creazione di una kernel interface.

4.1 L'interfaccia di rete

Il nome dell'interfaccia varia in base al driver e al chipset installato.
Alcuni esempi sono wlan0, eth1, o ath0.
Nota: udev non è perfetto. Se il modulo appropriato non è caricato all'avvio, basta attivarlo tramite il comando modprobe e aggiungerlo in un file .conf in /etc/modules-load.d/ . Si noti inoltre che udev può occasionalmente caricare più di un driver per una periferica, e il conflitto risultante impedirebbe il successo della configurazione.
Questo particolare mette in risalto nuovamente come ogni macchina possa avere differenti driver installati e di conseguenza come possa seguire differenti politiche di scan.
Nel nostro caso di studio, i moduli caricati dall'interfaccia erano quelli relativi ai driver Broadcom wireless.

4.1.1 Broadcom wireless

Broadcom è famosa per il suo supporto per la sua scheda Wi-Fi su GNU/Linux.

Fino ad ora, la maggior parte dei chip erano completamente non supportati oppure richiedevano all'utente di lavorare con i firmware. Soltanto un numero limitato di chip wireless era supportato da diversi driver come brcm4xxx, b43, etc.

Il b43 è stato inserito nel kernel fin dalla versione 2.6.24.

Nell'agosto 2008 Broadcom ha rilasciato il "802.11 Linux STA driver" che supportava ufficialmente l'hardware Broadcom wireless sul sistema GNU/Linux. Questi driver avevano una licenza riservata, ma Broadcom ha promesso di lavorare in proposito per un approccio più aperto in futuro.

Nel settembre 2010, Broadcom ha rilasciato la release definitiva dei driver open source per il suo driver. Questo driver, **brcm80211,** è stato incluso nel kernel dalla versione 2.6.37. Con la release 2.6.39, questi driver sono stati rinominati **brcmsmac** e **brcmfmac**.

4.2 Gestione della scansione nei moduli del kernel

Per studiare il comportamento della scansione in modo più generale possibile, evitando le peculiarità che contraddistinguono i diversi driver, abbiamo utilizzato come interfaccia di rete, una chiavetta USB predisposta per caricare i soli moduli del kernel, senza inserire alcun driver esterno.
Creata questa situazione generica, abbiamo potuto approfondire la ricerca sul codice sorgente del kernel 3.6.8 di Linux; ci siamo in particolare focalizzati sui seguenti moduli:

- scan.c
- offchannel.c

tutti all'interno della cartella /net/mac80211/

4.2.1 scan.c

Questo modulo è il centro nevralgico della scansione.
Abbiamo studiato le principali funzioni che implementano questo meccanismo, avvicendando la ricerca di AP sui vari canali.

Per meglio comprendere il funzionamento, ci siamo ulteriormente concentrati su alcune funzioni che abbiamo ritenuto più importanti:

- ieee80211_scan_work:

Possiamo dire che questa funzione sia la principale del modulo **scan.c**, poiché non è definita come static ed è quindi richiamata dall'esterno, dai moduli che vogliono interagire con l'interfaccia facendo una scansione delle reti.
Quando viene chiamata, controlla che tutto sia pronto per la scansione, successivamente, attraverso la funzione __ieee80211_start_scan, valuta la possibilità di eseguire una scansione hardware, altrimenti effettua una scansione software. Ha inoltre il compito di valutare se il device sia già connesso o meno ad un canale, gestendo, nell'ultimo caso, il PSM (Power Save Mode).
Le funzioni sottostanti vengono richiamate all'interno di questa funzione, all'interno di una serie di SWITCH/CASE.

- ieee80211_start_sw_scan:

Nel caso non sia possibile una scansione hardware, il kernel implementa una sua scansione a livello software. Nel caso il device sia già connesso ad un AP, la prima cosa eseguita da questa funzione è una notifica all'AP dell'intenzione di entrare in PSM, inviando un Null Data Frame (nullframe data).
Tale frame presenta un bit di controllo, all'interno del campo Power Management, che puo essere settato a 0 o 1.
Nel primo caso informa l'AP di essere awake e quindi la trasmissione di pacchetti dovrebbe essere normale, altrimenti notifica l'intento di entrare in PSM, modificando il comportamento dell'AP nel trasmettere i pacchetti ad esso diretti.
Discuteremo il funzionamento di questo meccanismo in maniera più approfondita nel capitolo seguente.
Infine, la funzione fa partire le routine che eseguono la scansione vera e propria.

- ieee80211_scan_state_set_channel:

Come già si può intuire dal nome della funzione, il suo compito è quello di impostare

correttamente il canale da scandire.
In seguito, se l'AP ha pacchetti pendenti da inviare al device, la scansione potrebbe essere sospesa, per permettere la ricezione dei dati, prima di riprenderla.
Altrimenti, verrà incrementato il numero del canale da scandire e le operazioni di scansione potranno continuare.

- ieee80211_scan_state_send_probe:

Questa funzione agisce in modo differente nel caso di scansione attiva o passiva.
Nel primo caso, verrà effettuato l'invio di probe e il device attenderà la risposta da parte degli AP ricercati su quel canale, nel secondo invece si metterà semplicemente in attessa di ricevere i beacon da parte degli AP.

4.2.2 offchannel.c

In questo modulo viene gestito il sistema di Power Save, che verrà approfondito nel prossimo capitolo inerente allo Sleep Mode.
Abbiamo riscontrato un ruolo fondamentale in questo modulo poiché molte delle funzioni qui implementate, vengono richiamate durante la scansione, per sincronizzare il nodo mobile nel caso in cui sia già connesso ad un AP e vi sia un traffico di dati.
Le funzioni principali su cui abbiamo posto la nostra attenzione sono:

- ieee80211_offchannel_ps_enable:

Dopo aver correttamente sincronizzato il timer del Beacon Interval, questa funzione di occupa di inviare realmente il Nullfunc Frame all'AP, impostando il flag a 1.

- ieee80211_offchannel_ps_disable:

Questa funzione, al contrario di quella precedentemente descritta, informa l'AP di ritornare in stato awake, impostando il flag del Nullfunc Frame a 0.

- ieee80211_offchannel_stop_vifs:

Questa funzione notifica all'AP che il device sta per lasciare il canale.

Tale operazione avviene ogni qualvolta il device inizi o riprenda una scansione.
In seguito vengono disattivate le interfacce WiFi e viene deciso se abilitare o disattivare il beaconing.

- ieee80211_offchannel_return:

Al termine della scansione, o quando viene invocata la scan_suspend, il device si occupa di informare l'AP di essere ritornato sul canale operativo.
In questo modo le operazioni ordinarie di scambio di pacchetti possono riprendere correttamente.

