

UNIVERSITA' DEGLI STUDI DI ROMA
TOR VERGATA



FACOLTÀ DI INGEGNERIA

TESI DI LAUREA SPECIALISTICA IN
INGEGNERIA DELLE TELECOMUNICAZIONI

**Soluzioni di autoconfigurabilità e qualità di servizio in reti
mesh 802.11: progetto e sviluppo**

Candidato

Francesco Saverio Proto

Relatore

Prof. Giuseppe Bianchi

Correlatore

Ing. Vito Ammirata

Anno Accademico 2005-2006

Sommario

SOMMARIO.....	2
INDICE DELLE FIGURE.....	4
INTRODUZIONE.....	5
CAPITOLO 1 - SCENARIO	9
IEEE 802.11	9
802.11b	11
802.11g	11
802.11a.....	12
802.11n	13
802.11e.....	13
Altre estensioni 802.11	14
Architettura di rete	15
Ad Hoc Network	17
Infrastructure Network.....	18
Concetto di Distribution System in 802.11	20
Mesh Wireless Distribution System	21
Hardware.....	23
Motherboards e firmwares.....	23
Chipsets e Drivers	27
CAPITOLO 2 - RASTA – RADIO AWARE SPANNING TREE AUTOCONFIGURATION	29
Autoconfigurazione.....	31
Radio Aware Spanning Tree	32
Statistiche Radio Passive	33
Organizzazione software delle statistiche	34
Aggiornamento dei path cost.....	35
Files di configurazione.....	38
Scripts	38
Program Flow	40
CAPITOLO 3 – YAQOSA – YET ANOTHER QUALITY OF SERVICE APPROACH.....	41
Lo standard IEEE 802.1Q	44
L'implementazione user space	45

Regole statiche e dinamiche.....	47
Files di configurazione.....	50
Program Flow	51
CAPITOLO 4 – TESTBED Sperimentale	53
OpenWRT con hardware Linksys.....	53
Prova del modulo distributore di YaQoS.....	55
Scripts	56
Testbed in architettura mipsel	59
RASTA, sviluppo e testbed	60
Linuxino con hardware Soekris.....	64
Testbed su Soekris net4801	66
Preparazione Testbed su Soekris net4826	67
BIBLIOGRAFIA	73

Indice delle figure

Figura 1 - Esempio di rete mesh.....	5
Figura 2 - Rappresentazione a strati 802.11.....	10
Figura 3 - Esempio di ESS	15
Figura 4 - Formato del frame 802.11	16
Figura 5 - Dettaglio dell'area "Frame Control" nel formato del frame 802.11	16
Figura 6 – Relazione tra campi address e bit "to DS" e "from DS"	17
Figura 7 - Esempio di rete ad-hoc	18
Figura 8 - Esempio di rete con infrastruttura	19
Figura 9 - Esempio di Wireless Distribution System (WDS).....	22
Figura 10 - Cavo JTAG su scheda madre di un Linksys WRT54G.....	25
Figura 11 - Schema del cavo JTAG.....	25
Figura 12 - Uso di Spanning Tree in una Wireless Mesh.....	30
Figura 13 - Snapshot di Ethereal: il Prism Monitoring Header	33
Figura 14 - Diagramma di flusso del codice di RASTA	40
Figura 15 - Esempio di tagging VLAN nel DS	42
Figura 16 - Architettura di YaQoS.....	45
Figura 17 - Architettura Distributore/Decisore di YaQoS	47
Figura 18 - Diagramma di flusso del modulo Distributore.....	51
Figura 19 - Diagramma di flusso del modulo Decisore.....	52
Figura 20 - Testbed sperimentale con Linksys WRT54G	59
Figura 21 - Traccia di ethereal sul testbed sperimentale con Linksys WRT54G	60
Figura 22 - Primo Testbed su architettura x86	66
Figura 23 - Soekris net4826	67

Introduzione

La comunicazione wireless sta acquistando interesse crescente, grazie alla massiccia diffusione di hardware 802.11, e di reti mesh senza fili in ambiente outdoor.

Per rete mesh s'intende un ESS 802.11 dove i collegamenti tra gli AP, ovvero il Distribution System (DS), sono realizzati anch'essi con tecnologia wireless in una topologia radio multi-hop.

In altre parole le reti mesh sono sistemi auto-configuranti, dove ogni nodo della rete può connettersi ad altri nodi, aumentando così la copertura e la banda disponibile.

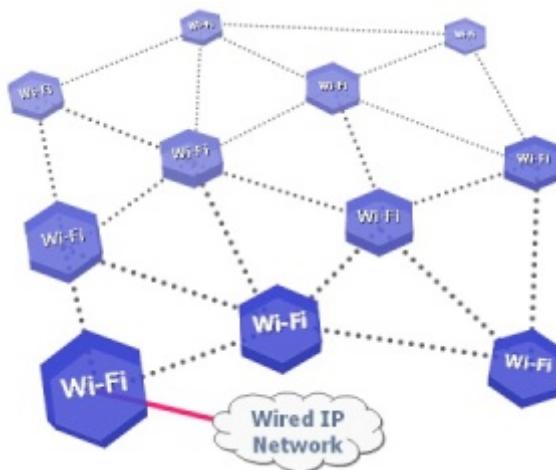


Figura 1 - Esempio di rete mesh

Sebbene le reti mesh siano già utilizzate in alcuni ambiti, non esiste ancora uno standard definito per il Wi-Fi in modalità mesh. Il Wi-Fi mesh è stato pensato partendo dagli standard 802.11a/b/g/i esistenti, aggiungendo funzionalità che permettono ad ogni singolo nodo della rete di cercare altri nodi, di autenticarsi e di stabilire una connessione. Tutto questo sarà standardizzato nell'802.11s, nel quale Intel ha proposto che sia implementato il QoS (Quality of Service), introducendo inoltre l'idea dei Mesh Portals, apparati in grado di connettere le reti mesh con reti

INTRODUZIONE

non mesh (ad esempio Ultra Wide Band). Lo standard definitivo arriverà a cavallo tra il 2006 ed il 2007 e le prime applicazioni commerciali arriveranno entro il 2008.

In questo lavoro di tesi sono stati studiati i problemi delle implementazioni open source per reti WDS mesh 802.11, ne sono stati analizzati i difetti, e sono state progettate ed implementate opportune soluzioni ora disponibili in pacchetti software. In particolare sono stati affrontati i problemi dell'autoconfigurabilità, dell'adozione di metriche adeguate a caratterizzare il canale radio e della garanzia di una QoS adeguata per servizi interattivi e multimediali che necessitano di requisiti in termini di prestazioni, a volte critici, in ambiente radio. In altre parole l'idea è stata quella di migliorare le prestazioni delle implementazioni pre-standard già presenti nello scenario del software open source.

L'approccio adottato è stato quello di lavorare completamente a livello due della pila protocollare ISO/OSI, garantendo così un'implementazione conforme allo standard, in modo da non causare problematiche di tipo cross-layer ai protocolli degli strati superiori.

Particolare attenzione è stata posta sul problema dell'autoconfigurabilità della rete, caratteristica che diventa di fondamentale importanza all'aumentare del numero dei nodi e che spesso ne limita la scalabilità in termini di dimensioni.

Inoltre, in molte reti wireless mesh esistenti non c'è una gestione centralizzata della configurazione dei nodi della rete, ma ogni nodo è gestito da un diverso amministratore. Il concetto di configurazione dell'intera rete da parte di un'unica entità è quindi stato abbandonato, a favore di un approccio di configurazione del singolo nodo al fine di cooperare con tutti gli altri.

Questo è il caso delle reti mesh delle wireless community, fenomeno che sta aumentando sempre di più e che sta portando alla costruzione di vere e proprie infrastrutture di telecomunicazioni.

Spesso accade che, in fase di implementazione, protocolli sviluppati in ambiente wired vengono trasportati in scenari wireless senza chiedersi se le ipotesi di base, sulle quali sono fondate gli algoritmi di tali protocolli, continuano ad essere valide. In altre parole potremmo dire che questi protocolli non sono "radio aware", ovvero non tengono conto del fatto che il canale utilizzato è di tipo radio. Il risultato è un comportamento subottimo, o altre volte disastroso, di questi protocolli. In particolare quando questo accade con protocolli di routing o di spanning tree le metriche

INTRODUZIONE

adottate sono spesso inadeguate a caratterizzare i collegamenti di una rete senza fili, che ha caratteristiche di latenza e Bit Error Rate molto diverse da una rete wired.

Per questi motivi sono state studiate modifiche software per i protocolli di Spanning Tree usati nelle tradizionali ethernet 802.11, in modo da adeguarli con metriche che meglio descrivono la qualità del canale radio.

I requisiti minimi di banda latenza e jitter, necessari al corretto funzionamento di applicazioni multimediali e/o interattive, come ad esempio il VoIP, sono per lo più ampiamente soddisfatti in reti cablate, mentre sono spesso ottenuti con difficoltà o con poco margine di tolleranza nelle reti wireless.

Per questo motivo, in una rete wireless in cui differenti tipi di servizio devono coesistere, è essenziale disporre di meccanismi di QoS per differenziare la priorità assegnata ai diversi flussi informativi.

Inoltre una rete mesh potrebbe disporre di più gateway verso l'esterno (ad esempio verso internet). E' quindi opportuno prevedere meccanismi per bilanciare il carico tra i diversi gateway in modo da massimizzare l'efficienza.

Il presente lavoro si pone all'interno di un più ampio progetto condotto dall'Università di Tor Vergata in collaborazione con altre università italiane per la realizzazione di una mesh wireless LAN 802.11, denominato PRIN TWELVE.

All'interno del progetto TWELVE l'obiettivo, al cui raggiungimento il presente lavoro si pone come una delle basi, è quello di sviluppare una rete mesh all'interno del campus universitario, in cui gli Access Points siano caratterizzati dalla capacità di supportare differenziazione di servizio.

In pratica quella che si realizzerà è una rete in cui gli Access Points sono in grado di instradare il traffico dei client verso uno o più gateway connessi ad Internet attraverso link outdoor differenti, in base al livello di servizio che si vuole offrire.

Nel primo capitolo descriveremo lo scenario nel quale è stato svolto questo lavoro. Dapprima daremo una descrizione dello standard 802.11 ed in modo particolare verranno approfonditi i suoi aspetti più importanti al fine della comprensione del lavoro svolto.

Verrà poi descritto lo stato dell'arte per ciò che riguarda hardware e software presente sul mercato per sistemi operanti a 2,4Ghz in 802.11

Nel secondo e nel terzo capitolo descriviamo i due pacchetti software sviluppati: RASTA e YaQoS. Per entrambi saranno descritte le *features*, gli algoritmi, l'implementazione e il modo d'uso. Nella spiegazione del metodo implementativo

INTRODUZIONE

saranno anche trattate delle parti di codice fondamentali. Per il modo d'uso è presente una descrizione dettagliata riguardante cosa e come viene specificato nei files di configurazione.

Infine nel quarto capitolo viene documentato il lavoro sperimentale di laboratorio, dove vengono illustrate le varie scelte implementative e le problematiche riscontrate durante lo sviluppo del codice.

Capitolo 1 - Scenario

In questo capitolo si vuole dare una panoramica sull'utilizzo del protocollo IEEE 802.11 nel caso specifico del Wireless Distribution System (WDS).

Lo standard IEEE 802.11 specifica il livello fisico ed il livello MAC (Media Access Control) corrispondenti ai primi due livelli del modello di riferimento ISO/OSI. Una rete 802.11 è un'estensione di una rete locale cablata, che permette la comunicazione sul tratto wireless. Questo tipo di rete mantiene una completa compatibilità con le comuni reti Ethernet (802.3), che vengono così arricchite delle proprietà di mobilità e portabilità.

Il tratto wireless per sua natura inaffidabile impone delle soluzioni diverse rispetto alle soluzioni adottate per le reti cablate, inoltre l'elevato Bit Error Rate del canale radio non permette di raggiungere prestazioni paragonabili a quelle raggiungibili su un filo di rame.

IEEE 802.11

Lo standard IEEE 802.11 definisce un livello fisico (PHY) e un protocollo di accesso al mezzo (MAC) per reti locali wireless, queste sono utilizzate come estensione o come alternativa ad una rete cablata. Utilizzando una tecnologia a radio frequenza le WLAN trasmettono in aria, minimizzando l'utilizzo del cablaggio.

La prima versione dello standard 802.11 venne presentata nel 1997 e viene chiamata "802.1y", specificava velocità di trasferimento comprese tra 1 e 2 Mbit/s e utilizzava i raggi infrarossi o le onde radio nella frequenza di 2.4 Ghz per la trasmissione del segnale. La trasmissione infrarosso venne eliminata dalle versioni successive, dato lo scarso successo. La maggior parte dei costruttori, infatti, non aveva optato per lo standard IrDA, preferendo la trasmissione radio. Il supporto di questo standard, per quanto riguarda la trasmissione via infrarossi, è incluso delle evoluzioni dello standard 802.11 per ragioni di compatibilità. Poco dopo questo standard vennero prodotte da due produttori indipendenti, delle evoluzioni dello

standard 802.1y che, una volta riunite e migliorate, portarono alla definizione dello standard 802.11b.

Una rete locale wireless del tipo 802.11 appare come una comune 802 LAN al livello Logical Link Control (LLC) nella corrispondente pila ISO-OSI.

802.2			Data Link Layer
802.11 MAC			
FH	DS	IR	Livello Fisico

Figura 2 - Rappresentazione a strati 802.11

Alle funzionalità svolte da un tipico livello MAC si aggiungono la frammentazione, la ritrasmissione e i riscontri, in questo modo si cerca di combattere i difetti di un canale radio con elevato Bit Error Rate. La frammentazione dei pacchetti ha una duplice motivazione: se il pacchetto è più corto, è più probabile che venga ricevuto senza errori, per cui fare frammentazione vuol dire cercare di limitare gli effetti dovuti al canale radio, inoltre la frammentazione è realizzata per mantenere una compatibilità con Ethernet (la dimensione massima del pacchetto è 1518 bytes).

Come già spiegato in precedenza lo standard è suddiviso in varie versioni, quella presa in considerazione in questo lavoro di tesi è la versione IEEE 802.11b che ha le seguenti caratteristiche:

- Una rete 802.11b lavora alla frequenza di 2,4 GHz riservata ad applicazioni industriali, scientifiche e mediche (ISM) e dispone di 83 MHz di banda.
- Area di copertura di 150m indoors, fino a 600m outdoors (con antenne omnidirezionali).
- Lo standard specifica tre interfacce per il livello fisico (Direct Sequence Spread Spectrum, Frequency Hopping Spread Spectrum, Infrared Pulse Position Modulation)
- I rate di trasmissione vanno da 1-2 Mbps fino a 11 Mbps a seconda del tipo di modulazione del segnale (BPSK,QPSK,CCK) .

- La potenza massima del segnale è regolata da norme internazionali ed è fissata al valore di 1 Watt EIRP (Equivalent Isotropically Radiated Power) perciò in base ai vari schemi di modulazione, per mantenere la stessa qualità del segnale, varia l'area di copertura essendo la potenza vincolata.

802.11b

802.11b ha la capacità di trasmettere al massimo 11Mbit/s e utilizza il Carrier Sense Multiple Access con Collision Avoidance (CSMA/CA) come metodo di trasmissione delle informazioni. Una buona parte della banda disponibile viene utilizzata dal CSMA/CA. In pratica il massimo trasferimento ottenibile è di 5.9 Mbit/s in TCP e di 7.1 Mbit/s in UDP. Metallo, acqua e in generale ostacoli solidi riducono drasticamente la portata del segnale. Il protocollo utilizza le frequenze nell'intorno dei 2.4Ghz.

Utilizzando antenne esterne dotate di alto guadagno si è in grado di stabilire delle connessioni punto a punto del raggio di molti chilometri. Utilizzando ricevitori con guadagno di 80 decibel si può arrivare a 8 chilometri o, se le condizioni del tempo sono favorevoli, anche a distanze maggiori ma sono situazioni temporanee che non consentono una copertura affidabile. Quando il segnale è troppo disturbato o debole lo standard prevede di ridurre la velocità massima a 5.5, 2 o 1 Mbit/s per consentire al segnale di essere decodificato correttamente.

Il primo produttore commerciale a utilizzare il protocollo 802.11b è stata Apple Computer con il marchio AirPort. Il primo produttore per IBM compatibili è stato Linksys, l'attuale leader di questi prodotti.

802.11g

Nel giugno del 2003 questo standard venne ratificato. Utilizza le stesse frequenze del b cioè la banda di 2.4 Ghz e fornisce una banda teorica di 54 Mbit/s che nella realtà si traduce in una banda netta di 24.7 Mbit/s, simile a quella dello standard 802.11a. È totalmente compatibile con lo standard b ma quando si trova a operare con periferiche b deve ovviamente ridurre la sua velocità a quella dello standard b.

Prima della ratifica ufficiale dello standard 802.11g avvenuta nell'estate del 2003 vi erano dei produttori indipendenti che fornivano delle apparecchiature basate sulle specifiche non definitive dello standard. I principali produttori comunque preferirono aspettare le specifiche ufficiali e quando furono disponibili molti dei loro prodotti vennero resi compatibili con il nuovo standard.

Alcuni produttori introdussero delle ulteriori varianti chiamate g+ o Super G nei loro prodotti. Queste varianti utilizzavano l'accoppiata di due canali per raddoppiare la banda disponibile anche se questo indiceva interferenze con le altre reti e non era sopportato da tutte le schede.

Il primo grande produttore a rilasciare schede con le specifiche ufficiali 802.11g fu nuovamente Apple che presentò i suoi prodotti "AirPort Extreme". Cisco decise di entrare nel settore acquistando Linksys e fornì i suoi prodotti con il nome di "Aironet".

802.11a

Nel 2001 venne ratificato il protocollo 802.11a approvato nel 1999. Questo standard utilizza lo spazio di frequenze nell'intorno dei 5 Ghz e opera con una velocità massima di 54 Mbit/s sebbene nella realtà la velocità reale disponibile all'utente a strato applicativo sia di circa 20 Mbit/s. La velocità massima può essere ridotta a 48, 36,34,18,9 o 6 se le interferenze elettromagnetiche lo impongono. Lo standard definisce 12 canali non sovrapposti, 8 dedicati alle comunicazioni interne e 4 per le comunicazioni punto a punto. Quasi ogni stato ha emanato una direttiva diversa a per regolare le frequenze, ma dopo la conferenza mondiale per la radiocomunicazione del 2003 l'autorità federale americana ha deciso di rendere libere le frequenze utilizzate dallo standard 802.11a.

Questo standard non ha riscosso i favori del pubblico dato che l'802.11b si era già molto diffuso e in molti paesi l'uso delle frequenze a 5 Ghz è tuttora riservato. Esistono schede dual standard o tri standard in grado di accettare oltre allo standard a anche il b e per le schede tri standard anche il g. Ovviamente esistono anche degli Access point multi standard.

802.11n

Nel gennaio 2004 IEEE ha annunciato di aver avviato lo studio di un nuovo standard per realizzare reti wireless di dimensioni metropolitane. La velocità reale di questo standard dovrebbe essere di 100 Mbit/s (quella fisica dovrebbe essere prossima a 250 Mbit/s), quindi dovrebbe essere 5 volte più rapido del 802.11g e 40 volte più rapido dell'802.11b. Si presume che la standardizzazione terminerà nel 2006.

802.11n introdurrà anche la possibilità di utilizzare la tecnologia MIMO (multiple-input multiple-output). Questo consentirà di utilizzare più antenne per trasmettere e più antenne per ricevere, incrementando la banda disponibile utilizzando una multiplazione di tipo spaziale attraverso una codifica tipo quella di Alamouti.

802.11e

IEEE 802.11e è un draft, ancora in attesa di approvazione definitiva da parte dell'IEEE 802.11 Working Group, che prevede la suddivisione dei servizi all'interno delle stazioni trasmittenti. Le code dei pacchetti da trasmettere vengono differenziate a seconda dell'appartenenza a definite Access Category. Queste sono caratterizzate da tempi di accesso al servizio differenti, in modo da garantire differenti livelli di priorità. Normalmente i servizi che richiedono minori ritardi di trasmissione, tipicamente servizi Real Time (VoIP per videoconferenze, telecontrollo, ecc.), sono associati a livelli di priorità più elevati. Al contrario il traffico Best Effort (web browsing e FTP) viene associato ad una priorità inferiore.

La differenziazione delle classi di traffico si ottiene differenziando l'ampiezza delle Contention Windows e la durata degli Inter-Frame Space: una maggiore durata di questi tempi è legata ad una minore priorità. La possibilità di aggiornare questi parametri durante la trasmissione permette di adattare la rete allo stato di congestione, garantendo una maggiore Qualità del Servizio (QoS) e la fairness della rete. Alla base di questa modalità operativa c'è la necessità da parte delle stazioni di informare l'Access Point del tipo di traffico che si propongono di generare.

IEEE 802.11e introduce anche la definizione di un periodo Contention Free Burst (CFB period), durante il quale la stazione che accede al canale può trasmettere pacchetti in un Burst (ovvero una sequenza continua intervallata solo da brevi SIFS - Short InterFrame Space) senza dover ripetere l'accesso al canale tramite CSMA/CA. Si può scegliere se avere un acknowledge separato per ogni pacchetto oppure un unico acknowledge globale per tutto il burst trasmesso. Queste tecniche sono volte alla riduzione dei tempi morti del protocollo 802.11.

Altre estensioni 802.11

Questi sono gli standard approvati o in fase di studio dall'IEEE:

- IEEE 802.11 - Lo standard originale 2 Mbit/s, 2.4 GHz
- IEEE 802.11a - 54 Mbit/s, 5 GHz standard (1999, approvato nel 2001)
- IEEE 802.11b - Miglioramento del 802.11 col supporto di 5.5 e 11 Mbit/s (1999)
- IEEE 802.11d - Libero
- IEEE 802.11e - Miglioramento: Gestione della qualità del servizio.
- IEEE 802.11F - Inter-Access Point Protocol (IAPP)
- IEEE 802.11g - 54 Mbit/s, 2.4 GHz standard (compatibile con il 802.11b) (2003)
- IEEE 802.11h - 5 GHz spectrum, Dynamic Channel/Frequency Selection (DCS/DFS) e Transmit Power Control (TPC) per compatibilità con l'Europa
- IEEE 802.11i (ratified 24 giugno 2004) - Miglioramento della sicurezza
- IEEE 802.11j - Estensione per il Giappone
- IEEE 802.11k - Misurazione delle sorgenti radio
- IEEE 802.11n - Aumento della banda disponibile
- IEEE 802.11p - WAVE - Wireless Ability in Vehicular Environments (gestione per autoveicoli, ambulanze, ecc...)
- IEEE 802.11r - Roaming rapido
- IEEE 802.11s - Gestione della topologia della rete
- IEEE 802.11T - Gestione e Test
- IEEE 802.11u - Connessione con reti non 802 , tipo le reti cellulari.
- IEEE 802.11v - Gestione delle reti wireless

Architettura di rete

Le reti locali che utilizzano lo standard IEEE 802.11 sono basate su un architettura di tipo cellulare, il sistema è suddiviso in celle. Ogni cella, chiamata Basic Service Set o più brevemente BSS, è controllata da una stazione base chiamata Access Point (Punto d'Accesso o AP). Sebbene una wireless LAN possa essere formata da un singola cella e da un solo Punto d'Accesso, molte installazioni sono formate da molte celle, dove gli Access Point sono collegati da una sorta di backbone chiamato Distribution System (Sistema di Distribuzione o DS). Questo backbone è tipicamente Ethernet e in alcuni casi, come esposto in questa tesi, è anche wireless.

L'intera rete wireless, incluse le celle, i rispettivi Access Point e il Distribution System, è vista come una singola rete 802 dai livelli superiori del modello OSI ed è chiamato Extended Service Set (ESS).

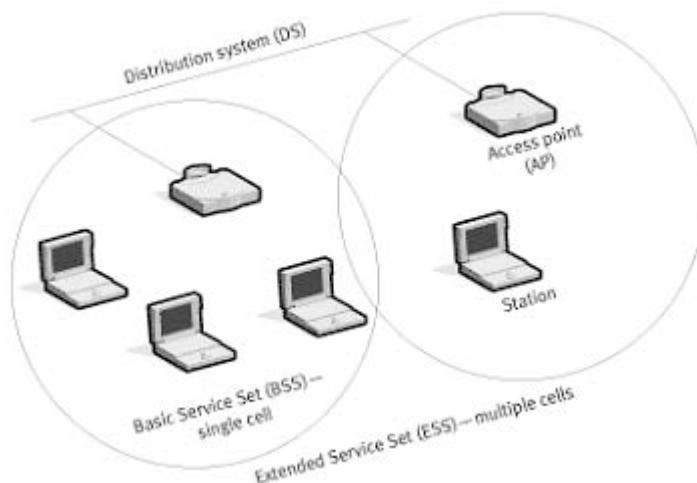


Figura 3 - Esempio di ESS

Lo standard prevede anche un'entità tra la rete 802.11 e una qualsiasi rete 802, questa viene detta Portale ed è a tutti gli effetti un bridge. Comunemente il portale e l'Access Point risiedono su una stessa macchina anche se lo standard non lo

specifica. Nello standard sono definite due architetture di rete la Ad Hoc Network e la Infrastructure Network.

La differenza fondamentale tra le due architetture è lo schema di indirizzamento dei frames. In una rete ad-hoc la STA può comunicare solo con le stazioni che sono direttamente raggiungibili via radio, e nelle comunicazioni il mittente ed il destinatario sono sempre anche il trasmettitore ed il ricevitore radio. In una rete con infrastruttura, all'interno di un BSS, ogni STA deve parlare solo ed esclusivamente con l'AP a cui è associata, ovvero tutte le comunicazioni tra le STA o verso l'esterno passano tutti esclusivamente per l'AP.

Per distinguere tra le due configurazioni all'interno del frame sono stati riservati 2 bit, “to DS” e “from DS” che si trovano all'interno del campo “Frame Control”.

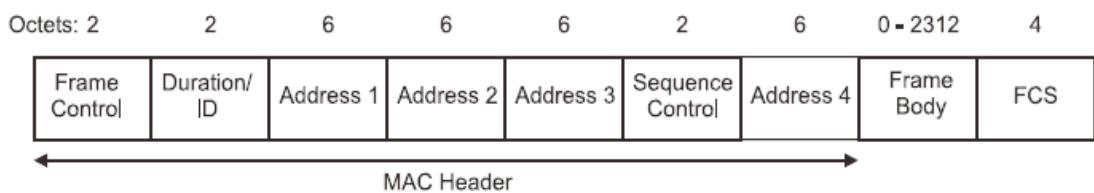


Figura 4 - Formato del frame 802.11

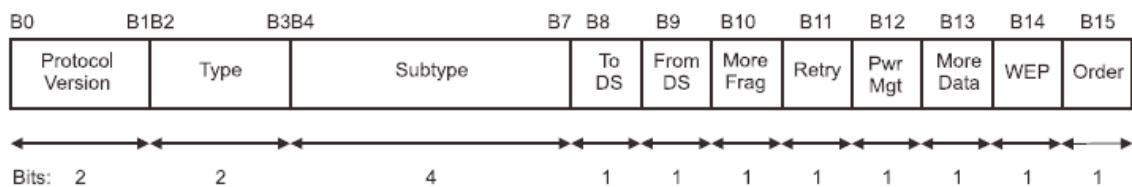


Figura 5 - Dettaglio dell'area "Frame Control" nel formato del frame 802.11

Tramite questi 2 bit è possibile mappare 4 valori che descrivono le seguenti situazioni.

- 1) Il frame fa parte di un IBSS (rete ad-hoc senza infrastruttura)
- 2) Il frame viaggia dall'AP verso la STA
- 3) Il frame viaggia dalla STA verso l'AP
- 4) Il frame sta attraversando un tratto di DS

A seconda dei valori prendono significati diversi i 4 campi address all'interno del frame 802.11. E' da notare come il quarto indirizzo sia utilizzato solo quando il frame 802.11 attraversa il DS, e contiene invece dati quando non viene utilizzato nella BSS o in un IBSS.

Function			Receiver	Transmitter		
	To DS	From DS	Address 1	Address 2	Address 3	Address 4
IBSS	0	0	RA = DA	SA	BSSID	N/A
From AP	0	1	RA = DA	BSSID	SA	N/A
To AP	1	0	RA = BSSID	SA	DA	N/A
Wireless DS	1	1	RA	TA	DA	SA

Figura 6 – Relazione tra campi address e bit "to DS" e "from DS"

Ad Hoc Network

Una Ad Hoc Network è un'architettura usata per supportare mutue comunicazioni tra dispositivi wireless. Questo tipo di architettura tipicamente non permette l'accesso ad altre reti cablate e non necessita di un Access Point. In una rete ad-hoc, diversi dispositivi come portatili e palmari possono essere avvicinati per formare una rete “al volo”.

Non c'è nessuna struttura che caratterizza la rete appena formata, né punti fissi, e tipicamente ogni nodo può comunicare con ogni altro nodo.

I frames viaggiano tutti con i bit “to DS” e “from DS” posti a 0. Il terzo indirizzo del frame 802.11 contiene il BSSID, un MAC address che serve come riferimento per la rete appena costituita.

Per mantenere l'ordine nella rete appena formata possono essere utilizzati algoritmi come lo Spokesman Election Algorithm (SEA) che serve ad eleggere una macchina come stazione base (master) a cui tutte le altre macchine (slaves) faranno riferimento, oppure utilizzare un algoritmo che fa uso di messaggi broadcast e di flooding per stabilire quanti nodi sono presenti nella rete.

Un esempio di scenario in cui può essere utilizzata una rete ad-hoc potrebbe essere un incontro di lavoro, dove gli impiegati utilizzano computer portatili per condividere dati, grafici o informazioni finanziarie.

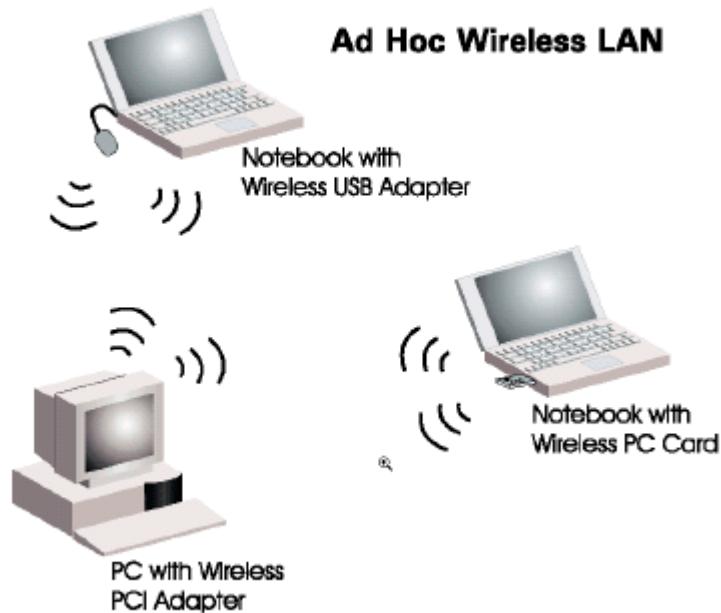


Figura 7 - Esempio di rete ad-hoc

Infrastructure Network

Il secondo tipo di architettura di rete è quella che fa uso di una infrastruttura. Questa consente la comunicazione tra dispositivi wireless e le risorse della rete wired. Il passaggio di dati tra le due tipologie di rete è regolato dall'Access Point. L'architettura fa uso di AP mediante i quali i terminali mobili (Wireless Terminal o WT) possono comunicare. Per estendere la capacità della WLAN gli Access Point svolgono la funzionalità di bridge consentendo ai nodi della cella di accedere a punti remoti nella rete, con un architettura che è molto simile a quella cellulare.

CAPITOLO 1 - SCENARIO

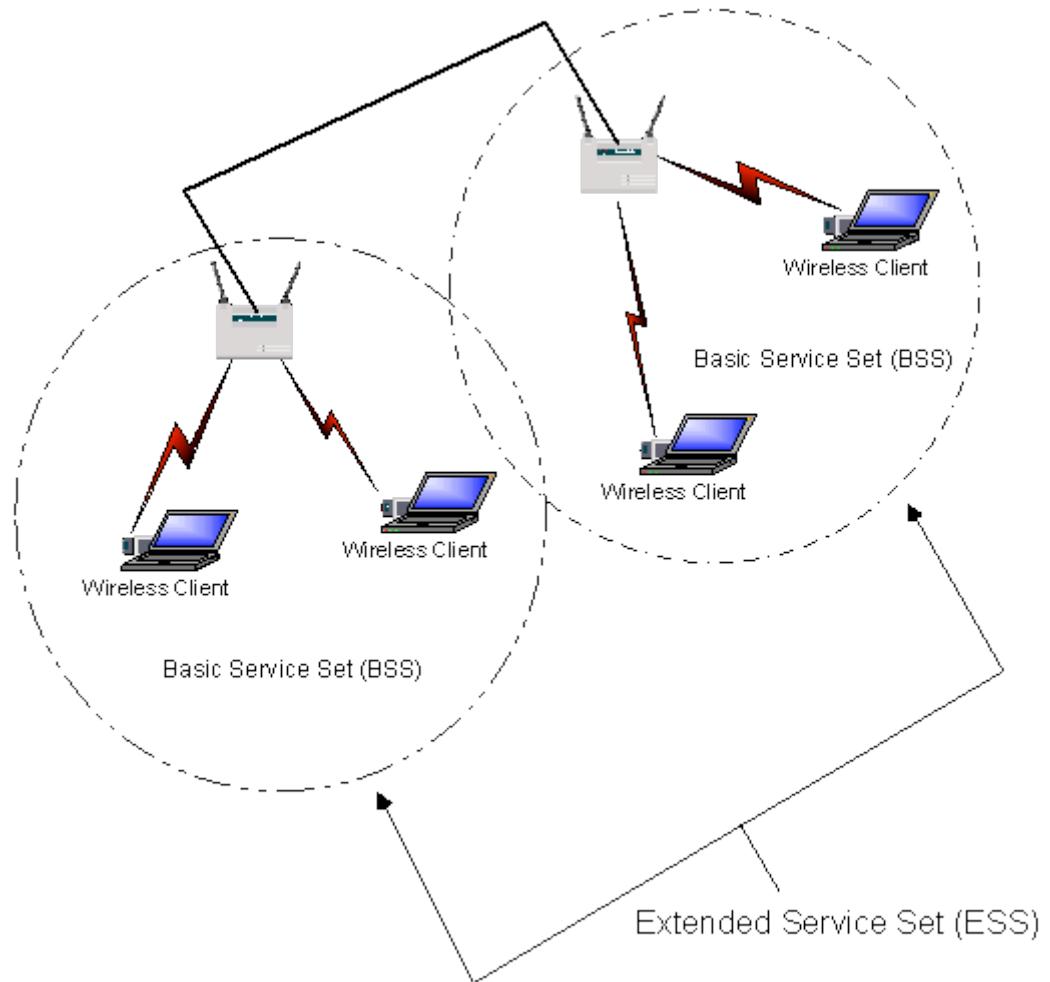


Figura 8 - Esempio di rete con infrastruttura

Concetto di Distribution System in 802.11

Le limitazioni del PHY determinano la massima distanza diretta tra due stazioni che vogliono comunicare. Per molte reti questa distanza è più che sufficiente, per altre è richiesto d'incrementare la copertura. Invece di esistere indipendentemente, una BSS può anche formare un componente di una forma estesa di rete, che è costruita con BSS multiple.