CAPITOLO 5
Sleep Mode

Tra le fonti di consumo di batteria da parte di un device wireless vi sono la ricezione e la trasmissione di pacchetti, e si ha consumo anche quando il device è inattivo. Ovviamente in quest'ultimo caso si avrà un consumo di energia inferiore rispetto a quando il device effettua una ricezione o trasmissione di dati. Nonostante ciò, il consumo di energia da parte di un dispositivo inattivo non è assolutamente trascurabile. Per esempio, per il protocollo IEEE 802.11, un device inattivo rimane comunque in ascolto in continuazione sul canale wireless per testare lo stato del canale stesso, oltre che per controllare che ci sia qualche altro host intenzionato ad iniziare una trasmissione di pacchetti. Con ciò notiamo che anche se un device è inattivo, rimane comunque impegnato in attività utili, che possono consumare quantità di energia non trascurabili.

Per permettere ai dispositivi wireless di conservare energia, un modo alternativo all'inattività è offerto dalla possibilità di mandare il device in “sleep mode”. Durante la sleep mode, il device wireless non può nè trasmettere nè ricevere alcun pacchetto, e nemmeno sondare lo stato del canale.

In questo capitolo utilizzeremo il termine “power save mode” (PSM) per riferirci al meccanismo che permette ai device di entrare in sleep mode, in cui in pratica vengono disattivati sia il trasmettitore che il ricevitore per un piccolo periodo di tempo al fine di conservare batteria.

Si può constatare un grande risparmio di batteria quando i dispositivi utilizzano questo meccanismo. Ad esempio molti telefoni che utilizzano una tecnologia VoIP beneficiano di

questo sistema. E' però necessario stare attenti che alcune applicazioni potrebbero risentire in negativo di una politica di sleep troppo aggressiva.
Come già accennato, lo svantaggio è che un device in sleep mode non può comunicare con nessun altro device. Perciò, il meccanismo di "power save" necessita di un sistema che sia in grado di "risvegliarlo" quando necessario.
Prima di spiegare quali siano questi sistemi che permettono di risvegliare un dispositivo dalla sleep mode, introduciamo un elemento fondamentale per comprendere quanto segue: il Null Data Frame (o Nullframe).

5.1 Null Frame Data

Il Null Data Frame è un frame di controllo che viene trasmesso solamente dai dispositivi (client wireless), e non dagli access point. Questo frame non ha payload. Infatti, il suo unico scopo è di trasportare il bit di power management nel campo di controllo del frame. Il power management bit può essere settato sia a 0 che a 1.
Quando viene inviato il Nullframe con il bit impostato a 0 all'access point (AP) a cui si è associati, si vuole comunicare che il device è awake (ovvero attivo) e che quindi la trasmissione di frame da parte dell'access point dovrebbe essere normale.
Quando il device invece invia il Nullframe con il bit impostato a 1, allora stiamo informando l'AP che stiamo andando in sleep e che quindi dovrebbe bufferizzare i pacchetti indirizzati al device fino all'invio di un nuovo Nullframe di valore 0.

I due motivi principali per cui un dispositivo invia un Nullframe all'AP sono:

1) Power Save Mode (PSM): il PSM permette al device di andare in sleep e quindi risparmiare energia.

2) Scansione Attiva/Passiva: l'altra ragione per cui potrebbe essere inviato un Nullframe è quando il device è pronto ad effettuare una scansione. Supponiamo che un client voglia effettuare una scansione dell'area, per trovare tutti gli access point è necessario uscire dal

canale attuale (ovvero andare “offchannel”), per far ciò è necessario informare l'AP a cui siamo associati che non saremo disponibili a ricevere frame per un certo periodo di tempo.

5.2 Beacon Interval e Tim window (Atim window)

Il Target Beacon Transmission Time (TBTT) è il tempo al quale un nodo deve spedire un beacon (AP o device in caso di connessione Ad-Hoc). La differenza di tempo tra due TBTTs è detta beacon interval. Il beacon interval è misurato in Time Units (TU), ogni TU rappresenta 1024 microsecondi. Il beacon interval è tipicamente impostato a 100 TUs (102400 microsecondi, ovvero 102.4 ms) .

Quando un device decide di sfruttare il PSM il tempo viene suddiviso in questi beacon interval, l'istante di inizio del beacon interval viene sincronizzato tra tutti i nodi connessi.

Ogni nodo rimane sveglio per un piccolo lasso di tempo all'inizio del beacon interval. Questo periodo è detto Tim window, o Atim window. Tim è l'abbreviazione di “Traffic information Map” e viene untilizzata quando si trattano connessioni di tipo Infrastructure. Atim è invece l'abbreviazione di "Ad-hoc Traffic information Map" che come si può facilmente intuire viene usato nell'ambito delle connessioni Ad-hoc.

La Traffic information Map è formata da 2008 bits, ciascuno dei quali rappresenta l'Association Id (AID) di un dispositivo. Il TIM information element permette di trasferire da 1 byte fino ad un massimo di 251 bytes, che è la dimensione del TIM stesso. Si è quindi abilitati a trasmettere una TIM bitmap più piccola se ci aspettiamo che ci siano pochi device che andranno in sleep. Perciò il valore della bitmap passata nel TIM è chiamata “Partial Virtual Bitmap”. Per poter trasmettere solamente una Partial Bitmap si devono settare in modo adeguato i campi del bitmap control e il campo relativo alla dimensione del TIM.

5.2.1 Struttura dell'elemento TIM:

[ID elemento|lunghezza|contatore DTIM|durata DTIM|bitmap control|PVM]

PVM: partial virtual bitmap
Nota: tutti i campi sono di dimensione 1 byte eccetto il PVM che può andare da 1 a 256 bytes.

5.3 Power Save Mode con connessione di tipo Infrastructure

Per aiutare i dispositivi con il power save, gli AP sono stati creati con la capacità di bufferizzare al loro interno i frame destinati ai dispositivi in power save mode, per poi trasmetterglieli in un secondo momento, cioè quando l'AP viene a conoscenza che il dispositivo si è risvegliato. Come già detto, quando un device è in power save mode, disabilita il trasmettitore e ricevitore per preservare energia. È meno dispendioso per il dispositivo accendere il ricevitore per ascoltare i frame in arrivo piuttosto che attivare il trasmettitore per trasmettere i frame. Per questo motivo, è molto più efficiente dal punto di vista energetico che sia l'AP ad avvisare il device che ci sono frame per lui, piuttosto che sia il device a richiederlo sistematicamente tramite delle PS-Poll all'AP.
Durante l'associazione ad un AP, viene impostato un campo "Listen Interval" dal dispositivo. Il listen interval è misurato in “beacon interval units”, essenzialmente serve per informare l'AP sul numero di beacon che il device ha intenzione di ignorare prima di riattivare il ricevitore. Sono utilizzati 2 bytes per rappresentare il listen interval. L'associazione potrebbe o meno essere rifiutata, in base a com è impostato l'AP, generalmente la decisione si basa sulla quantità di spazio destinata ai frame da bufferizzare. Terminato il listen interval l'AP non garantisce più di bufferizzare i frame per il dispositivo e può quindi liberarsene.
In base allo standard IEEE-802.11 si è deciso di utilizzare una bitmap per indicare ad ogni device in sleep se l'AP ha dei frame bufferizzati per lui. Col fatto che ogni dispositivo può captare al più un beacon prima del listen interval, l'AP periodicamente invia questa bitmap all'interno dei suoi beacons come information element. Se nessun device ha frame

bufferizzati dall'AP, la PVM è settata come un singolo byte a 0 e l'offset della bitmap è impostato a sua volta a 0.