I componenti architetturali usati per interconnettere le BSS sono denominati Distribution System (DS).

L'IEEE 802.11 separa "logicamente" il Wireless Medium (WM) dal Distribution System Medium (DSM). Ciascun mezzo logico è usato per un fine diverso, subordinato al tipo di componente dell'architettura.

Riconoscere che questi due mezzi sono "logicamente" differenti è la chiave per comprendere la flessibilità dell'architettura, cioè non viene fatta nessuna assunzione sulla reale tipologia del mezzo in discussione. Questi possono, quindi, essere definiti di volta in volta per ogni applicazione specifica. Un Access Point è uno STA che fornisce l'accesso al DS. I dati si spostano tra una BSS ed il DS attraverso un AP, questo a sua volta è anche uno STA, quindi un'entità indirizzabile. L'indirizzo utilizzato dall'AP per comunicare sul WM e quello utilizzato per comunicare sul WDM non sono necessariamente lo stesso.

Il DS e la BSS permettono di creare una rete wireless di complessità e dimensione arbitraria. Quando IEEE 802.11 si riferisce a questo tipo di rete parla di Extended Service Set (ESS). La chiave di questo concetto è che l'ESS appare come un LLC di una rete IBSS. La stazione mobile in una ESS può muoversi tra una BSS e l'altra in modo trasparente e non si assume nulla a riguardo della collocazione delle BSS.

Mesh Wireless Distribution System

Quando si parla di Wireless Mesh si intende una rete senza fili in cui gli Access Point sono collegati tra loro attraverso collegamenti radio, tipicamente anch'essi con tecnologia 802.11. Il sistema di distribuzione (DS) è quindi di tipo radio, si parla allora di Wireless Distribution System (WDS). Se il sistema di distribuzione ha una topologia multihop non completamente magliata si ottiene una rete mesh.

Una rete di questo tipo si differenzia notevolmente dalle reti Ad-Hoc (anche note con il termine IBSS – Indipendent Basic Service Set) nelle quali non esiste alcun tipo di controllo centralizzato, ed ogni client si occupa anche di instradare il traffico proveniente da altre stazioni. In una rete mesh, al contrario, i client si associano ad un singolo AP, attraverso il quale inviano e ricevono tutto il loro traffico. Contrariamente a quanto avviene oggi nelle reti ad infrastruttura di una certa dimensione (ovvero in cui sono presenti più Access Point), in una rete mesh il sistema di distribuzione attraverso cui gli AP scambiano dati tra di loro non è di tipo cablato, ma utilizza un'interfaccia radio, ad una frequenza che può coincidere o essere differente da quella utilizzata per la rete di accesso. Reti mesh sono state recentemente sviluppate da produttori commerciali (es. Tropos Networks, Firetide, MeshDelivery, etc.) all'interno di reti cittadine e di campus universitari. In campo accademico viene portato avanti uno sviluppo a scopo di ricerca e sono in fase di standardizzazione le estensioni per il mesh networking di cui si occupa il Task Group 802.11s.

Il forte interesse che si sta sviluppando attorno alle reti mesh senza fili, è dovuto alla possibilità che queste offrono di fornire una connessione a Internet su una vasta area outdoor, con un costo minimo in termini di realizzazione dell'infrastruttura.

CAPITOLO 1 - SCENARIO

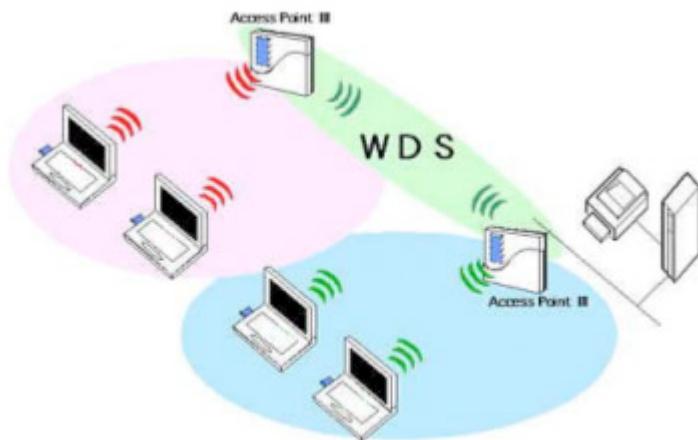


Figura 9 - Esempio di Wireless Distribution System (WDS)

Hardware

Nel lavoro sperimentale sul campo un ruolo fondamentale è giocato dall'hardware utilizzato. Il nostro obiettivo era di concludere con qualcosa di tangibile, con il deployment di una rete reale che dimostrasse la qualità del nostro lavoro. Per fare le scelte più opportune è stato quindi necessario conoscere cosa è disponibile a livello di hardware firmwares e drivers nel modo dell'802.11.

Motherboards e firmwares

Esistono moltissimi vendor che producono hardware standard IEEE 802.11 (cisco, linksys, netgear, 3com, sparklan, etc..). Solitamente un AP 802.11 che si trova in commercio è un sistema composto da:

- Scheda madre con processore di architettura mipsel con frequenza intorno ai 200Mhz
- Memoria Flash tra i 2 Mb e i 16Mb
- Memoria RAM tra gli 8Mb e i 32Mb
- Scheda radio 802.11 direttamente intergrata sulla motherboard o collegata tramite un interfaccia mini-pci.

La flash è quella memoria dove risiede il firmware, ed è suddivisa nel seguente modo:

```
flash_area_type flash_area_list[] = {
//----- -----
//chip_size area_name area_start area_length
//----- -----
{ size2MB, "CFE", 0x1FC00000, 0x40000 },
{ size4MB, "CFE", 0x1FC00000, 0x40000 },
{ size8MB, "CFE", 0x1C000000, 0x40000 },
{ size16MB, "CFE", 0x1C000000, 0x40000 },

{ size2MB, "KERNEL", 0x1FC40000, 0x1B0000 },
{ size4MB, "KERNEL", 0x1FC40000, 0x3B0000 },
{ size8MB, "KERNEL", 0x1C040000, 0x7A0000 },
{ size16MB, "KERNEL", 0x1C040000, 0x7A0000 },

{ size2MB, "NVRAM", 0x1FDF0000, 0x10000 },
{ size4MB, "NVRAM", 0x1FFF0000, 0x10000 },
{ size8MB, "NVRAM", 0x1C7E0000, 0x20000 },
{ size16MB, "NVRAM", 0x1C7E0000, 0x20000 },
```

```

{ size2MB, "WHOLEFLASH", 0x1FC00000, 0x200000 },
{ size4MB, "WHOLEFLASH", 0x1FC00000, 0x400000 },
{ size8MB, "WHOLEFLASH", 0x1C000000, 0x800000 },
{ size16MB, "WHOLEFLASH", 0x1C000000, 0x800000 },

{ 0, 0, 0, 0 }
};

```

Il sistema operativo di molti AP è un linux embedded che viene installato nell'area KERNEL della memoria flash. L'uso di un sistema operativo open source, in questo tipo di prodotti, è la mossa vincente per i vendor che hanno adottato questa strategia di mercato, in quanto ora esistono in rete comunità open source che sviluppano firmware per questi access point. Tra le più importanti ricordiamo OpenWRT, Freifunk e Sveasoft.

La parola firmware indica il vero e proprio sistema operativo dell'apparato, che ne gestisce tutte le funzioni. Il firmware possiede un'interfaccia utente spesso non banale (accessibile via porta seriale, o via rete con i protocolli SNMP, telnet, SSH, HTTP, TFTP o anche FTP per il trasferimento di file di configurazione o nuove versioni del firmware), permette di monitorare ed intervenire sul funzionamento dell'apparato e di modificarne la configurazione.

Parlare di versioni differenti di firmware per wireless access points è un po' come parlare di diverse distribuzioni di linux per PC.

Riassumendo quindi l'ambiente in cui si lavora è un linux essenziale con un hardware con limitata capacità di calcolo.

La memoria flash pone dei limiti per chi sviluppa software per questo tipo di hardware. Infatti, nel caso di caricamento di firmware non funzionante (cosa che accade spesso in fase di testing del codice), l'unico modo per recuperare l'apparato è riprogrammare la memoria flash accedendo direttamente alla scheda madre dall'interfaccia JTAG, con un cavo che va costruito artigianalmente. Inoltre gli AP commerciali non hanno all'esterno un connettore standard JTAG, è necessario smontare l'AP e saldare i componenti necessari. In questo lavoro di tesi è stato realizzato un cavo JTAG, in quanto strumento essenziale per produrre codice per questo tipo di sistemi.



Figura 10 - Cavo JTAG su scheda madre di un Linksys WRT54G

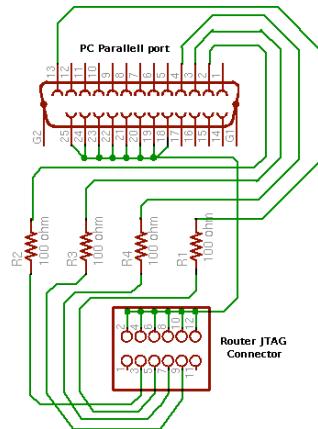


Figura 11 - Schema del cavo JTAG

Gli Access Points appena descritti sono uno dei motivi della diffusione così forte della tecnologia 802.11, che ha invaso il mercato per il suo ottimo rapporto qualità prezzo. Questo hardware, così a buon mercato per l'utente finale, non è però la scelta migliore per testare il codice sviluppato in questo lavoro di tesi, così dopo alcuni mesi di lavoro ci siamo resi conto del fatto che, per sviluppare software, questo non è l'ambiente ideale. Infatti, le limitate capacità di calcolo delle piattaforme appena descritte, non sono adatte al testing di software ancora in fase di ottimizzazione.

Piattaforme più prestanti sono le Soekris, piattaforme general purpose per fare networking, le cui principali caratteristiche sono:

- Processore con architettura x86
- Memoria RAM fino a 256 MB estraibile
- Memoria compact flash fino a 4GB estraibile
- Slot mini-PCI per il collegamento di periferiche come schede radio 802.11

- Compatibile con OS di tipo UNIX

Queste schede sono compatibili con un qualsiasi sistema operativo di tipo Unix (Linux, FreeBSD...), il quale è installato nella Compact Flash. Questa verrà preparata prima tramite un lettore di memorie, ponendoci sopra ad esempio il root file system del sistema operativo linux che risiederà sull'apparato, e solo successivamente verrà montata e configurata sulla motherboard come memoria da cui fare il boot. In alcuni casi la scheda compact flash è saldata sulla motherboard e, quindi, per scriverci sopra l'immagine del root file system è necessario utilizzare il protocollo PXE standardizzato da Intel: tramite un server dhcp ed un server tftp il dispositivo scarica ed installa dalla rete il proprio sistema operativo.

Un vantaggio della Compact Flash è che, quando non è saldata sulla scheda madre, è possibile programmarla da un normale PC senza dover modificare a livello hardware l'apparato. In caso di malfunzionamenti nella fase di testing del sistema operativo risulta più facile intervenire.

E' quindi possibile usare questo tipo di hardware con un linux più familiare in architettura x86. La nostra ricerca di un sistema operativo minimale, che poteva poi essere portato una volta sviluppato su un router commerciale, ci ha spinto a preparare una nuova distribuzione linux: Linuxino.

Questo è un embedded linux costruito da zero, con l'essenziale e le librerie C uclibc, che è stato sviluppato per un apparato minimale che possa svolgere le funzioni di Internet gateway, router, firewall, e wireless access point.

Le principali caratteristiche di Linuxino-x86 sono:

- Kernel 2.6.16
- IDE HD and CF support
- uClibc library
- Java support

Linuxino-x86 è stato progettato per una generica architettura x86 ma è stato ottimizzato per la piattaforma soekris net4801.

Chipsets e Drivers

Per realizzare un ESS, dove la tecnologia di collegamento tra i vari AP è 802.11, i nodi vengono configurati in modalità Wireless Distribution System (WDS). La componente software che si occupa di gestire il WDS è quindi il driver del chipset della scheda 802.11. Non tutti i chipset hanno drivers che supportano la modalità WDS, sebbene questa sia descritta nello standard, molti prodotti commerciali destinati ad un mercato SoHo non la implementano. Quando l'AP è abilitato alla modalità WDS con un altro AP, su entrambi si deve configurare il MAC address del AP con cui si vuole instaurare il canale WDS. Nasce quindi una nuova interfaccia di rete virtuale `wds0` che rappresenta il canale di comunicazione verso l'altro AP. Tutte le interfacce virtuali `wds` e l'interfaccia radio di accesso per le STA vengono poste sotto il controllo di un ethernet bridge, l'AP diviene quindi un wireless bridge. In altre parole per chi avesse più familiarità con lo standard 802.3 possiamo pensare ad ogni AP come un ethernet bridge e ad ogni link WDS con un AP a una porta di questo bridge.

I chipset che hanno drivers per linux che supportano WDS sono:

Chipset	Linux Driver
Broadcom	Broadcom <code>wl.o</code>
Atheros	MadWiFi
Intersil Prism2/2.5/3	HostAP

Il driver broadcom è molto diffuso, è quello utilizzato sul celebre AP Linksys WRT54G. Purtroppo questo driver non è open source, quindi non è possibile modificarlo a scopo di ricerca. Inoltre la versione binaria è disponibile solo per architettura mipsel e quindi non esiste un driver ufficiale per linux utilizzabile in architettura x86.

Un gruppo di programmatore hanno aperto un *project* sul famoso sito di sviluppo di software open source <http://sourceforge.net>, dove si sta lavorando ad un reverse engineering del driver in modo da poter compilare il codice anche per piattaforme diverse dalla piattaforma mipsel, ed avere comunque a disposizione il codice

CAPITOLO 1 - SCENARIO

sorgente. Esiste già una release di questo driver ma non è completa di tutte le funzionalità del driver originale.

Capitolo 2 - RASTA – Radio Aware Spanning Tree Autoconfiguration

Lo spanning tree è un algoritmo utilizzato dai bridge ethernet per realizzare reti complesse con percorsi ridondanti. Una LAN può infatti crescere unendo diversi segmenti di rete attraverso dei bridge, ma la topologia di una LAN non può contenere cicli, ovvero tra ogni coppia di calcolatori deve sempre esistere un solo percorso. Se così non fosse, alcuni pacchetti verrebbero replicati all'infinito sulla rete, con risultati disastrosi. Il bridge, infatti, conosce i MAC address degli host connessi su ogni segmento, ma se riceve un pacchetto con destinazione sconosciuta, o un pacchetto broadcast, lo invia a tutti segmenti. Se esiste un ciclo nella rete, il pacchetto raggiungerà nuovamente il segmento da cui è partito, venendo nuovamente replicato. Questo porterebbe alla proliferazione di infinite copie dello stesso pacchetto sulla rete e quindi alla saturazione della rete stessa. Una rete complessa priva di percorsi ridondanti è però estremamente fragile, perché il guasto di un solo bridge o collegamento la partiziona in due reti che non comunicano tra di loro. In una LAN complessa, è necessario che ci siano dei collegamenti tra bridge ridondanti, ma che alcuni di questi siano mantenuti "fuori servizio" fino a quando non si rendono necessari per soppiare a guasti di altri collegamenti o bridge.