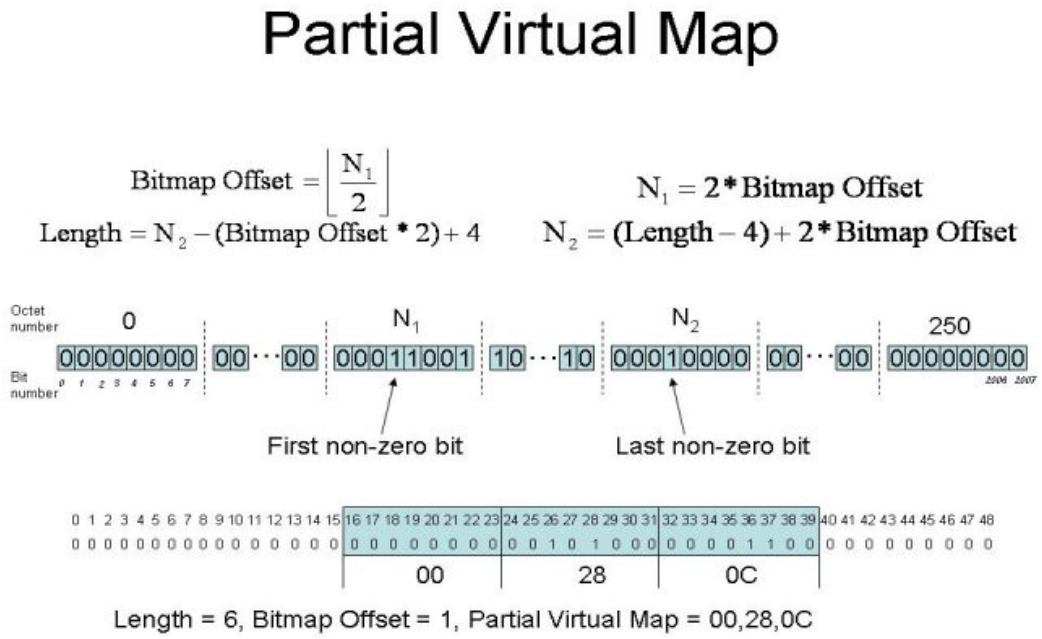

immagine 3: Partial Virtual Map

Dopo che un device riceve un TIM e nota che l'AP ha dei frame bufferizzati per lui, deve inviare un Power Save Poll (PS-Poll) control frame per recuperare ogni frame memorizzato dall'AP. Il dispositivo può tornare in sleep al termine dello scambio di frame tramite la PS-Poll o non appena riceve un TIM che non contiene più il suo AID.

Quando si ha a che fare con frame multicast o broadcast l'AP li bufferizza in maniera simile e devono impedire a tutti i device associati all'AP di dormire. Per comunicare ai dispositivi che ci sono dei frame broadcast o multicast memorizzati l'AID di 0 viene abilitato nel TIM, il quale corrisponde al primo bit del TIM bitmap control, e non al primo bit della TIM bitmap. Questo non viene settato nei TIMs di tutti i beacon ma solo in un tipo particolare di TIM.

Questo TIM particolare è detto DTIM (Delivery TIM), dopo aver inviato il DTIM l'access point invia tutti i frame broadcast o multicast che aveva memorizzato. Il DTIM viene inviato ogni DTIM period, che viene specificato nel TIM information element all'interno del

campo DTIM period. Questo campo è di un solo byte e rappresenta il numero di beacon interval che devono trascorrere prima di poter inviare il DTIM. Invece il campo DTIM count, sempre contenuto nel TIM information element, contiene il numero di beacon che verranno trasmessi prima dell'invio del prossimo DTIM[2].

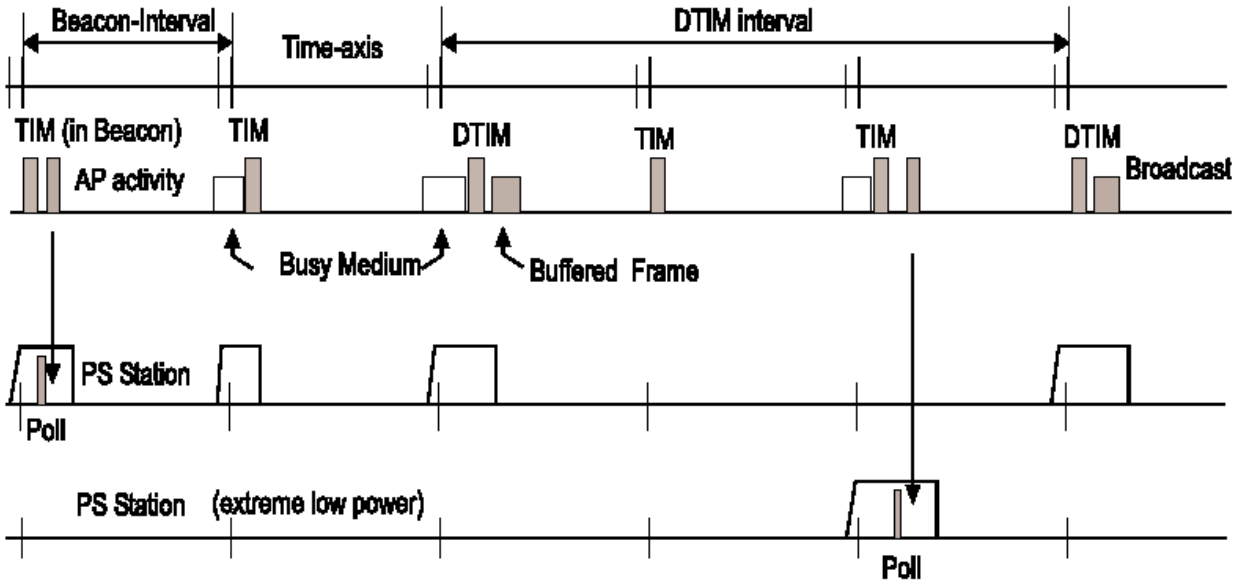

Immagine 4: Beacon Interval

5.4 Power Save Mode con connessione Ad-Hoc

Mostreremo ora il comportamento di un device in PSM durante una connessione Ad-hoc, procederemo alla descrizione per passi:

• All'inizio del beacon interval, se un nodo X ha un pacchetto da spedire ad un altro host Y, l'host X trasmette un messaggio di tipo Atim Request all'host Y. L'Atim Request viene trasmessa attraverso l'utilizzo del protocollo di accesso al canale 802.11 DCF. L'host X cerca di mandare la richiesta Atim a tutti gli host per i quali ha dei pacchetti in attesa di essere inviati. Queste richieste Atim possono essere spedite solo durante il periodo di Atim window.

• Se un host Y riceve una richiesta Atim da un qualche host X, allora Y risponde immediatamente inviando un Atim Ack, e poi rimane "sveglio" per il resto del beacon interval. La trasmissione di un Atim Ack è analoga alla trasmissione di un Ack al livello

MAC 802.11, in seguito alla ricezione di un data frame.

• Se un host non invia o riceve nessuna Atim Request durante l'Atim window, l'host va in sleep al termine dell'Atim window, e rimane addormentato per il resto del beacon interval.

• Qualsiasi host che abbia inviato o ricevuto un'Atim Request durante l'Atim window rimarrà attivo per il resto del beacon interval. Tale host potrà trasmettere e ricevere informazioni per tutta la durata del beacon interval utilizzando il protocollo di accesso medio 802.11.

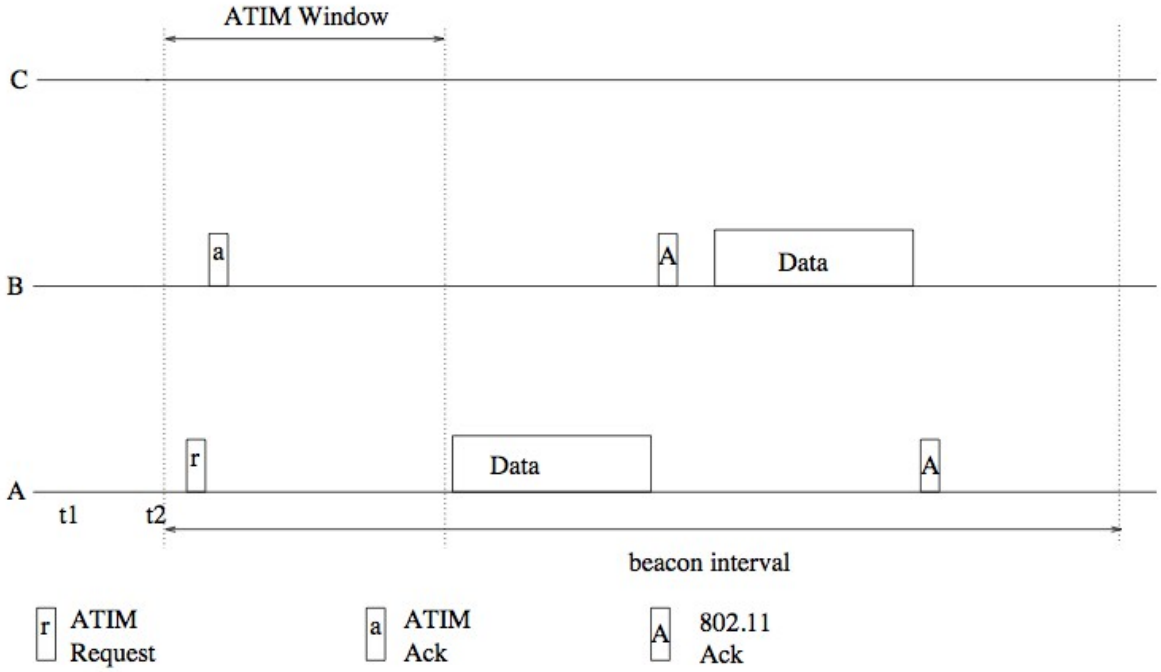

immagine 6: ATIM Window

• Se un'interfaccia wireless di un host sta dormendo durante un beacon interval, e l'host riceve un pacchetto da inoltrare ad un altro nodo, il pacchetto deve aspettare fino al prossimo beacon interval. Così, con il meccanismo di power save, si può incorrere in ritardi paragonabili ad un beacon interval prima che un pacchetto possa essere trasmesso da un nodo al suo vicino. Se un pacchetto deve essere trasmesso in più frammenti, il ritardo può aumentare in proporzione al numero di frammenti.

• Tutti i nodi rimangono svegli durante l'Atim window, cosa che può provocare un consumo energetico significativo. Per esempio, se la durata di un Atim window è 20 ms e con il

beacon interval diventa 100 ms, allora ogni nodo può dormire fino all'80% del tempo totale. Il consumo energetico durante l'Atim window può essere ridotto diminuendo la durata dell'Atim window stessa. Ad ogni modo, un'Atim window più corta permette meno tempo per lo scambio di Atim Request. Se un nodo non riesce a mandare un'Atim Request ad un altro host (a causa di una contesa durante l'Atim window), allora i pacchetti dell'host incorreranno in un ritardo maggiore.

Alternativamente, il consumo energetico può essere ridotto allungando la durata del beacon interval. In questo caso, aumenterà comunque il ritardo nella consegna dei pacchetti. Così, la scelta conveniente tra Atim window o beacon interval dipende dalle necessità del ritardo, oltre che dalla frequenza con la quale i nodi in power save mode necessitano di trasmettere pacchetti[3].

CAPITOLO 6

Test e valutazioni

Il nostro primo approccio di analisi del modello di sistema descritto nel capitolo 3, si è basato sullo studio del codice sorgente del kernel di Linux (3.6.8).
La scelta di questa versione del kernel è dovuta al fatto che, guardando le versioni precedenti, abbiamo notato come non fossero implementate funzioni di gestione del PSM (Power Save Mode).
Scaricati i sorgenti, supponendo che la directory di root sia /Linux-3.6.8, ci siamo concentrati sull'analisi dei moduli presenti all'interno della directory /Linux-3.6.8/net/mac80211.
In primis ci siamo occupati dello studio del protocollo IEEE802.11 in merito alle politiche di gestione del PSM.
Visto il nostro obiettivo finale, abbiamo ritenuto di primaria importanza l'analisi più approfondita dei moduli scan.c e offchannel.c, che si occupano della scansione e del PSM.
Il passo successivo è stato quello di cercare dei riscontri pratici riguardo il comportamento del kernel, che potessero confermare quanto descritto dai documenti esplicativi del PSM.
In pratica, per aiutarci a capire con esattezza l'avvicendamento delle chiamate delle funzioni, abbiamo inserito numerose printk() all'interno del codice sorgente del kernel.
Giunti a questo punto, per controllare il comportamento del kernel, abbiamo effettuato i seguenti passaggi:

- esecuzione della scansione tramite il comando *iwlist*
- controllo del log di sistema tramite il comando *dmesg*

Esecuzione della scansione tramite il comando ***iwlist****:*

Iwlist viene utilizzato per mostrare informazioni aggiuntive, riguardanti un'interfaccia wireless, che non sono mostrate dal comando *iwconfig*.

Nel nostro caso, il primo argomento passato al comando *iwlist* è il nome dell'interfaccia virtuale di rete, il secondo invece è scan (argomento che impone al comando *iwlist* di effettuare una scansione).