L'algoritmo di spanning tree è un algoritmo distribuito, che opera su tutti i bridge, facendo in modo che in ogni istante la rete sia connessa ma priva di cicli, ovvero che il grafo dei collegamenti disponibili sia "coperto" da un albero. Ciò si ottiene mediante la creazione di una gerarchia di bridge. Un bridge viene individuato come radice dell'albero coprente ("root bridge"), e una parte dei collegamenti tra bridge disponibili viene messa in standby, portando in stato "BLOCKING" alcune delle porte dei bridge. Nel caso in cui un nodo diventa irraggiungibile, oppure cambia il costo di connessione, il bridge cerca di arrivare al nodo attivando i percorsi alternativi che sono in stand-by, ripristinando in questo modo la connettività completa della rete, se possibile. Nella teoria dei grafi, questo problema è noto come Albero spanning. Questo processo avviene periodicamente per cui se si scollega o si rovina un bridge si ricostruisce lo spanning tree e la rete continua a funzionare.

L'algoritmo tende automaticamente a mantenere in funzione i collegamenti di capacità superiore, ma talvolta la scelta di collegamenti da mantenere attivi è inadeguata alle caratteristiche della rete o del traffico che la attraversa. Configurando opportuni parametri sugli switch, ovvero andando ad operare sulla metrica che pesa i vari collegamenti, è possibile influenzare sia la scelta del root bridge che la scelta dei collegamenti da mantenere in servizio.

Tale algoritmo è stato inventato da Radia Perlman e standardizzato in IEEE 802.1D.

In un Wireless Distribution System con topologia mesh, possiamo modellizzare ogni AP come un ethernet bridge e ogni link WDS come una porta del bridge.

E' noto che in una rete tra bridge ethernet non sono ammessi cicli, tipicamente viene usato il protocollo STP (IEEE 802.1D) per configurare un albero minimo di copertura. Il protocollo STP usa una metrica che caratterizza il costo di un percorso in base al suo *bit rate*. Nel caso di una rete ethernet con collegamenti tutti a 100Mbit/s la metrica dell'STP si riduce ad *hop count*. Questo tipo di metrica non è adatta a descrivere il costo di un *link* radio in quanto il bit rate di un link wireless è variabile nel tempo. RASTA modifica i *path cost* del protocollo STP adeguandolo alle necessità di una rete wireless.

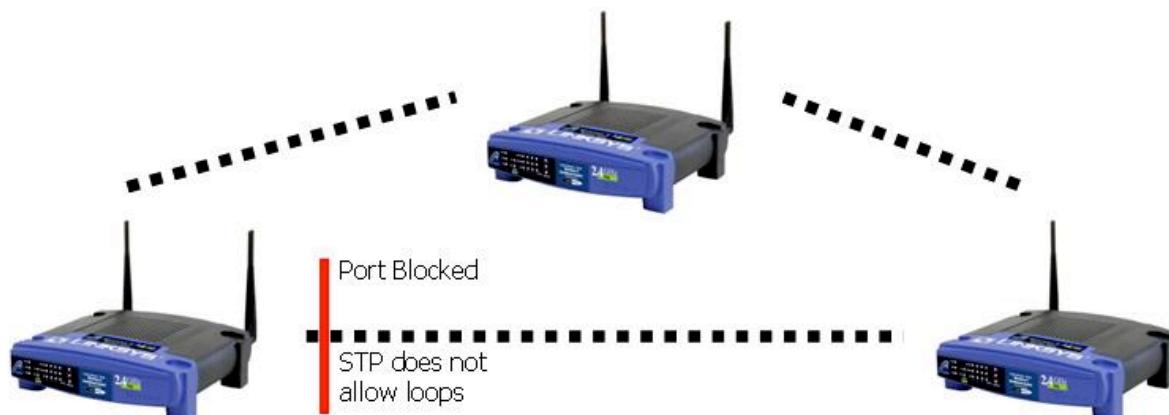


Figura 12 - Uso di Spanning Tree in una Wireless Mesh

Autoconfigurazione

Per instaurare un link WDS tra due AP, su ognuno va configurato l’indirizzo MAC dell’altro. Questa è una operazione banale se effettuata su una rete di due o tre nodi, ma non è banale su una rete di grandi dimensioni. Nasce quindi la necessità di un protocollo di autoconfigurazione per rendere gestibili situazioni dove il numero di nodi è elevato, o addirittura è variabile.

L’autoconfigurazione in RASTA è ottenuta sfruttando i beacon che gli AP inviano periodicamente sul canale radio (di default ogni 100ms). Dall’accensione in poi l’AP ascolta il canale passivamente. Una apposita interfaccia in monitor mode viene attivata, questa sniffa il canale radio senza disturbare in nessun modo il normale esercizio dell’apparato. Se l’AP intercetta un beacon da un altro AP, con un ESSID che soddisfa una condizione di pattern matching, RASTA si preoccupa di stabilire un link WDS con questo AP se non esistesse già. Presupponendo che RASTA sia in esecuzione su ogni AP della rete, i links WDS tra tutti gli AP verranno stabiliti automaticamente tra tutti i nodi che sono in visibilità radio tra loro. Questa metodologia garantisce che ogni coppia di AP in visibilità radio instaura un link WDS, risultato difficile da ottenere con la configurazione manuale in una grande rete, in quanto si dovrebbe prima valutare quali siano i links possibili, e successivamente configurarli sulle coppie di AP.

Radio Aware Spanning Tree

Il protocollo di Spanning Tree (STP) previene la formazione di cicli quando bridges ethernet sono interconnessi con percorsi multipli. L’STP implementa l’algoritmo definito nello standard IEEE 802.1D, scambiando messaggi BPDU con gli altri bridge della rete rivela i cicli e li rimuove spegnendo opportune interfacce su alcuni bridges. Questo algoritmo garantisce che c’è solo un percorso attivo tra due elementi della rete. Ovviamente i percorsi migliori vengono scelti e vengono disattivati i percorsi ridondanti, seguendo una metrica di tipo hop count pesata in base al bit rate del link fisico considerato. Il costo per transitare su un percorso tra un bridge e un altro è quindi funzione della capacità trasmissiva del canale fisico, e viene definito “path cost”. In una rete dove il bit rate è costante su ogni link (ad esempio 100Mbit/s) la metrica dell’STP è un semplice hop count. Questa metrica non è adatta al canale radio, dove non possiamo mai dire quale è il bit rate a priori, e non esiste quindi possibilità di pesare la metrica hop count utilizzata.

L’implementazione del bridge di linux permette di modificare il valore del path cost manualmente. Nella rete mesh “TuscoloMesh” i path cost venivano calibrati manualmente in modo euristico analizzando il comportamento del canale radio. Da qui l’idea di automatizzare questo processo manuale tramite il software RASTA, che utilizza questa interfaccia dell’implementazione software del bridge di linux, per modificare il path cost, e far diventare la metrica radio aware.

Statistiche Radio Passive

Nessun protocollo di spanning tree con una metrica radio aware è ancora stato implementato dalla comunità scientifica, ma esistono ormai molti protocolli di routing radio aware (OLSR, AODV etc). Molti di questi protocolli per caratterizzare il canale radio introducono dei pacchetti di probe, e calcolano delle statistiche sulla base di informazioni estrapolate da questi. Questa metodologia introduce a volte un significativo overhead in rete. RASTA invece usa un approccio passivo, senza introdurre nessun tipo di overhead, ascolta il canale e colleziona informazioni dai pacchetti ricevuti.

Il metodo delle statistiche attive ha il vantaggio di essere indipendente dall'hardware, infatti il demone di routing non interroga mai l'hardware per avere informazioni. Nel caso di statiche passive è necessario prelevare le informazioni dall'hardware utilizzato, ed è quindi necessario interagire con il driver. Per questo motivo RASTA non è impiegabile con un qualsiasi chipset 802.11 ma solo con i chipsets per i quali è stato sviluppato il software.

Per utilizzare RASTA è quindi necessario usare un chipset con driver che supporta il monitor mode, ovvero che crea una interfaccia virtuale in modalità monitor, mentre l'AP è in regolare funzione. Inoltre il driver deve fornire l'header di livello PHY ARPHDR_IEEE80211_PRISM che viene chiamato comunemente Prism Header, nome che viene dal chipset Prism che è stato il primo ad introdurre questo tipo di header di livello PHY, che fornisce informazioni radio sul frame ricevuto. I drivers che hanno queste features e che sono supportati da RASTA sono il MadWiFi ed il Broadcom wl.o.

```

Frame 22 (272 bytes on wire, 272 bytes captured)
Prism Monitoring Header
  Message Code: 65
  Message Length: 144
  Device: eth1
    Host Time: 0x4845 (DID 0x1041, Status 0x0, Length 0x4)
    MAC Time: 0xa57c8c4 (DID 0x2041, Status 0x0, Length 0x4)
    RSSI: 0xfffffff6 (DID 0x4041, Status 0x0, Length 0x4)
    SQ: 0x5b (DID 0x5041, Status 0x0, Length 0x4)
    Data Rate: 1.0 Mb/s
    Frame Length: 0x80 (DID 0xa041, Status 0x0, Length 0x4)
IEEE 802.11
  Logical-Link Control
  Internet Protocol, Src: 192.168.1.148 (192.168.1.148), Dst: 192.168.1.2 (192.168.1.2)
  Transmission Control Protocol, Src Port: 1129 (1129), Dst Port: 22 (22), Seq: 676, Ack: 792, Len: 52
  SSH Protocol

```

Figura 13 - Snapshot di Ethereal: il Prism Monitoring Header

Nella figura possiamo vedere un esempio di prism header.

Descriviamo il significato dei campi:

- Message Code: Numero di codice del messaggio
- Message lenght: Lunghezza in byte del prism header
- Device: nome dell’interfaccia sulla quale è stato ricevuto il pacchetto
- Host Time: timestamp di quando il pacchetto è stato prelevato dal card buffer
- MAC Time: timestamp di quando il pacchetto è stato ricevuto dalla scheda
- RSSI: Potenza alla quale è stato ricevuto il pacchetto
- SQ: Qualità del segnale
- Data Rate: Rate al quale il pacchetto è stato ricevuto
- Frame Length: lunghezza del frame 802.11

Questa prima versione di RASTA utilizza solo il campo RSSI per generare le statistiche che caratterizzano il link radio.

Un possibile problema potrebbe essere quello che in assenza di traffico, o con troppo poco traffico, non è possibile caratterizzare correttamente il canale. Pensando però a reti ethernet di grandi dimensioni sappiamo che un minimo quantitativo di traffico è sempre presente, garantito dalla presenza delle BPDU dello STP e dai pacchetti broadcast dei vari protocolli della rete IP. In una rete ethernet di grandi dimensioni molto traffico è generato solamente dal protocollo ARP.

Organizzazione software delle statistiche

Per contenere le statistiche che caratterizzano ogni link è stata definita la struttura `statistica_link` dichiarata nel file `statistica.h`

```
struct statistica_link {
    unsigned char MAC_AP[6];
    int record_number;

    //statiche generali link
    int bestRSSI;
    int worstRSSI;
    int avgRSSI;
```

```

//statistiche colore 5
int bestRSSIcolor5;
int worstRSSIcolor5;
int avgRSSIcolor5;

//statistiche colore 6
int bestRSSIcolor6;
int worstRSSIcolor6;
int avgRSSIcolor6;

struct statistica_link* next;

sem_t mutex;
} ;

```

Non sapendo a priori il numero di link WDS che RASTA deve monitorare, varie istanze di statistica link vengono controllate tramite una lista gestita da puntatori.

Questa prima versione di RASTA utilizza solo il campo RSSI del prism header. Per ogni link si registra il valore minimo e massimo e la media dell’RSSI. La media viene calcolata con la formula: $\text{average} = (1-P) * \text{average} + P * \text{newvalue}$

“P” è un parametro specificato nel file di configurazione, il valore di default è 0,1. La funzione che calcola la media è una funzione “passa basso”, ovvero da una lenta variabilità al valore che viene calcolato di media ed il parametro P serve a calibrare quanto questa variabilità deve essere lenta.

Inoltre se lavora con YaQoS le statistiche sono separate per 2 diversi colori di VLAN, ma questo verrà spiegato in seguito nel capitolo 3.

Aggiornamento dei path cost

I path cost vengono aggiornati chiamando opportune system call. Un thread apposito parte all’avvio del demone, e si occupa soltanto di aggiornare i path cost ciclicamente allo scadere del timeout definito nel file di configurazione. Il thread scorre tutta la lista delle statistiche ed aggiorna il path cost di ogni link. Il codice di questo thread è nel file `comandi.c`.

```

...
do
{
prec=prec->next;
record=record+1;

//bridge commands here

if (!DISTRIBUTORE) {

memset(command,0,50);
if (BROADCOM) sprintf(command,"brctl setpathcost br0 wds0.49153.5 %d",prec->avgRSSI*-2);
if (ATHEROS) sprintf(command,"brctl setpathcost br5 ath3.5 %d",prec->avgRSSI*-2);
if (DEBUG) printf("%s\n",command);
system(command);
}

if (DISTRIBUTORE) {
memset(command,0,50);
if (BROADCOM)
sprintf(command,"brctl setpathcost br5 wds0.4915%d.5 %d",record,prec->avgRSSIcolor5*-2);
if (ATHEROS)
sprintf(command,"brctl setpathcost br5 ath%d.5 %d",record,prec->avgRSSIcolor5*-2);
system(command);
memset(command,0,50);
if (BROADCOM)
sprintf(command,"brctl setpathcost br6 wds0.4915%d.6 %d",record,prec->avgRSSIcolor6*-2);
if (ATHEROS)
sprintf(command,"brctl setpathcost br6 ath%d.6 %d",record,prec->avgRSSIcolor6*-2);
system(command);
}
}

while (prec->next!=NULL);
prec=primo;
succ=primo;
...
}
```

I path cost vengono settati secondo la seguente tabella:

```
/*
until RSSI    Path cost
-50          100
-60          120
-70          140
-80          160
-100         200
*/
```

Ovviamente questa tabella di valori è molto grezza. Se nella normale implementazione dell'STP si associa un valore di path cost al bit rate del canale fisico, qui si associa un valore di path cost alla potenza di segnale ricevuta sul canale fisico. In realtà non si è lavorato molto sul definire una metrica perfetta, ma l'obiettivo primario era quello di sviluppare un software che introducesse il concetto di metrica radio aware nel protocollo di Spanning Tree, in modo da poter implementare la metrica più adatta nel codice una volta che questa sia stata maturata dalla comunità scientifica. Inoltre, per il modo in cui è stata concepita l'interazione tra RASTA ed il protocollo di Spanning Tree, sarà possibile utilizzarlo anche con l'implementazione del Rapid Spanning Tree Protocol (RSTP) che a breve sarà disponibile per il Kernel di Linux. In definitiva si è voluto costruire un tool per dare un valore aggiunto alle implementazioni dei protocolli di Spanning Tree in modo elastico e configurabile.

Impostare il timeout di validità dei path cost non è banale. Un timeout troppo corto potrebbe portare a delle statistiche poco consistenti, soprattutto se la rete è molto scarica, e quindi decidere per dei valori di path cost non indicativi della realtà, mentre un timeout troppo lungo potrebbe non intervenire per lungo tempo in caso di degradazione di un link. In genere è meglio scegliere un timeout più lungo che uno troppo corto. Inoltre dato che è ancora in fase di sviluppo la funzione definitiva, che a partire dalle informazioni radio genera i path cost, lasciare un timeout configurabile era d'obbligo. Dalle misure fatte in laboratorio e dalle prove fatte su reti in esercizio

si è scelto di inserire come default del file di configurazione un timeout di 60 secondi.

Files di configurazione

RASTA cerca il suo file di configurazione `rasta.conf` nella cartella `/etc/rasta/`. Nel caso vogliate spostare questo file in un altro percorso, a seconda della vostra distribuzione linux, potete modificare il codice della funzione di lettura del file di configurazione che si trova nel file `readconf.c`. I valori di defaults scelti nel file di configurazione di RASTA sono stati scelti in modo da dare buone prestazioni in una generica rete mesh. E' buona regola però calibrare al meglio i valori nel file di configurazione per una prestazione ottimale sulla propria rete mesh. In particolare, come già ampiamente discusso, fate attenzione al timeout dei path costs.

I valori configurabili sono:

- Chipset/Driver (nella versione corrente Broadcom o Atheros)
- Sniffing interface (dipende dall'hardware)
- ESSID della rete
- Peso “P” nella funzione di media pesata: $\text{average} = (1-P) * \text{average} + P * \text{newvalue}$
- Messaggi di Debug
- Timeout dei path cost
- Attivazione del supporto VLAN 802.1Q per YaQoS

Scripts

Oltre al file di configurazione esistono nella directory `/etc/rasta` due scripts, `madwifi-ng.script` e `wl.script`. Per configurare l'AP in base al tipo di scheda wireless presente, RASTA tramite una chiamata di sistema, lancia l'opportuno script. La scelta di lasciare i comandi di configurazione al boot al di fuori del file binario di RASTA è dovuta alla forte differenza tra le varie piattaforme. Usando lo script è possibile personalizzarlo per lo specifico AP su cui si sta facendo

girare RASTA senza dove ricompilare necessariamente il sorgente. Vediamo ad esempio lo script madwifi-ng.script:

```
wlanconfig ath0 destroy
wlanconfig ath1 destroy
wlanconfig ath0 create wlandev wifi0 wlanmode ap
iwconfig ath0 essid peace-and-love
iwconfig ath0 channel 3
echo Creating monitor iface
wlanconfig ath1 create wlandev wifi0 wlanmode monitor
echo Enabling ath1 with ifconfig
ifconfig ath1 promisc up
ifconfig ath0 down
ifconfig ath0 up
echo RASTA ok!
```

In questo script ad esempio, per problemi del driver di madwifi-ng, l’interfaccia ath0 viene distrutta e ricreata, e viene riconfigurato l’ESSID dell’apparato. E’ intuitivo che questo genere di operazione sia più flessibile contenuta in uno script piuttosto che nel codice sorgente, dove una modifica necessita di compilare per avere un nuovo file binario.