Il nostro interesse nel lanciare questo comando non era legato all'output del comando stesso (la lista degli AP disponibili), quanto più alle funzioni del kernel che tale comando andava a richiamare, seppur indirettamente.

Controllo del log di sistema tramite il comando ***dmesg****:*

La funzione di sistema printk(), permette di stampare ciò che le viene passato come argomento all'interno del log di sistema.

Il comando *dmesg*, quando invocato, restituisce in output video il log di sistema, permettendo quindi di valutare quali delle printk() inserite nel codice fossero state stampate.

Abbiamo inserito alcune printk() nei punti critici del codice, ad esempio:

• Come primo comando all'interno delle funzioni principali dei moduli ispezionati, per accertarci che queste funzioni venissero realmente richiamate.

• Ogni qualvolta ci fosse una decisione importante da prendere da parte del kernel (if/then/else, switch/case)

• Ogni volta che il valore di una variabile di nostro particolare interesse potesse essere modificato.

6.1 Test

Esiti del primo test:

Da un primo test semplice, che prevede una scansione, con il device collegato ad un AP, ma senza traffico di dati, prevedevamo, in base a tutti gli studi precedenti, che venissero stampate le printk() relative alle funzioni di scansione dei canali.

Con sorpresa, in seguito al comando *dmesg*, non abbiamo ottenuto alcuna stampa del log.
Dopo alcune ricerche abbiamo concluso che probabilmente il kernel affidasse il controllo della scansione ai driver della scheda di rete in uso, implementati con politiche differenti.
Per ovviare al problema abbiamo disabilitato l'interfaccia di rete, ed utilizzato un dispositivo esterno (chiavetta USB Digicom).
Ripetuti i test, con la certezza che la nuova interfaccia di rete caricasse solamente i moduli del kernel, senza introdurre alcun modulo esterno, abbiamo ottenuto i seguenti risultati.

Esiti del secondo test:

Le printk() ci hanno permesso di capire la sequenza temporale delle chiamate alle funzioni del codice del kernel.
In particolare le informazioni che abbiamo ottenuto sono le seguenti:

• Chiamata della funzione ieee80211_scan_work
• Chiamata della funzione ieee80211_start_sw_scan
• Stampa della variabile che indica il tipo di scansione (attiva/passiva): nel nostro caso passiva
• Chiamata della funzione ieee80211_offchannel_ps_enable
• Invio del nullfunc frame all'AP, con valore 1

Da questo momento in poi, per ogni canale da scandire esegue le stesse operazioni, elencate di seguito.

• Settaggio del canale da scandire
• Chiamata della funzione ieee80211_offchannel_stop_vifs, con conseguente uscita dal canale operativo.
• Recupero delle informazioni relative alla scansione di quel canale

All'inizio di ogni beacon interval, il kernel controlla la se il proprio AID sia presente o meno all'interno della bitmap inviatagli dall'AP. In caso affermativo, vengono invocate la ieee80211_scan_suspend e la ieee80211_offchannel_ps_disable.

Se il canale scandito non era l'ultimo, allora si riprende il ciclo di chiamate, a partire dal settaggio del nuovo canale da scandire, altrimenti, la scansione termina.

Giunti a questo step, abbiamo proceduto con altri test simili a quello appena descritto, cambiando però lo scenario.

Esiti del terzo test (flood UDP):

Per testare il comportamento della scansione con traffico di dati, abbiamo creato una semplice applicazione p2p che facesse scambio di pacchetti UDP della dimensione di 1KB l'uno.
Per evitare che venissero persi pacchetti in seguito al riempimento dei buffer di sistema, è stata inserita di tanto in tanto una sleep di un secondo.
Durante questo flood UDP sono state richieste diverse scansioni con il comando *iwlist.*
Al termine della scansione, non abbiamo riscontrato differenze rispetto ai test fatti senza traffico di dati attivi.
Diversi sono stati i risultati del quarto test.

Esiti del quarto test (traffico TCP):

Ripetendo quanto eseguito per il test numero 3, ma con un traffico di dati di tipo TCP, abbiamo ottenuto risultati sensibilmente differenti, che confermavano le ipotesi fatte in seguito a quanto studiato e spiegato nei capitoli precedenti.
In questo caso, la ieee80211_scan_suspend veniva richiamata, non ogni 5 canali scanditi, ma con periodi temporali molto minori, e più frequenti, quasi ogni singolo canale.
Abbiamo così potuto affermare che la scansione effettivamente viene sospesa ogni volta che l'AP invia un beacon per avvisare il nodo mobile della presenza di frame bufferizzati pendenti.

Premesso che quando parliamo di Power Saving, intendiamo un meccanismo di memorizzazione e successiva consegna dei frame da parte dell'AP ai dispositivi che operano in PSM, utilizzato quindi solo per il traffico in downlink.

Per quanto riguarda la trasmissione di pacchetti da parte di un device in PSM, questo controllo è affidato interamente al proprio kernel.

Gli ultimi test fatti, hanno evidenziato uno strano comportamento nella gestione dei pacchetti pendenti, in trasmissione. Infatti è evidente come i pacchetti di tipo UDP vengano ignorati, mentre per quelli di tipo TCP venga risvegliato il dispositivo.
Tali dati, non sono stati confermati dal supporto di alcuna documentazione, sarebbe quindi consigliabile approfondire la ricerca. È consigliabile, dal nostro punto di vista, iniziare le ricerche studiando il meccanismo di funzionamento del modulo tx.c ed in particolare della funzione ieee80211_tx_h_check_assoc.

Come ultima prova abbiamo fatto uno sniffing con Wireshark, per vedere che pacchetti venissero scambiati fra l'AP e il device.
Per poter intercettare nullfunc frame e beacon, è stato necessario impostare l'interfaccia di rete in monitor mode.

6.2 Monitor Mode:

La modalità monitor permette ad un un'interfaccia wireless di un device di monitorare tutto il traffico della rete wireless. A differenza della modalità promiscua, anch'essa usata per il packet sniffing, la modalità monitor permette di catturare i pacchetti senza essere necessariamente associati ad un AP o ad una rete Ad-Hoc.
Questa modalità si applica solamente alle reti wireless, quando invece la modalità promiscua può essere applicata sia alle reti wireless, che a quelle via cavo.
Vi sono altre 5 modalità possibili con cui una scheda wireless puo lavorare:

- Master: quando lavora come AP
- Managed: quando lavora come client, chiamato anche Station(STA)
- Ad-Hoc
- Mesh
- Repeater

Una volta impostata la scheda per lavorare in questa modalità, abbiamo ripetuto il test di scansione, e abbiamo intercettato lo scambio di pacchetti fra device e AP.