Program Flow

Rasta è scritto in C. Il codice è compilabile sia per architettura mipsel che per x86, con librerie glibc e ulibc. Il codice è organizzato in *threads* concorrenti. In figura vediamo il diagramma di flusso delle operazioni che esegue di codice:

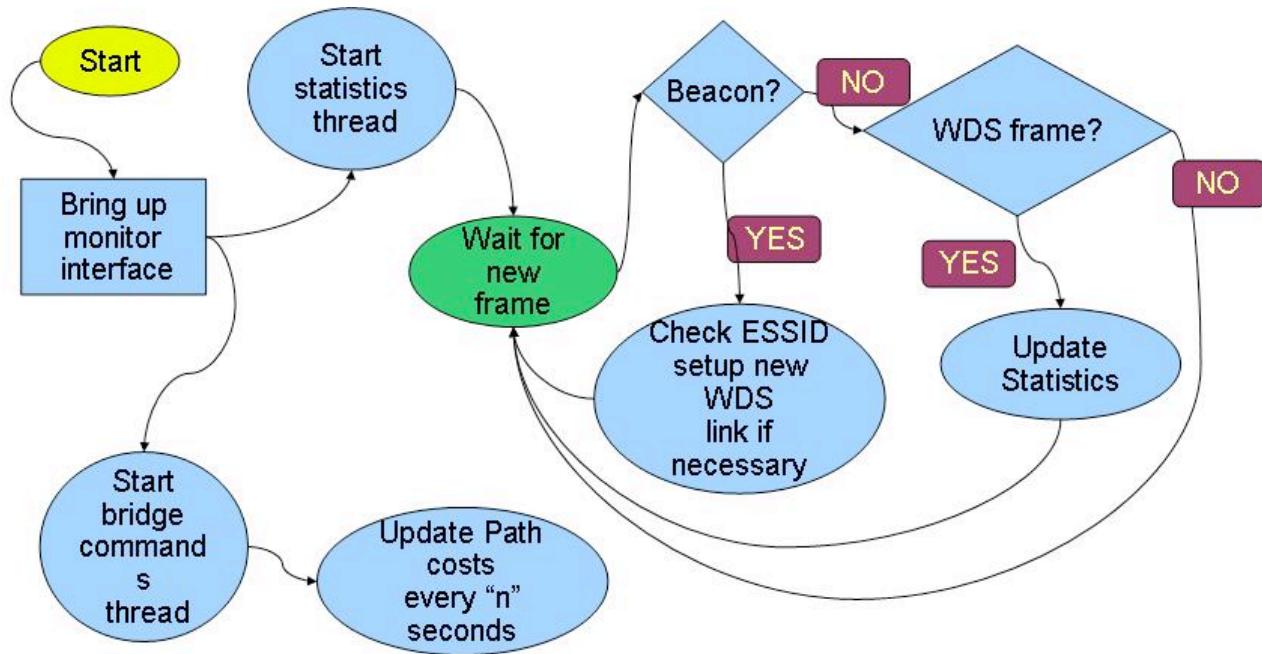


Figura 14 - Diagramma di flusso del codice di RASTA

Capitolo 3 – YaQoSa – Yet Another Quality of Service Approach

In questo lavoro di tesi si è lavorato per trovare dei meccanismi di QoS adatti ad una rete mesh 802.11. L’approccio utilizzato, dal punto di vista dell’architettura di rete, è stato quello di lavorare completamente a layer 2. L’idea è di usare i meccanismi di QoS dello standard VLAN 802.1Q nel Distribution System (DS), in modo da suddividere il traffico in diverse classi di priorità di servizio. Per fare questo, l’accesso dei frames 802.11 al DS, e l’uscita da questo, sono monitorate da un apposito software che aggiunge o rimuove i tags 802.1Q.

Questa idea è stata implementata in un pacchetto software per linux chiamato YaQoSa: Yet Another Quality of Service Approach.

Dal punto di vista implementativo si è adottato un approccio di tipo user space: sfruttando caratteristiche già implementate nel kernel di linux e devices di tipo tun/tap, è stato possibile sviluppare software in modo veloce.

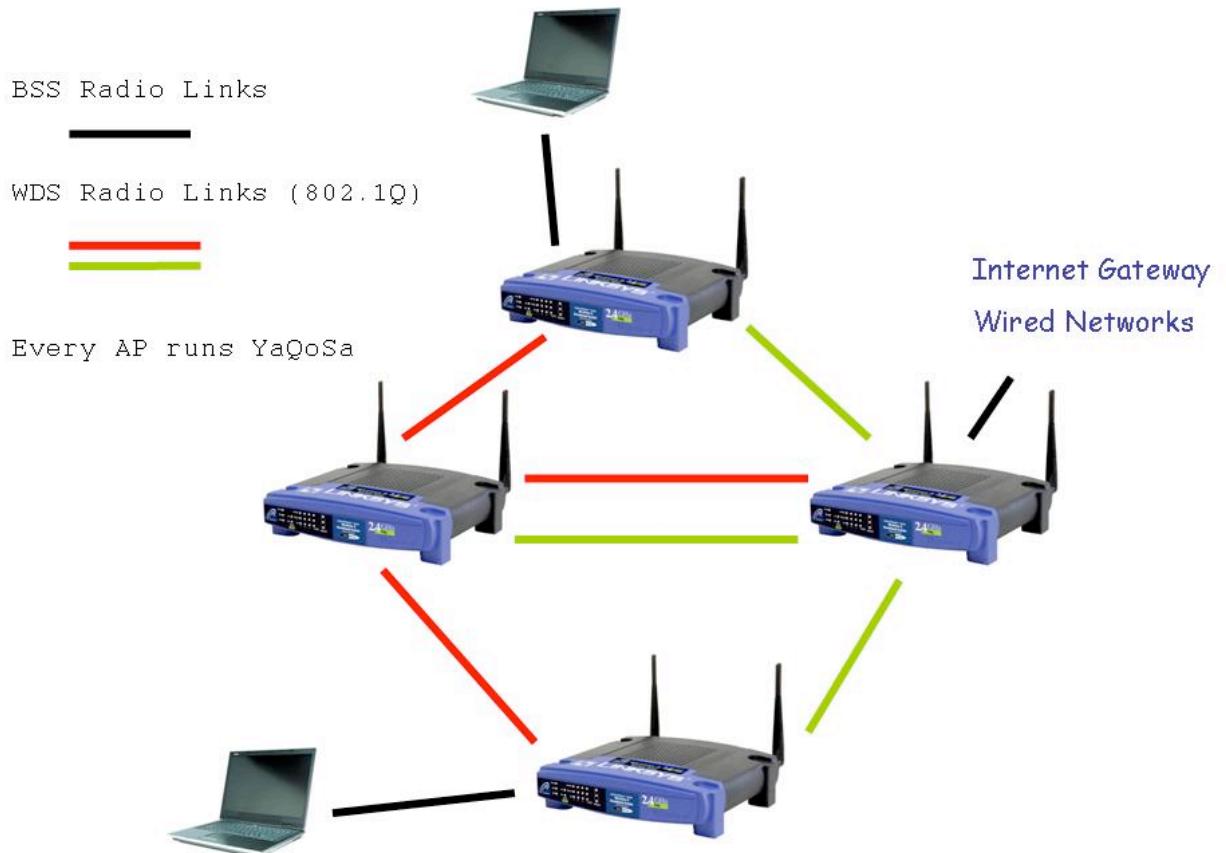


Figura 15 - Esempio di tagging VLAN nel DS

QoS – Quality of Service

Nel campo delle reti a commutazione di pacchetto e nelle reti di calcolatori, il termine Qualità di servizio o più semplicemente QoS (dall'inglese Quality of Service) è usato per indicare la probabilità che una rete possa soddisfare particolari specifiche sul traffico; spesso è usata in modo informale per riferirsi alla probabilità che un pacchetto possa riuscire a passare tra due punti di un rete.

Quando in principio è stata creata Internet, non era stata percepita la necessità di un certo QoS per le applicazioni. Infatti l'intera Internet è un sistema che segue la filosofia del best effort, cioè il sistema garantisce di fare tutto il possibile per portare a termine un'operazione, ma non garantisce affatto che l'operazione verrà compiuta, e tanto meno in che modo. Sebbene il protocollo IP prevedesse 4 bit per il tipo di servizio (type of service) e 3 per la precedenza di ciascun pacchetto, questi bit sono largamente inutilizzati. Ci sono molte cose che possono accadere ad un pacchetto che

viaggia da una sorgente verso la sua destinazione, in particolare possono insorgere i seguenti problemi:

- Pacchetti persi o *dropped packets* - i router possono non riuscire a consegnare alcuni pacchetti se questi arrivano quando i loro *buffers* sono già pieni. Alcuni, nessuno, o tutti i pacchetti possono venir scartati (*dropped*) a seconda dello stato della rete, ed è impossibile prevedere in anticipo che cosa accadrà. L'applicazione ricevente, non prima di aver atteso un tempo ragionevole per credere di essersi persa un pacchetto, deve chiedere che l'informazione venga ritrasmessa, causando anche gravi ritardi (*delay*) nella trasmissione complessiva.
- Ritardo o *delay* – un pacchetto può richiedere parecchio tempo prima di raggiungere la destinazione a causa di lunghe attese in coda ad un router oppure a causa di un instradamento indiretto per evitare un tratto congestionato. Tuttavia potrebbe arrivare anche molto velocemente seguendo un collegamento veloce e diretto. Il ritardo perciò è molto imprevedibile ed è anche la causa della consegna fuori ordine.
- Consegnata fuori ordine o *out-of-order* – quando una collezione di pacchetti affini viene instradata attraverso internet, pacchetti diversi possono prendere strade diverse, ciascuna soggetta ad un ritardo diverso. Il risultato è che i pacchetti possono arrivare a destinazione con un ordine diverso di quando sono stati inviati. Questo problema rende necessario speciali protocolli aggiuntivi per riordinare i pacchetti fuori ordine una volta che sono giunti a destinazione.
- Errore – a volte i pacchetti sono instradati verso i router sbagliati, o combinati tra loro, o semplicemente vengono corrotti da un disturbo sulla linea. Il ricevente deve individuare la cosa e trattare il pacchetto come se fosse stato perso richiedendone la ritrasmessione.

Ci sono fondamentalmente due modi per fornire garanzie sulla Qualità di servizio. La prima consiste semplicemente nel fornire risorse in quantità, abbastanza da soddisfare la domanda di picco attesa, con un sostanziale margine di sicurezza. Una soluzione semplice, ma alcuni credono che in pratica sia troppo costosa, e non è

applicabile se la domanda di picco cresce più velocemente di quando predetto: disporre nuove risorse richiede tempo.

Il secondo modo è quello di richiedere di fare delle prenotazioni, che vengono accettate solo se i routers sono capaci di servirle adeguatamente. Naturalmente, le prenotazioni possono venire tariffate, differenziando così il traffico QoS da quello *best effort*.

Ci sono due varianti conosciute:

- IntServ
- DiffServ

DiffServ sono tipicamente usate con:

- weighted round robin, WRR.
- RED, WRED - Abbassa la possibilità che vengano droppati pacchetti.
- Traffic shaping
- **VLAN IEEE 802.1p e IEEE 802.1D.**

Apparati di rete che supportino DiffServ o magari IntServ si dicono apparati di rete *multilayer*. Uno switch che supporti DiffServ o magari IntServ è chiamato Multilayer switch.

Lo standard IEEE 802.1Q

IEEE 802.1Q è uno standard che permette a più reti virtuali VLAN di condividere lo stesso collegamento fisico senza pedita di informazioni tra un'apparato e un'altro.

802.1Q è il nome del protocollo di encapsulamento utilizzato nel processo di trunking nelle reti Ethernet.

802.1Q non incapsula il frame originale, ma modifica 4 byte dell'header.

I primi 2 byte modificati riguardano il tag protocol identifier TPID. Questo contiene il tag EtherType che viene cambiato in 0x8100, numero che evidenzia il nuovo formato del frame.

I successivi 2 byte riguardano il tag control information TCI così suddiviso:

- user_priority: questo campo a 3 bit può essere utilizzato per indicare un livello di priorità per il frame. L'utilizzo di questo campo è definito in: IEEE 802.1p.

- CFI: campo di 1 bit che indica se i MAC address nel frame sono in forma canonica.
- VID: campo di 12 bit che indica l'ID delle VLAN, che possono così essere fino a 4096 (ossia 212). Di queste, la prima (VLAN 0) e l'ultima (VLAN 4095) sono riservate, quindi gli ID realmente usabili sono 4094.

Il resto del frame Ethernet rimane identico all'originale.

Poiché con la modifica di questo header il frame viene cambiato, il meccanismo di incapsulazione di 802.1Q richiede il ricalcolo del campo FCS nel trailer Ethernet.

L'implementazione user space

YaQoS si pone tra il DS ed il BSS:

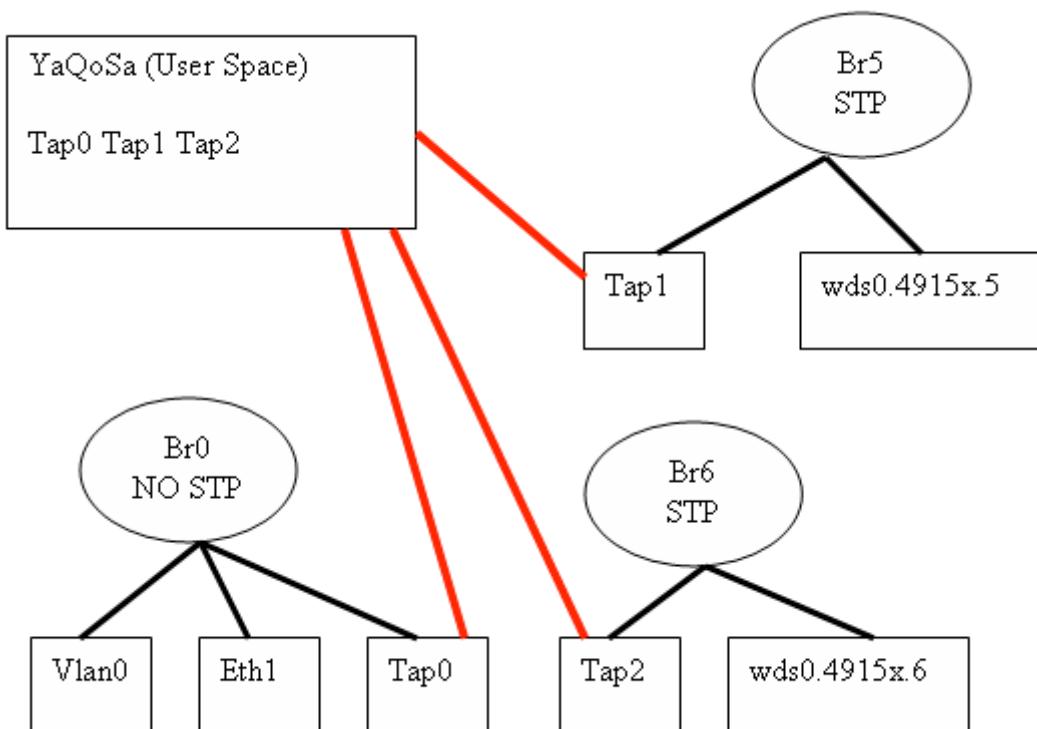


Figura 16 - Architettura di YaQoS

L’obiettivo è far passare tutto il traffico tra BSS e DS attraverso un modulo di YaQoS chiamato “Distributore”. Per fare questo è stata realizzata l’architettura in figura. Esistono su ogni AP della rete mesh più bridges completamente scollegati tra di loro. Ad ognuno di questi bridge è assegnata un interfaccia di tipo `tap`, attraverso la quale dialogano con YaQoS che gira in user space (i bridges girano in kernel space). In pratica su ogni apparato che fa parte della mesh è presente un bridge che contiene le interfacce che fanno parte della BSS e altri bridges colorati che fanno parte del DS. Il DS è così composto da mesh networks virtuali, ognuna con un proprio albero di copertura e ognuna con assegnato un certo tipo di traffico. I diversi alberi colorati presenti nel DS sono gestiti dal protocollo STP ed è anche possibile far girare sullo stesso apparato RASTA insieme a YaQoS per gestire al meglio gli Spanning Trees.

In questo modo è come se usassimo MSTP. Possiamo parlare di Fake MSTP. Il Multiple Spanning Tree Protocol è un algoritmo che fornisce un albero minimo di copertura senza cicli per una rete Ethernet con delle VLAN 802.1Q. Nel caso ci sia solo una Virtual LAN (VLAN) in questa rete, allora l’MSTP farebbe lo stesso lavoro del tradizionale STP. Nel caso di più VLAN MSTP organizza un albero di copertura per ogni diverso colore di VLAN, organizzando la rete in zone tra di loro gerarchiche. Ovviamente l’MSTP è molto più efficiente, ma anche molto più complesso, del nostro Fake MSTP, ma riteniamo che la nostra soluzione sia sufficientemente buona considerato il tipo di rete su cui andiamo ad operare.

Questo tipo di approccio, che consiste nell’operare politiche di qualità di servizio sul backbone di trasporto, si adotta spesso in scenari dove viene utilizzato MPLS, nel nostro caso però tutto è implementato con standards 802.1.

Il tagging e il detagging 802.1Q dei frames che entrano ed escono dal DS è effettuato dal kernel di linux, in quanto non sono i bridges ad essere interfacce 802.1Q, ma i links WDS. I frames che escono dai `brX` non sono taggati, ma verranno taggati alle interfacce `wdsx.x` dove il tag 802.1Q è abilitato. Così facendo non è stato necessario scrivere codice per taggare e staggare i pacchetti, in quanto questa funzionalità è già implementata nell’ambiente in cui lavoriamo.

L’instradamento verso l’uno o l’altro bridge, ovvero la lettura e la scrittura vera e propria dei frames sulle interfacce `tap`, viene effettuata da un modulo software che abbiamo chiamato distributore. La logica con la quale il distributore deve eseguire queste operazioni è invece implementata in un altro modulo chiamato decisore. Il

software è organizzato in due moduli principali, il distributore che gestisce il tagging ed il detagging, ed il decisore che invece istruisce il distributore sui colori di tag da assegnare:

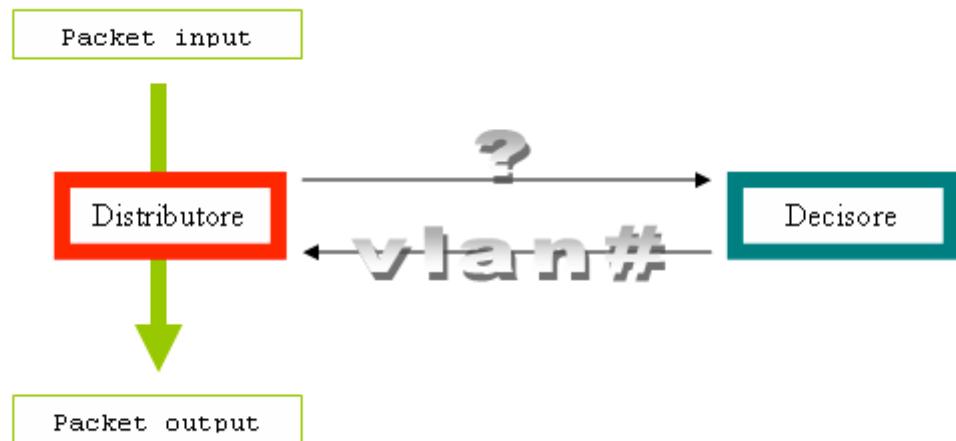


Figura 17 - Architettura Distributore/Decisore di YaQoS

Regole statiche e dinamiche

Il distributore interroga il decisore per sapere su quale colore di VLAN il frame che sta processando deve essere spedito. Il decisore fa il parsing di una lista dinamica di regole e restituisce al distributore un numero intero, che rappresenta appunto la VLAN di destinazione. Se il frame non può essere accettato nel DS viene restituito per convenzione 0, ed il frame viene lasciato a terra.

All'avvio del demone delle regole statiche vengono eventualmente caricate da file di configurazione. Sono stati implementati 8 diversi tipi di regole:

```

/*Mappatura regole

1: check sul Mac Address Sorgente
2: check sul Mac Address Destinazione
3: check sull'IP Sorgente
4: check sull'IP Destinazione
5: check su essere TCP

```

```

6: check su essere UDP
7: check su porta TCP specifica
8: check su porta UDP specifica
*/

```

Non sapendo a priori il numero di regole che YaQoS deve gestire, queste vengono controllate tramite una lista gestita da puntatori.

Ogni regola è conservata dentro una struttura del tipo `rules`, che è scritta nel file `rules.h`:

```

struct rules {
    int rule_type; //8 tipi, vedi mappatura

    //mac Addresses
    unsigned char MAC1[6]; //sorgente
    unsigned char MAC2[6]; //destinazione

    //IP Addresses
    unsigned int saddr; /* indirizzo sorgente 32 bit */
    unsigned int daddr; /* indirizzo destinazione 32 bit */

    //Service Access Point SAP
    unsigned char SAP;

    //Port Number
    unsigned short port;

    //bridge di uscita
    int output;

    //traffico generato dalla stazione (per future
    implementazioni)
    unsigned long int bw;

    struct timeval timeout;
}

```

```
struct rules* next;
};
```

Ogni regola è caratterizzata da 5 elementi:

1. Il suo tipo,
2. Il valore su cui viene fatto il controllo (MAC address, IP, SAP o port number)
3. Il colore di VLAN in uscita
4. Il suo timeout
5. Il puntatore alla regola successiva nella lista

All'avvio del demone YaQoSa controlla se deve abilitare la modalità dynamic. Questa è una speciale modalità che permette l'uso di regole dinamiche. Quando il decisore sta processando un frame, nel caso non trovi un match con una regola statica, crea una regola ad-hoc per quel frame e applica un timeout a questa regola. Ogni volta che una regola sta per essere applicata, YaQoSa controlla la validità del timeout. Se questo è scaduto viene creata una nuova regola.

Le regole statiche non hanno timeout e quindi il valore è posto a 0. Le regole inserite con la modalità dynamic vengono poste alla fine della lista, in modo che le regole statiche abbiano sempre una priorità più alta.

Nella attuale implementazione di YaQoSa, le regole in modalità dynamic hanno come obiettivo quello di bilanciare il traffico tra i vari alberi. Per implementare tale funzionalità YaQoSa tiene traccia di quanto traffico scorre sui vari alberi.

Se l'opzione dynamic è attivata (tramite file di configurazione), il decisore crea una nuova regola quando un pacchetto non fa match con nessuna regola esistente. La nuova regola consiste nel passare il pacchetto alla VLAN più scarica come numero di bytes transitati in un intervallo di tempo. Le regole create con il modo dynamic hanno un timeout definibile nel file di configurazione, dopo questo timeout le regole vengono cancellate. Se una regola viene cancellata, per via dello scadere del suo timeout, il pacchetto dovrà di nuovo passare per la modalità dynamic del decisore ed essere assegnato ad una nuova regola.

Files di configurazione

YaQoS ha due file di configurazione:

- yaqosa.conf
- rules.conf

In yaqosa.conf è possibile abilitare il *debugging* e la modalità *dynamic*

```
#Configuration parameters that tune YaQoS behavior
#dynamic 0=no 1=yes
dynamic 1
#debug, print debug messages 0=no 1=yes
debug 1
```

In rules.conf vengono inserite le regole statiche su cui si basa la logica iniziale del decisore:

```
# YaQoS FILE DI CONFIGURAZIONE

# LE RIGHE CHE INIZIANO PER # SONO COMMENTI

# SINTASSI:
# TIPOREGOLA TRIGGER OUTPUT(COLOREVLAN)

# ATTENZIONE ALL'ORDINE DELLE RIGHE, OGNI TARGET è UN
TERMINATING TARGET NELLA LISTA DELLE REGOLE

1 00E0187FBABA 5
2 192.168.0.8 6
```

Program Flow

YaQoS è scritto in C. Il codice è compilabile sia per architettura mipsel che per x86, con librerie glibc e ulibc. In figura vediamo nei diagrammi di flusso le operazioni che eseguono il codice del distributore e il codice del decisore:

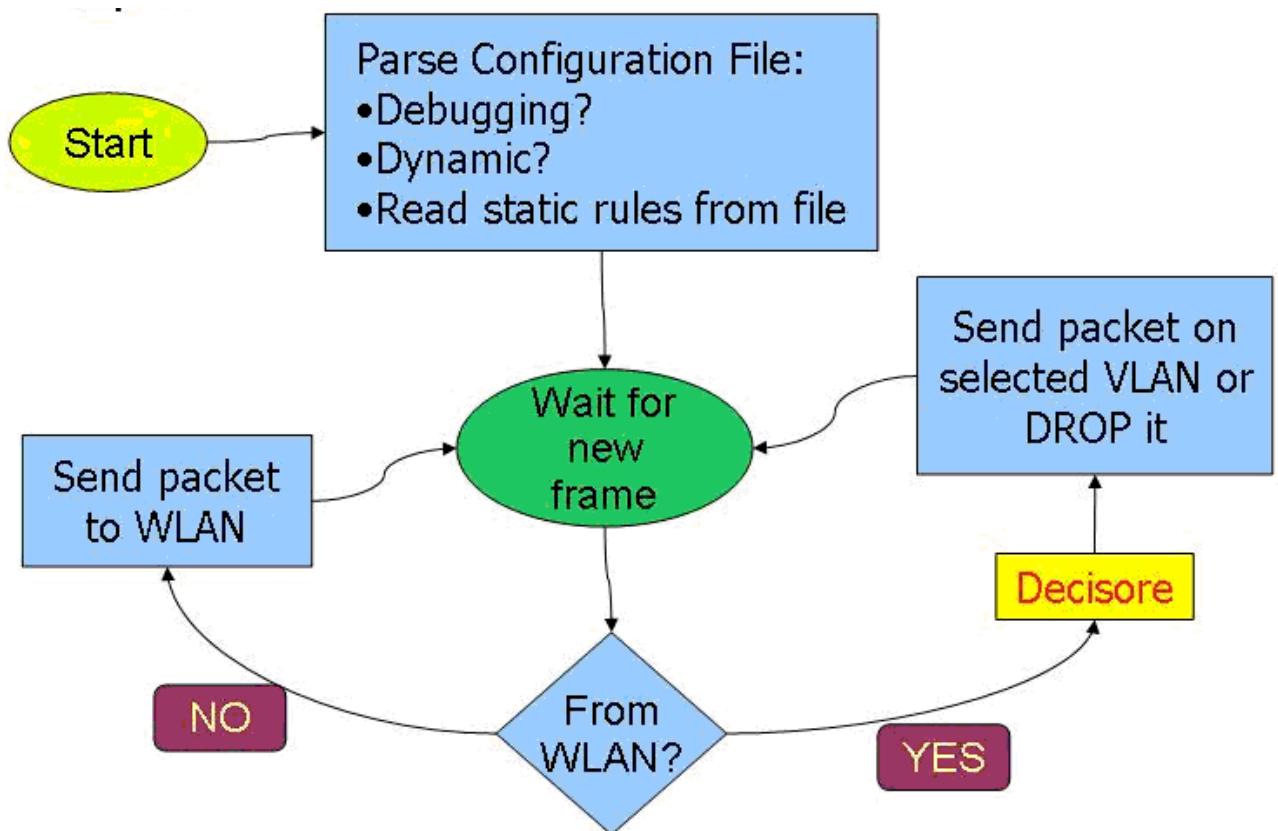


Figura 18 - Diagramma di flusso del modulo Distributore

All'avvio del suo demone YaQoS legge due files di configurazione. Questi vengono cercati nella cartella /etc/yaqosa/ e sono yaqosa.conf e regole.conf. Nel primo file è possibile abilitare il debugging verboso ed è possibile abilitare la modalità dynamic che descriveremo meglio in seguito. Il file regole.conf contiene invece delle regole statiche che è possibile introdurre per mandare sempre alcuni frames su certi alberi prefissati.

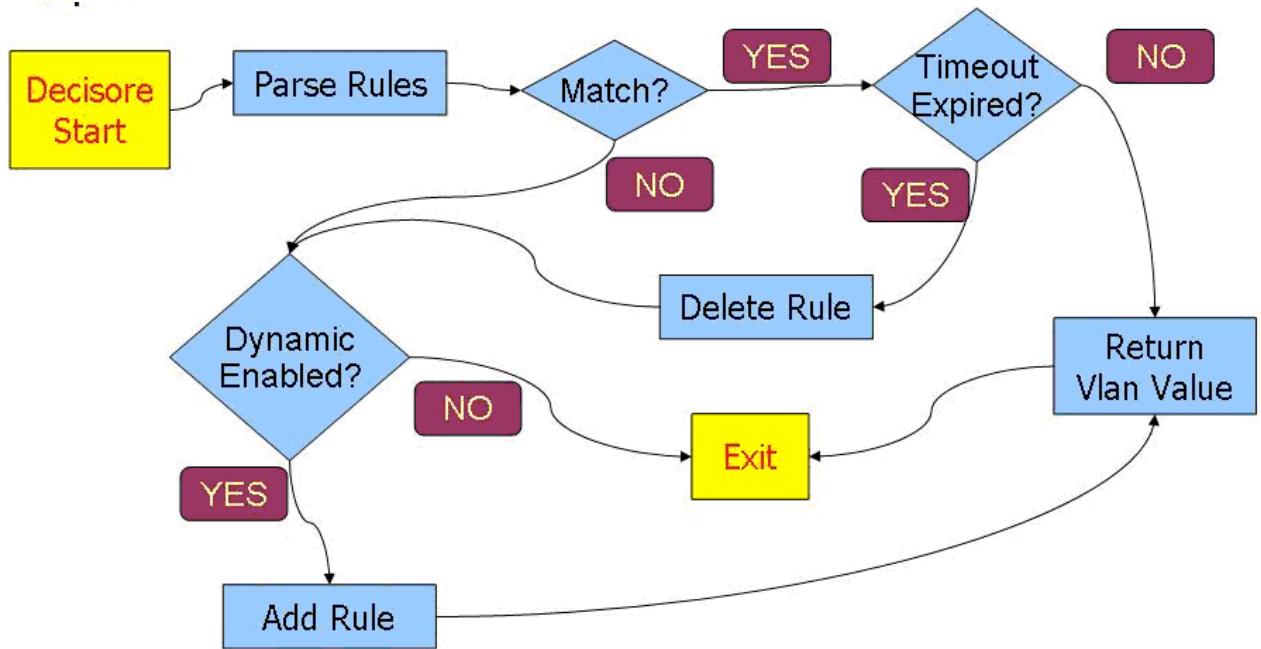


Figura 19 - Diagramma di flusso del modulo Decisore

Capitolo 4 – Testbed Sperimentale

Durante questo lavoro di tesi, la progettazione dei sistemi e del software è stata svolta parallelamente alla sperimentazione sul campo. In questo capitolo vedremo come sono stati sviluppati i software nei laboratori dell'università di Tor Vergata sulle piattaforme hardware disponibili sul mercato.

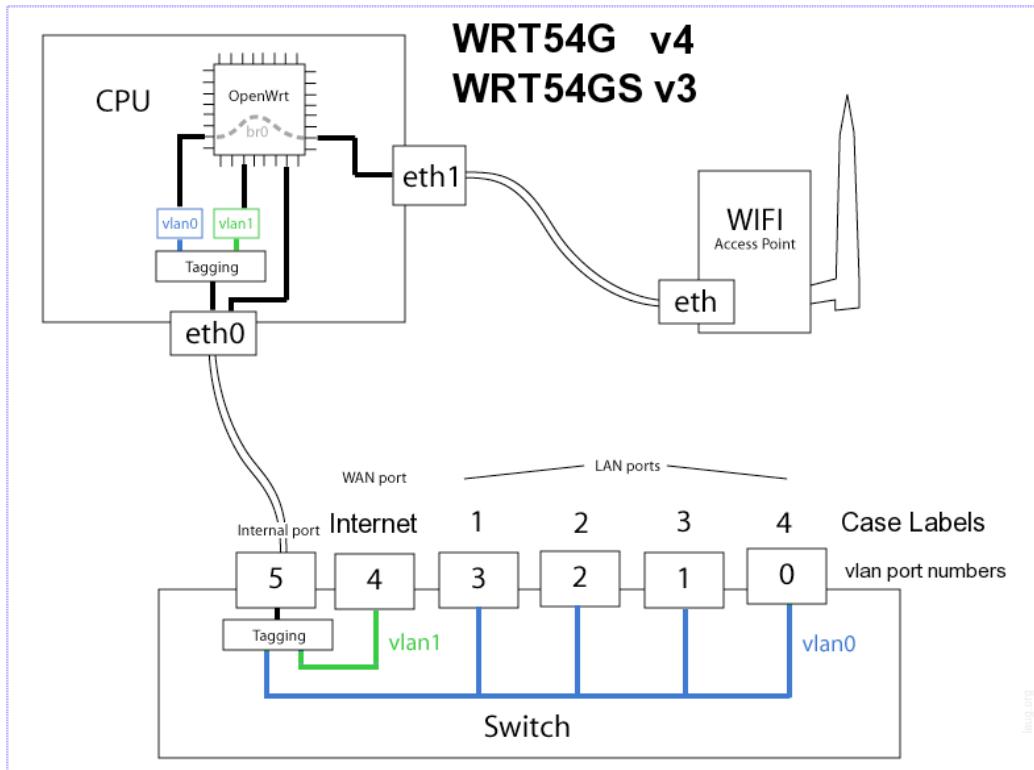
OpenWRT con hardware Linksys

OpenWRT è una distribuzione linux sotto forma di firmware, per Wireless Routers e Access Points. In altre parole è una distribuzione linux per sistemi embedded. La mentalità con cui è sviluppato è di tipo modulare, ovvero invece di provvedere a tutte le possibili funzionalità in un solo firmware, OpenWRT sviluppa un firmware minimale con supporto per pacchetti add-on. Questo permette di preparare firmwares personalizzati con le caratteristiche volute, aggiungendo o rimuovendo varie parti del sistema operativo. Inoltre gli sviluppatori possono concentrarsi sul lavoro di un singolo pacchetto, invece di dover testare un intero nuovo firmware per inserire una nuova funzionalità. Il successo della distribuzione OpenWRT è stato tale che, a partire da essa, ne sono nate molte altre tra cui una delle più importanti è Freifunk. È possibile fare un'analogia con Debian Linux, distribuzione dalla quale sono derivate molte altre.