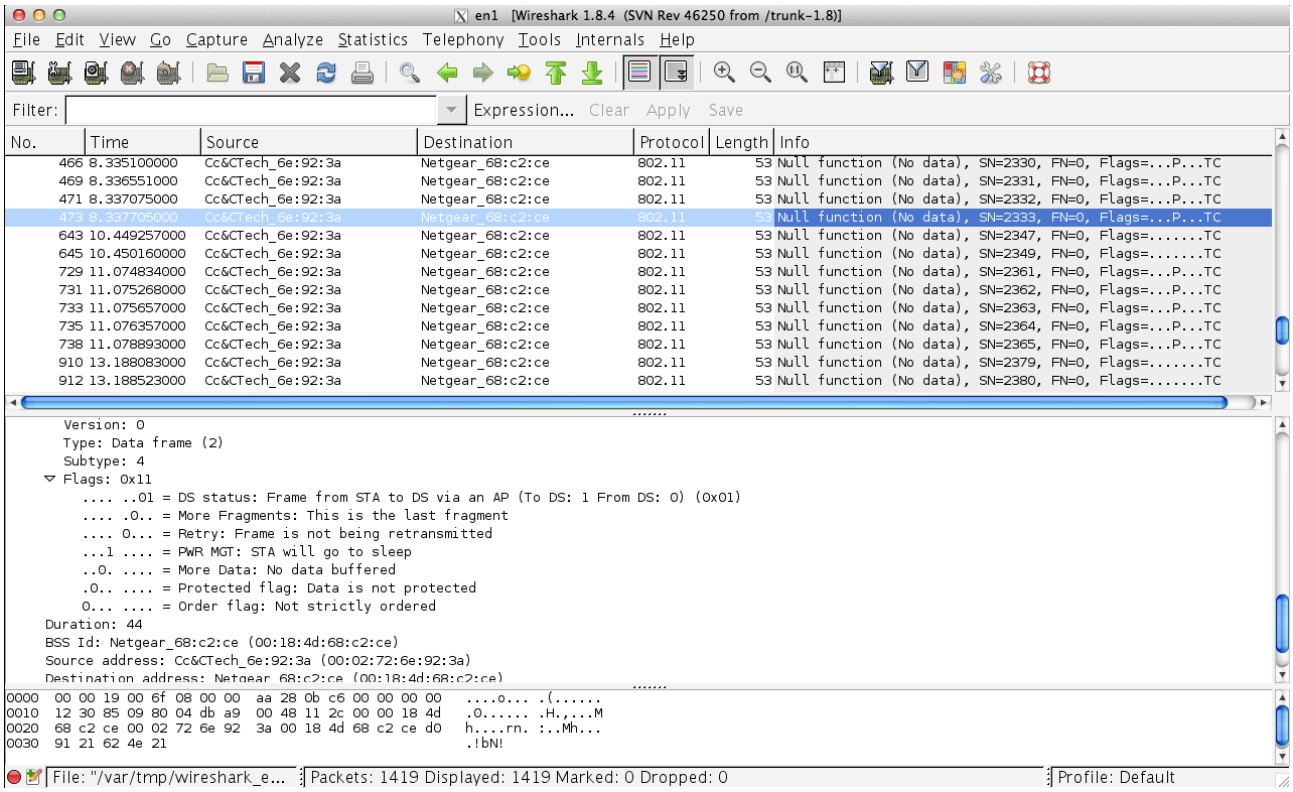

Immagine 7: PSM enabled

da questa immagine si può notare la comunicazione da parte del device all'AP delle proprie intenzioni di entrare in PSM attraverso il flag PWR MGT settato a 1.

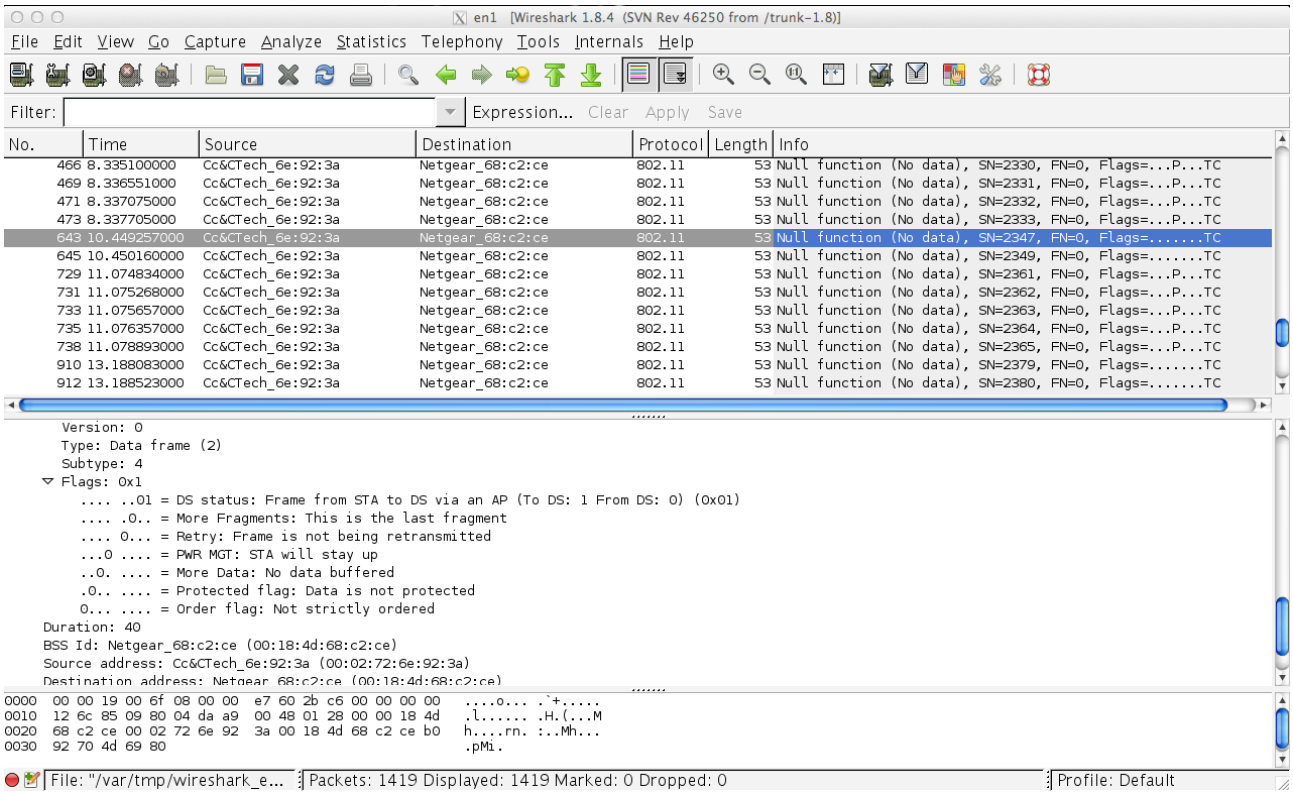

Immagine 8: PSM disabled

da quest'altra immagine, risulta chiara la comunicazione del device all'AP dell'uscita dal PSM con il flag PWR MGT settato a 0.