Il primo testbed sperimentale che abbiamo costruito consisteva in 2 AP Linksys WRT54GS con firmware OpenWRT RC4 “White Russian”.

Il router Linksys WRT54GS V4 (dove V4 rappresenta la versione dell'hardware) ha 3 interfacce hardware differenti, 2 wired (`vlan0` e `vlan1`) ed una wireless (`eth1`).

Il router utilizza lo standard 802.1Q in hardware per gestire le sue interfacce wired, questo fa sì che 2 colori di VLAN (`vlan0` e `vlan1`) sono riservati e non possono essere utilizzati. In figura mostriamo l'architettura interna del router.



Installato il firmware sul router quello che troviamo è un familiare ambiente linux con kernel 2.4, lo stesso a cui siamo abituati su una macchina linux su architettura x86, e la console si ottiene tramite una sessione Secure Shell (SSH).

Abbiamo usato l'immagine con file system SquashFS, ovvero la memoria flash del router è divisa in 2 parti, una scrivibile come un normale file system, ed una in sola lettura.

In questo modo in caso di problemi è possibile attivare all'accensione il failsafe mode, una modalità di ripristino che rende possibile recuperare il router in caso di errori (ad esempio regole di firewall sbagliate che non consentono più di collegarsi al router).

Il nostro AP è adesso una mini macchina linux con 2MB di Flash e 8MB di RAM.

Per scrivere un programma in C e farlo girare sul WRT54G abbiamo bisogno di compilare per l'architettura mipsel, in questo il router monta un processore mipsel a 200Mhz

La Linksys nel suo repository GPL mette a disposizione il sorgente ed il firmware originale del router ed anche il compilatore.

Prova del modulo distributore di YaQoSa

I requisiti per usare YaQoSa sono:

1. Ambiente Linux
2. Kernel con supporto IEEE 802.1d (Linux Bridge)
3. Hardware & Drivers WiFi che supportano la modalità WDS
4. Kernel con supporto IEEE 802.1Q (VLAN)
5. Kernel con supporto Universal TUN/TAP driver

L’ambiente Linux da noi scelto inizialmente è stato OpenWRT, su hardware Linksys WRT54GS.

Per prima cosa abbiamo testato la compatibilità del modulo distributore di YaQoSa con la piattaforma OpenWRT, ovvero abbiamo verificato i 5 punti appena descritti.

Il supporto per l’802.1d (Linux Bridge) è nativo in OpenWRT, quindi non abbiamo avuto nessun problema da questo punto di vista.

Il router ha una scheda WiFi con chipset Broadcom, il driver incluso nel firmware è quindi il binario `wl.o` di cui non sono disponibili i sorgenti, ma nella sua versione per architettura mipsel supporta la modalità WDS.

In OpenWRT, quando l’AP viene configurato per parlare con un altro AP per fare Wireless Distribution System, il kernel tira su un’interfaccia del tipo `wds0.4915x` dove “x” è un numero che inizia da 3 e cresce in modo progressivo man mano che tiriamo su le altre interfacce.

Il primo link WDS avrà quindi il nome `wds0.49153`.

In OpenWRT è anche disponibile il supporto del Kernel per VLAN IEEE 802.1Q, in quanto i moduli sono già caricati di default al boot del sistema. Inoltre è incluso nella distribuzione il binario del tool in user space che serve a configurare le interfacce VLAN: `vconfig`.

Ad esempio, per creare un’interfaccia colorata di colore 5, il comando sarà:

```
vconfig add wds0.49153 5
```

che come risultato creerà sulla nostra macchina una interfaccia dal nome `wds0.49153.5`

Il driver TUN/TAP non è incluso nella distribuzione di default, ma come in molte altre distribuzioni linux, OpenWRT ha un sistema di pacchetti chiamati ipkg. È stato quindi sufficiente installare il pacchetto `vtun_2.6-1_mipsel.ipk` per avere tale driver.

Come già spiegato in precedenza YaQoSa è un software in user space che utilizza delle interfacce di tipo `tap` che sono sotto il controllo di altrettanti bridges.

Scripts

Sono stati inseriti in OpenWRT gli scripts per configurare l'AP all'accensione, ovvero per creare le opportune interfacce `tap`, i bridges, e taggare le interfacce automaticamente all'avvio del sistema, in modo da permettere il lancio del demone YaQoSa al boot dell'AP.

Il primo script si trova nella cartella `/etc/init.d/` e si chiama `S80distrib`, crea i bridges e le interfacce `tap`, le configura e lancia YaQoSa. Inoltre questo script prende come argomenti start, stop e restart, per lanciare, fermare o riavviare il demone.

```
#!/bin/sh

case "$1" in
start)
/usr/sbin/brctl addbr br5
/usr/sbin/brctl addbr br6

ifconfig br5 promisc up
ifconfig br6 promisc up

/usr/bin/yaqosa >> /tmp/distributore.log &

ifconfig tap0 promisc,
/usr/sbin/brctl addif br0 tap0
```

```

ifconfig tap0 up
/usr/sbin/brctl stp br0 off

ifconfig tap1 promisc
/usr/sbin/brctl addif br5 tap1
ifconfig tap1 up
/usr/sbin/brctl stp br5 on

ifconfig tap2 promisc
/usr/sbin/brctl addif br6 tap2
ifconfig tap2 up
/usr/sbin/brctl stp br6 on

;;
stop)
killall yaqosa
;;
restart)
$0 stop
$0 start
;;
*)
echo "Usage: $0 {start|stop|restart}"
exit 1
;;
esac

```

Un altro meccanismo da automatizzare era la creazione e l'inserimento delle interfacce WDS taggate 802.1Q dentro i bridges. Abbiamo quindi modificato lo script /etc/hotplug.d/net/01-wds

Questo script è il più interessante in quanto viene gestito da hotplug. Ogni volta che viene creata una interfaccia wds (anche in un momento futuro rispetto all'accensione) il demone di hotplug di preoccupa di creare le VLAN per il link WDS e di infilarle nei bridges opportuni. Questo, come vedremo più avanti, è

importante per l'autoconfigurazione del sistema nel momento in cui introduciamo nella mesh un nuovo AP.

```
[ "${INTERFACE%%[0-9]*}" = "wds" ] && {
    ifconfig $INTERFACE 0.0.0.0 up
if [ "${INTERFACE}" = "wds0.49153" ] || [ "${INTERFACE}" =
"wds0.49154" ]; then
    vconfig set_name_type DEV_PLUS_VID_NO_PAD
    vconfig add $INTERFACE 5
    vconfig add $INTERFACE 6
    /usr/sbin/brctl addif br5 $INTERFACE.5
    /usr/sbin/brctl addif br6 $INTERFACE.6
    ifconfig $INTERFACE.5 promisc
    ifconfig $INTERFACE.6 promisc

fi
}
```

Testbed in architettura mipsel

Il testbed sperimentale che abbiamo allestito è stato il seguente:

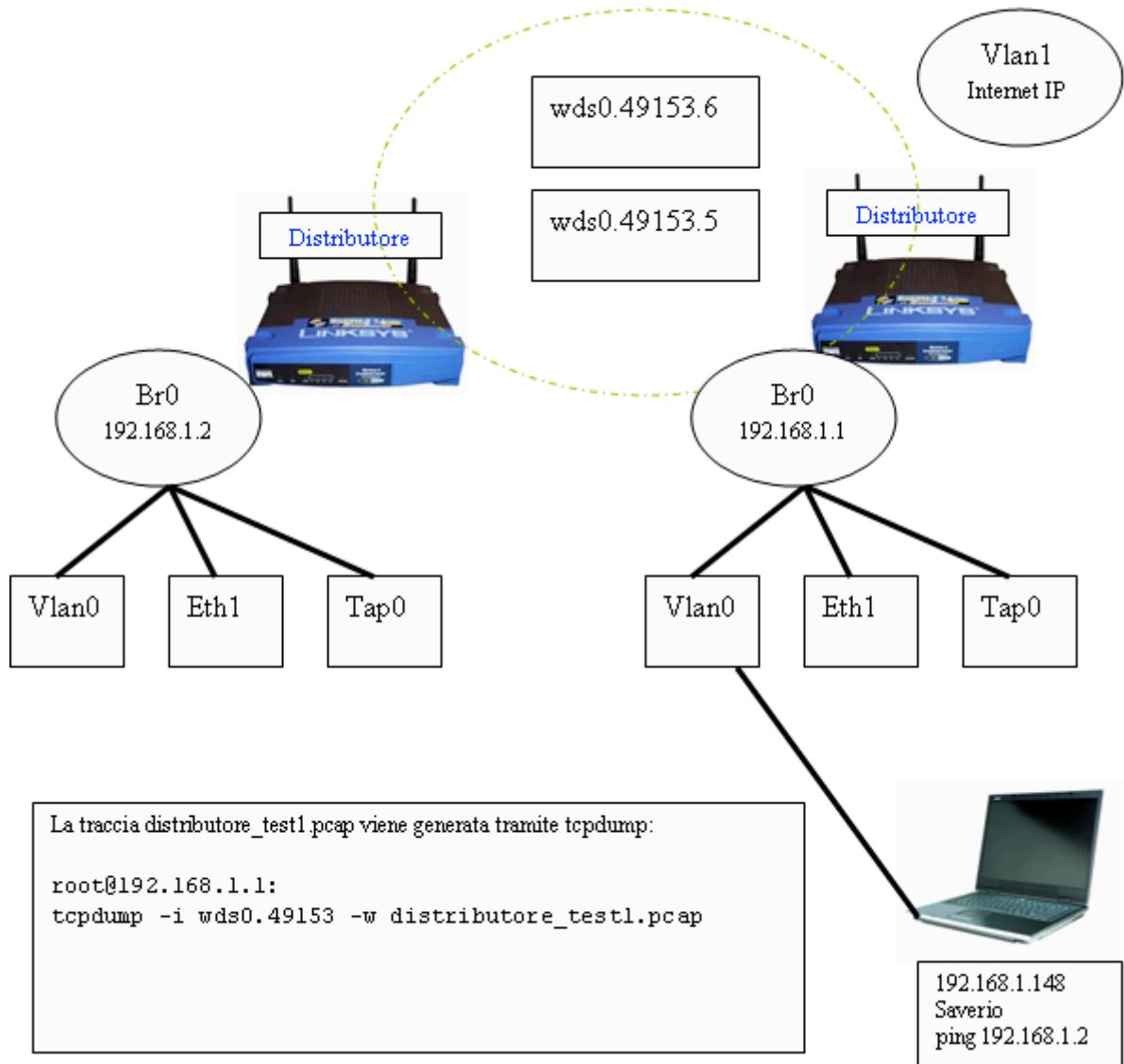


Figura 20 - Testbed sperimentale con Linksys WRT54G

CAPITOLO 4 – TESTBED SPERIMENTALE

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.2	192.168.1.148	SSH	Encrypted response packet Len=52
2	0.003385	192.168.1.2	192.168.1.148	SSH	Encrypted response packet Len=52
3	0.006578	192.168.1.2	192.168.1.148	SSH	Encrypted response packet Len=52
4	0.030188	192.168.1.148	192.168.1.2	TCP	2198 > 22 [ACK] Seq=0 Ack=104 win=17124 Len=0
5	0.031010	192.168.1.148	192.168.1.2	SSH	Encrypted request packet Len=52
6	0.031368	192.168.1.148	192.168.1.2	SSH	Encrypted request packet Len=52
7	0.031857	192.168.1.148	192.168.1.2	SSH	Encrypted request packet Len=52
8	0.032062	192.168.1.2	192.168.1.148	SSH	Encrypted response packet Len=432
9	0.036063	192.168.1.2	192.168.1.148	TCP	22 > 2198 [ACK] Seq=588 Ack=156 win=8576 Len=0
10	0.043611	192.168.1.148	192.168.1.2	SSH	Encrypted request packet Len=52
11	0.044294	192.168.1.148	192.168.1.2	SSH	Encrypted request packet Len=52
12	0.044793	192.168.1.148	192.168.1.2	SSH	Encrypted request packet Len=52
13	0.045175	192.168.1.148	192.168.1.2	SSH	Encrypted request packet Len=52
14	0.045707	192.168.1.148	192.168.1.2	SSH	Encrypted request packet Len=52
15	0.046698	192.168.1.2	192.168.1.148	TCP	22 > 2198 [ACK] seq=588 Ack=208 win=8576 Len=0
16	0.051628	192.168.1.2	192.168.1.148	TCP	22 > 2198 [ACK] seq=588 Ack=416 win=8576 Len=0
17	0.053391	192.168.1.148	192.168.1.2	SSH	Encrypted request packet Len=52
18	0.053875	192.168.1.148	192.168.1.2	SSH	Encrypted request packet Len=52
19	0.053912	192.168.1.148	192.168.1.2	SSH	Encrypted request packet Len=52
20	0.055802	192.168.1.2	192.168.1.148	TCP	22 > 2198 [ACK] seq=588 Ack=468 win=8576 Len=0 SLE=520 SRE=572
21	0.056293	192.168.1.2	192.168.1.148	TCP	22 > 2198 [ACK] seq=588 Ack=572 win=8576 Len=0
22	0.918282	192.168.1.148	192.168.1.2	ICMP	Echo (ping) request
23	0.919460	192.168.1.148	192.168.1.2	ICMP	Echo (ping) reply
24	1.469160	Cisco-L1_a5:2b:98		Spanning-tree-(for STP)	Conf. Root = 32768/00:14:bf:a5:2b:98 Cost = 0 Port = 0x8002
25	1.488216	Cisco-L1_a5:2b:98		Spanning-tree-(for STP)	Conf. Root = 32768/00:14:bf:a5:2b:98 Cost = 0 Port = 0x8002
26	1.919805	192.168.1.148	192.168.1.2	ICMP	Echo (ping) request
27	1.920971	192.168.1.2	192.168.1.148	ICMP	Echo (ping) reply
28	2.921013	192.168.1.148	192.168.1.2	ICMP	Echo (ping) request
29	2.922183	192.168.1.2	192.168.1.148	ICMP	Echo (ping) reply
30	3.458264	Cisco-L1_a5:2b:98		Spanning-tree-(for STP)	Conf. Root = 32768/00:14:bf:a5:2b:98 Cost = 0 Port = 0x8002
31	3.487926	Cisco-L1_a5:2b:98		Spanning-tree-(for STP)	Conf. Root = 32768/00:14:bf:a5:2b:98 Cost = 0 Port = 0x8002
32	4.925302	192.168.1.148	192.168.1.2	ICMP	Echo (ping) request
33	3.926468	192.168.1.2	192.168.1.148	ICMP	Echo (ping) reply
34	4.924299	192.168.1.148	192.168.1.2	ICMP	Echo (ping) request
35	4.925469	192.168.1.2	192.168.1.148	ICMP	Echo (ping) reply
36	5.462473	Cisco-L1_a5:2b:98		Spanning-tree-(for STP)	Conf. Root = 32768/00:14:bf:a5:2b:98 Cost = 0 Port = 0x8002
37	5.487980	Cisco-L1_a5:2b:98		Spanning-tree-(for STP)	Conf. Root = 32768/00:14:bf:a5:2b:98 Cost = 0 Port = 0x8002
38	5.926682	192.168.1.148	192.168.1.2	ICMP	Echo (ping) request
39	5.927851	192.168.1.2	192.168.1.148	ICMP	Echo (ping) reply
40	6.926182	192.168.1.148	192.168.1.2	ICMP	Echo (ping) request
41	6.927335	192.168.1.2	192.168.1.148	ICMP	Echo (ping) reply
42	7.457976	Cisco-L1_a5:2b:98		Spanning-tree-(for STP)	Conf. Root = 32768/00:14:bf:a5:2b:98 Cost = 0 Port = 0x8002

Figura 21 - Traccia di ethereal sul testbed sperimentale con Linksys WRT54G

La logica di YaQoS era ancora molto elementare quando questo testbed è stato montato in quanto il codice del decisore ancora non era stato scritto.