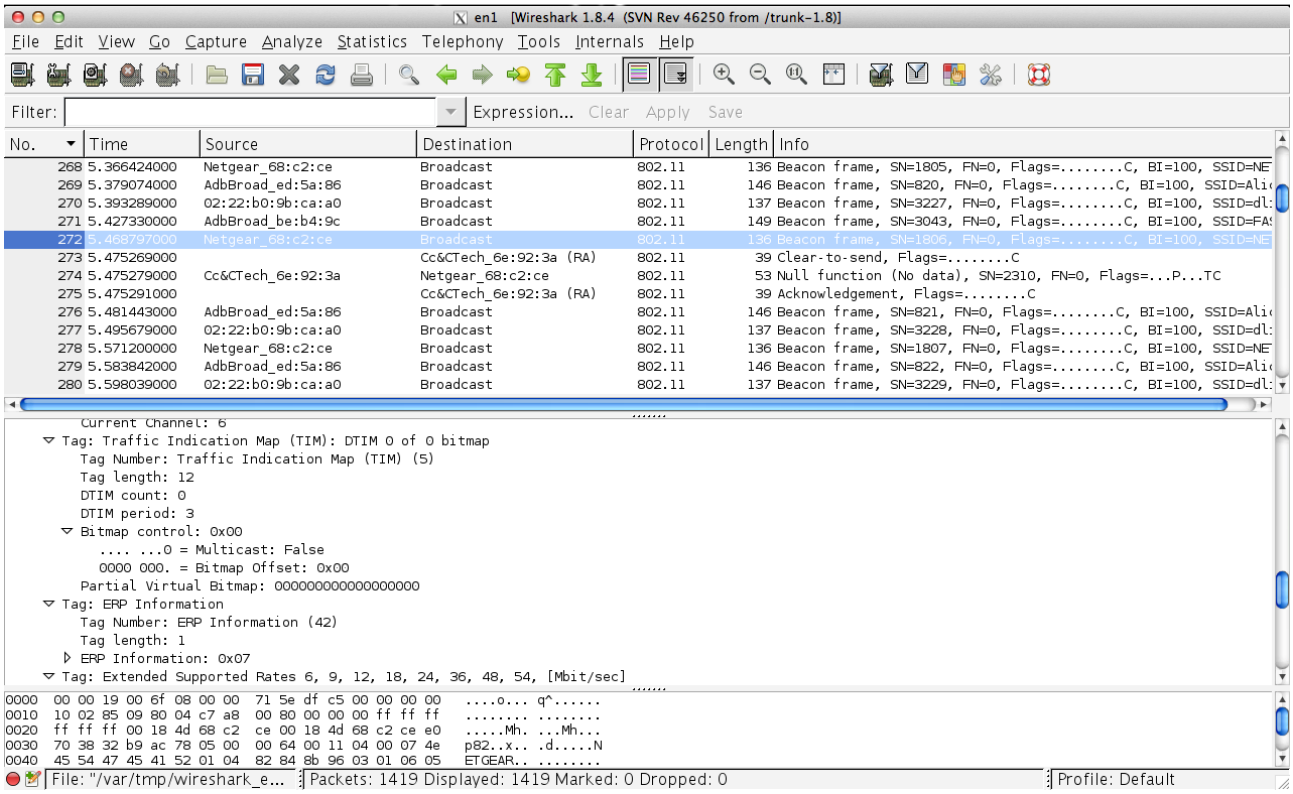

Immagine 9: bitmap

In questa immagine, in seguito ad un nullfunc frame con flag a 0, la bitmap viene settata a 0, così come il bitmap control.

CAPITOLO 7
Considerazioni e sviluppi

7.1 Considerazioni

In seguito agli studi ed ai test effettuati, abbiamo potuto concludere che il sistema che ci eravamo proposti di progettare (illustrato nel capitolo 3) è effettivamente realizzabile e il suo utilizzo è rivolto, non solo ad un mercato di nicchia (utenti esperti nel campo informatico), ma ad un vasto numero di utenti.
Infatti chiunque sia in possesso di un dispositivo mobile dotato di interfaccia wireless, potrebbe trarre enormi vantaggi dalle funzionalità del sistema.

7.2 Funzionalità del sistema

L'idea di base sarebbe quella di fornire all'utente un'applicazione che sia in grado di effettuare periodiche scansioni delle reti disponibili in una determinata area, salvando in un database tutti gli AP disponibili non ancora presenti all'interno del database stesso.
Le informazioni principali da memorizzare, per essere utilizzate in un secondo momento, potrebbero essere le seguenti:

- SSID della rete
- qualità del segnale
- posizione fisica dell'AP
- area di copertura dell'AP

Sono già disponibili applicazioni che permettono in qualche modo di compiere queste operazioni, è nostro interesse però effettuarle senza gravare troppo sul traffico di dati,

sfruttando le potenzialità offerte dal Power Save Mode (PSM).
Ciò avverrebbe customizzando la scansione dei canali secondo le linee proposte nei capitoli precedenti.

7.3 Linee guida per l'implementazione

A livello pratico, è necessario distinguere due livelli del codice da implementare per creare l'applicazione:

- user mode
- kernel mode

È nostro interesse lasciare la maggior parte del controllo delle operazioni da eseguire al livello utente, tuttavia sarà necessario implementare una serie di modifiche anche al codice del kernel.

-*User mode*:

L'utente dovrebbe poter essere in grado innanzitutto di scandire un singolo canale a sua scelta. Per fare ciò è necessario indicare al kernel su quale frequenza lavorare.

-*Kernel mode*:

In questo caso andrebbe modificato il codice del kernel aggiungendo una nuova costante per permettere il riconoscimento dell'operazione corrispondente (scansione su singolo canale), e predisporre lo stesso kernel per impostare la scansione del canale richiesto dall'utente.
Al termine di tale operazione sarebbe necessario terminare la scansione, impedendo la gestione automatica della ricerca sui canali successivi.

-*User mode*:

Sulla base di quanto appena descritto, dovrebbe essere relativamente semplice poter richiedere la scansione di tutti i canali, impostando il canale corretto a livello utente.
Questo si potrebbe implementare in un ciclo che richiama le funzionalità del kernel sopra introdotte, passando la costante da noi definita (scansione su singolo canale) e modificando di volta in volta il valore della variabile che rappresenta il canale da scandire.

-*Kernel mode*:

In questo caso non c'è nulla da implementare in più di quanto descritto precedentemente, si tratta soltanto di eseguire più volte l'operazione di scansione su un singolo canale.

Potremmo infine voler implementare una funzione che permetta di ottimizzare la scansione dei canali attualmente in uso nel kernel Linux.
L'idea sarebbe quella di lasciare al kernel il compito di scandire i canali senza indicare alcuna frequenza lato utente. Tuttavia, modificando il codice del kernel in maniera appropriata, si potrebbe permettere una scansione che controlli la presenza di frame pendenti con frequenza maggiore rispetto a quella di default.
Come spiegato nel capitolo 5, sappiamo che i device, quando entrano in PSM, possono scartare un numero arbitrario di beacon frame inviati dall'AP a cui erano connessi.
L'obiettivo sarebbe perciò quello di impedire che il device ignori i beacon dell'AP, o che ne ignori un numero arbitrario, in modo da poter controllare l'eventuale traffico di dati ogni N canali, definiti dall'utente.

-*User mode*:

Poiché viene lasciata al kernel la maggior parte del lavoro, in questo caso l'utente non deve far altro che passargli una nuova costante da noi creata, per comunicargli il nostro intento di effettuare una scansione completa dei canali nella versione ottimizzata ed eventualmente il numero di beacon frame da scartare.
-*Kernel mode*:

In questo caso il kernel andrebbe modificato affinché riconosca la nuova costante che richiama la funzione di scansione, evitando che venga ignorato alcun beacon.
Ricordiamo come durante una scansione ordinaria, quando il device richiede il PSM, si possa decidere di ignorare un numero arbitrario di beacon (inviati dall'AP su cui eravamo connessi) comunicandolo all'AP.
Pertanto, settando a 0 il numero di beacon da ignorare, avremmo una scansione, gestita totalmente dal kernel, ma che controlla la presenza di pacchetti pendenti ad ogni canale.

7.4 Tips

Consigliamo per l'implementazione delle funzioni, lo studio del codice sorgente di *iwlist* [4], in particolare è interessante la comprensione del passaggio di parametri da livello utente a livello kernel e viceversa.

7.5 Pro e Contro

In questo capitolo abbiamo evidenziato gli aspetti fondamentali dell'applicazione che ci eravamo proposti di progettare.
In seguito a quanto detto fino ad ora, presentiamo le nostre considerazioni conclusive sulla creazione di questa applicazione.
I vantaggi di tale applicazione, come già detto, riguardano per lo più device mobili.
Ci riferiamo dunque principalmente a smartphone, tablet o PC portatili.
Attualmente, il traffico di dati su questi dispositivi è consistente anche in background, viste le numerose applicazioni di instant messaging, social network, gli aggiornamenti stessi dei device o le applicazioni che utilizzano le funzionalità del gps, se disponibile.
Perciò appare evidente il beneficio che potrebbero trarre da una scansione che minimizzi il delay della trasmissione di pacchetti.

D'altro canto, l'implementazione di questa applicazione presenta anche alcuni aspetti negativi da prendere in considerazione.
Potrebbe non essere banale la modifica del kernel per permettere di inibire la scansione automatica.
Per fare ciò sarebbe necessario scrivere un nuovo modulo, o modificare moduli già esistenti; i tempi di modifica di un kernel sono sempre piuttosto lunghi, a causa della necessità di doverlo ricompilare ogni volta per testarne la correttezza.
Serve pertanto molta attenzione, prima di definire una nuova costante (come suggerito in questo capitolo) o implementare direttamente una nuova system call.
Vi è inoltre la possibilità (già affrontata nel capitolo 4) che la scheda di rete utilizzi i propri driver, senza far riferimento ai moduli del kernel; questo abbassa notevolmente la portabilità dell'applicazione.

Abbiamo anche studiato il comportamento della scansione su sistemi che utilizzano kernel Android, poiché tali dispositivi sono sempre più diffusi e riteniamo quindi di grande utilità la possibilità di sviluppare un'applicazione del genere anche su questi device.
Nonostante il kernel Android sia molto simile a quello Linux, non è stato semplice capire il funzionamento della scansione per i seguenti motivi:

- Livello di astrazione molto elevato da parte delle api che operano sulla scheda di rete
- Scarsa documentazione sui meccanismi di comunicazione dal livello applicazione al livello kernel
- Impossibilità di testare il kernel ricompilato su un emulatore in quanto, in tale emulatore, non sono implementate le funzionalità della scheda di rete; questo rende obbligatorio il test delle modifiche apportate al kernel direttamente su un dispositivo fisico.

Ciò non garantisce che il dispositivo non possa essere danneggiato da tali modifiche.

Definita certa la fattibilità del sistema su ambiente Linux e viste le somiglianze fra i due kernel, è certamente possibile intraprendere più concretamente studi di fattibilità anche in ambiente Android, per permettere la realizzazione di un progetto di portabilità del sistema anche su smartphone.

CAPITOLO 8
CONCLUSIONI

L'obiettivo di questo lavoro di tesi è stato quindi quello di analizzare i meccanismi di scansione e di Power Saving, partendo dallo studio dello standard IEEE802.11 (capitolo 2).
In particolare abbiamo focalizzato la nostra attenzione sul comportamento del kernel Linux (capitolo 4), proprio per quanto riguarda i moduli inerenti la scansione e il PSM (capitolo 5). Abbiamo inoltre riscontrato differente comportamento da parte di alcuni driver esterni al kernel, che l'interfaccia di rete può utilizzare (capitolo 4).
È pertanto necessario porre molta attenzione al tipo di driver utilizzato da una scheda di rete, prima di lavorare sulla scansione.
Abbiamo poi effettuato numerosi test sui moduli del kernel *scan.c* e *offchannel.c,* ricompilando il codice stesso del kernel, inserendo alcune printk() che permettessero di comprendere i passaggi chiave di questi meccanismi.
In seguito alla ricompilazione del kernel sono stati effettuati ulteriori test lanciando diverse scansioni e studiando l'output del log di sistema.
Inoltre, attraverso il software Wireshark, abbiamo analizzato il traffico di pacchetti presenti nella rete durante le nostre prove, per provare a capire meglio come avvenisse il dialogo tra device e AP. Questi test sono spiegati più nel dettaglio all'interno del capitolo 6.
Alla luce dei risultati ottenuti nei test, abbiamo potuto confermare la fattibilità di un sistema che implementi una scansione quasi completamente controllabile in user mode, lasciando quindi al kernel il minimo lavoro possibile. Ci siamo infine preoccupati di fornire consigli e linee guida per la realizzazione di questo sistema (capitolo 7).

Bibliografia

-Documentazione standard IEEE 802.11,
"802.11 Wireless Networks The Definitive Guide Oreilly Excellent" [0]

-Tesi di Baglivo, De Marco,
"*POLITICHE DI SELEZIONE DI ACCESS POINTS: ANALISI DEL TRAFFICO"* [1]

-Descrizione meccanismo di PSM,
http://linuxwireless.org [2]

-Documentazione Power Save Mode,
bookpowersave.pdf [3]

-Codice sorgente iwlist,
http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html [4]

-Costanti kernel linux,
http://lxr.free-electrons.com/source/include/uapi/linux/sockios.h#L35

-Codice sorgente API android,
http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/4.1.1_r1/android/provider/Settings.java?av=f

-Documentazione moduli android,
http://developer.android.com/reference/android/net/wifi/WifiManager.html

-Varie,
Wikipedia