Tutto quello che arrivava da tap1 e tap2 veniva mandato a tap0. Su 10 frames che arrivano a tap0 5 venivano mandati a tap1 e 5 a tap0.

Questo portava ad avere nel DS una metà di traffico taggata di un colore e l'altra metà dell'altro.

RASTA, sviluppo e testbed

Il testbed ci ha permesso di capire la difficoltà di configurare l'intero sistema. Noi, che siamo gli sviluppatori, abbiamo impiegato diverse ore per avere un testbed funzionante di solo due apparati.

L'obiettivo che ci siamo posti è stato quello di creare un meccanismo di autoconfigurazione. Il primo passo a cui puntavamo erano quello di avere un instaurazione automatica del link WDS, sniffando i beacon degli AP.

Il driver wl.o mette a disposizione un binario in user space per configurare la scheda. Questo binario si chiama wl ed è installabile su OpenWRT come un ipkg opzionale. È il solo ipkg della distribuzione per il quale non è disponibile il codice sorgente. Studiando il driver wl.o abbiamo trovato una funzionalità non

CAPITOLO 4 – TESTBED SPERIMENTALE

documentata nel supporto ufficiale della broadcom ma ben conosciuta su internet, la possibilità di sfruttare l’interfaccia prism0 abilitandola tramite il comando wl monitor 1:

```
login as: root
root@192.168.170.2's password:

BusyBox v1.00 (2005.11.23-21:46+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

| ____ | .-----.-.-----.| ____ | .-----.| ____ | | | | | | | | | | | | |
| - || - | - | | | | | | | | | | | | | |
| ____ | | | | | | | | | | | | | | | | |
|_ | W I R E L E S S F R E E D O M

WHITE RUSSIAN (RC4) -----
* 2 oz Vodka Mix the Vodka and Kahlua together
* 1 oz Kahlua over ice, then float the cream or
* 1/2oz cream milk on the top.

-----
NINUX.ORG RELOADED !!!!!
root@ZPNINUX:~# wl monitor 1
root@ZPNINUX:~# ifconfig
eth0      Link encap:Ethernet HWaddr 00:13:10:83:CA:11
          UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
          RX packets:16047 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15850 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4288798 (4.0 MiB) TX bytes:1157098 (1.1 MiB)
          Interrupt:5 Base address:0x2000

eth1      Link encap:Ethernet HWaddr 00:13:10:83:CA:13
          inet addr:192.168.170.2 Bcast:192.168.170.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:276112 errors:0 dropped:0 overruns:0 frame:12992494
          TX packets:325687 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:23477230 (22.3 MiB) TX bytes:31755634 (30.2 MiB)
          Interrupt:4 Base address:0x1000

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

prism0    Link encap:UNSPEC HWaddr 00-13-10-83-CA-13-00-00-00-00-00-00-00-00-00-00
          UP BROADCAST MULTICAST MTU:0 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

vlan0     Link encap:Ethernet HWaddr 00:13:10:83:CA:11
          inet addr:10.0.1.14 Bcast:10.0.1.15 Mask:255.255.255.252
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:16047 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15850 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3999952 (3.8 MiB) TX bytes:1157098 (1.1 MiB)

root@ZPNINUX:~#
```

Dopo l'esecuzione del comando è presente una nuova interfaccia `prism0` che permette di sniffare il canale radio mentre l'AP è in esercizio.

Per avere un'idea di che tipo di informazioni fosse possibile ottenere dall'interfaccia `prism0` abbiamo lanciato il seguente comando nel testbed prima descritto:

```
tcpdump -i prism0 -s 0 -w pcapprism1.pcap
```

La documentazione sulle informazioni che è possibile ottenere nel prism header è già stata trattata nel capitolo 2.

Da queste riflessioni, fatte sulla base dell'esperienza acquisita durante testbed sperimentale, nasce lo sviluppo di RASTA.

Volevamo creare un *tool* che fosse utilizzabile con qualsiasi driver che forniva sulla scheda una interfaccia di monitor con prism header accessibile durante il funzionamento dell'AP.

Oltre al Linksys WRT54G esistono altri AP compatibili con OpenWRT, ma con scheda WiFi Atheros spesso montata su un'interfaccia Mini-PCI. La versione corrente del driver per Atheros madwifi-ng offre la possibilità di avere a disposizione un interfaccia di monitor con prism header. Inoltre questo driver è open source ed è quindi possibile compilarlo per ogni architettura.

Abbiamo iniziato a sviluppare il codice di RASTA con l'obiettivo di renderlo compatibile a più drivers e più schede WiFi in modo da non renderlo un codice esclusivo all'utilizzo con un determinato hardware.

La maggior parte delle difficoltà in questa fase sono state riscontrate nel driver madwifi, che si è rivelato molto instabile. Spesso durante le nostre prove il driver crashava quando gli venivano passati dei comandi di configurazione. I comandi al driver vengono passati con un apposito tool in userspace che viene fornito con l'installazione del driver: `wlanconfig`. Il driver fa quindi uso di questo suo tool e non usa le linux-wireless-extesions (`iwconfig`). Da questo punto di vista il driver della broadcom anche se chiuso, e disponibile solo per architettura mipsel, è risultato decisamente migliore.

Inoltre, un altro problema riscontrato nel driver madwifi, è che sebbene compatibile al 100% con RASTA, non è compatibile per ora con YaQoS, in quanto non c'è supporto per le VLAN 802.1Q, ovvero il driver non tagga i pacchetti come dovrebbe invece fare secondo lo standard.

CAPITOLO 4 – TESTBED SPERIMENTALE

Il driver wl.o si è invece comportato sempre in modo eccellente sui Linksys WRT54G. Un testbed sperimentale di 3 AP con OperWRT RC4 e RASTA ha provato la possibilità effettiva di autoconfigurare la rete.

Linuxino con hardware Soekris

L’esperienza maturata durante la prima fase di sperimentazione ci ha spinto a provare nuove piattaforme hardware e nuovi sistemi operativi.

Abbiamo abbandonato la archittettura mipsel e siamo passati alla piattaforma x86, cambiamento che ci dà notevoli vantaggi. Innanzitutto la piattaforma x86 è più comune e quindi è possibile trovare più facilmente supporto dalle comunità di sviluppo open source ma, cosa più importante, è possibile sviluppare e testare codice direttamente sul PC e poi passarlo sui sistemi embedded.

Lo svantaggio dei sistemi su architettura mipsel embedded è il loro hardware, come ad esempio il Linksys WRT54G, estremamente ridotto. 8Mb di RAM spesso non sono sufficienti e le operazioni di debugging possono diventare impossibili. Senza una piattaforma di debugging seria e professionale, a volte dedicata per l’hardware specifico, non si è in grado di comprendere un eventuale problema software incorso, spesso imputabile a particolarità dell’hardware stesso. Inoltre anche il sistema operativo OpenWRT non è una release stabile ma soltanto una release candidate (RC4), e quindi a volte presenta strani problemi di stabilità.

La nuova piattaforma hardware che abbiamo adottato sono le schede Soekris con processore x86 (amd geode sc1100) e slot mini-pci per montare le schede radio. Per questa esistono molte più distribuzioni linux ed è molto semplice modificare il sistema operativo in quanto il vendor fornisce il pieno supporto per la compatibilità con i sistemi unix.

Come già accennato nel primo capitolo, un’apposita distribuzione linux per questo tipo di macchine è stata messa a punto nei nostri laboratori. L’idea è stata quella di costruire qualcosa che una volta ottimizzato potesse essere portato su AP commerciali.

Il sistema operativo che abbiamo realizzato è quindi essenziale, usa delle librerie che si chiamano uClibc.

uClibc (aka µClibc/pronounciato yew-see-lib-see) è una libreria C per sviluppare sistemi embedded Linux. È molto più piccolo della GNU C Library, ma quasi tutte le applicazioni supportate da glibc funzionano perfettamente anche con uClibc. Fare il porting delle applicazioni da glibc a uClibc di solito è una operazione che consiste

solo nel ricompilare il codice sorgente. Le uClibc supportano anche le librerie condivise ed il threading. La versione attuale gira su standard Linux e su sistemi MMU-less (also known as μ Clinux) con supporto per le architetture di processori alpha, ARM, cris, i386, i960, h8300, m68k, mips/mipsel, PowerPC, SH, SPARC, e v850.

Il progetto uClibc fornisce un ambiente linux pronto per sviluppare su queste librerie e, grazie a questo ambiente, è possibile compilare applicazioni con estrema semplicità. È sufficiente copiare in una cartella tale ambiente e successivamente usando il comando `chroot /nomecartella` è possibile compilare nel nuovo ambiente.

Lavorando in x86 molti bugs e problemi che avevamo sono spariti, erano dovuti probabilmente all’ambiente linux di OpenWRT che non è ancora stabile.

Testbed su Soekris net4801

Il nostro primo testbed è stato costruito con due soekris net4801. Il codice era già pronto prima dell'arrivo di tutto l'hardware e così abbiamo cominciato a montare un testbed completamente wired per iniziare a debuggare il codice. Le net4801 che abbiamo utilizzato non erano dotate di schede radio, ma avevano due distinte interfacce ethernet sufficienti per testare se il codice era funzionante.

Per simulare il “canale radio” ci siamo serviti di un vecchio HUB ethernet, in modo da non dover operare un attacco man in the middle per sniffare nel DS.

Siamo rimasti soddisfatti in quanto il sistema si comportava proprio nel modo che volevamo, e tutti i problemi legati alla non stabilità di OpenWRT (pacchetti persi nel nulla, crash del sistema...) sono scomparsi.

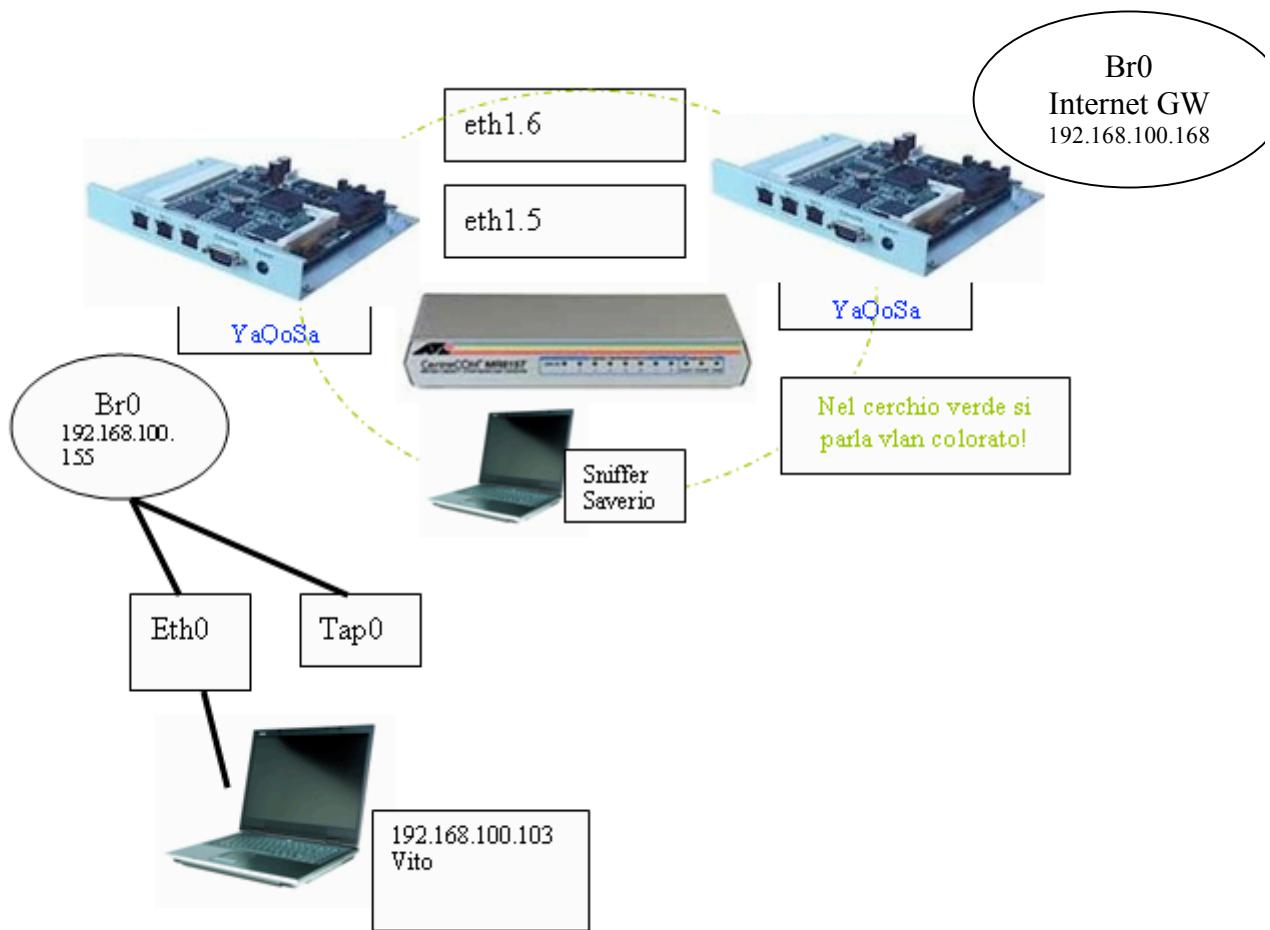


Figura 22 - Primo Testbed su architettura x86

Preparazione Testbed su Soekris net4826

Per il secondo testbed sperimentale abbiamo usato le soekris net4826, che dispongono di due slot miniPCI ed hanno quindi la possibilità di utilizzare due schede wireless contemporaneamente. Questo è utile se si vuole separare il canale radio delle BSS e quello del DS, che potrebbero addirittura essere implementati con due differenti estensioni dello standard. Ad esempio per il DS, dove si suppone ci sia una maggior mole di traffico ed i link radio sono più critici, è possibile usare 802.11a mentre per le BSS è sufficiente usare 802.11b.

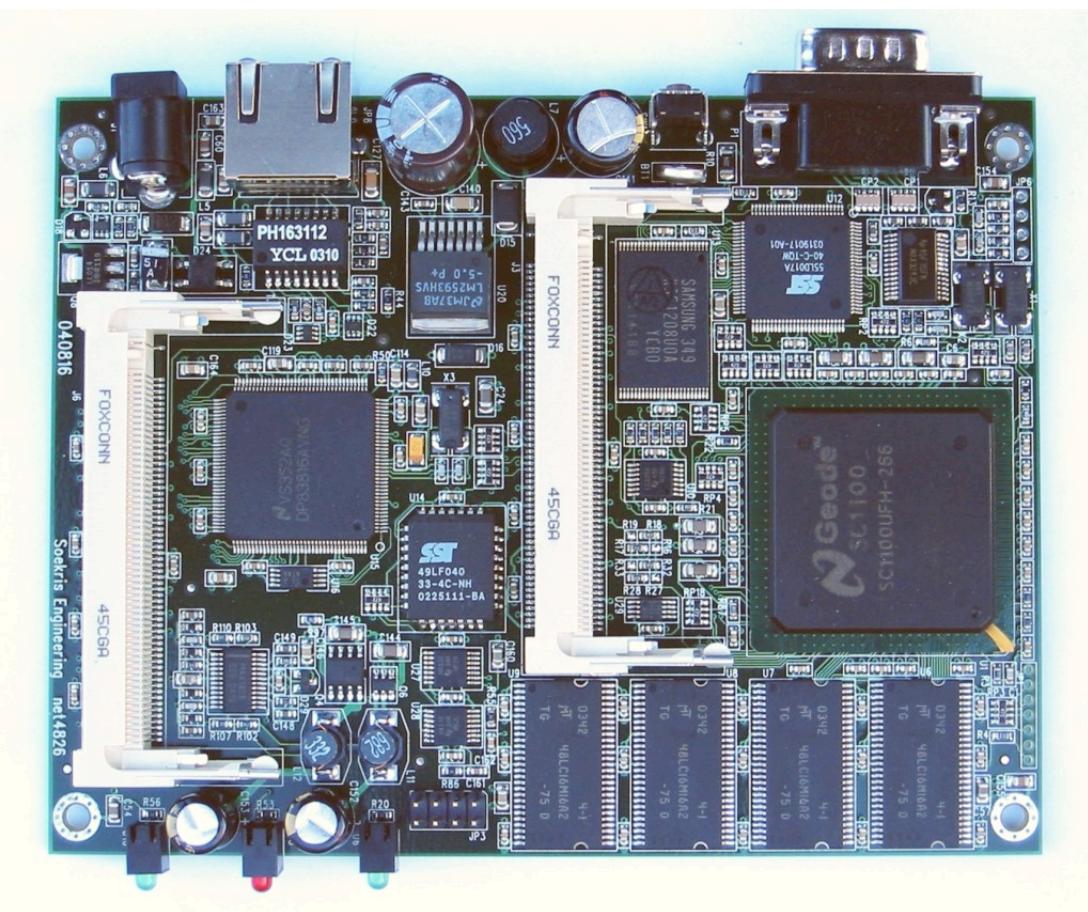


Figura 23 - Soekris net4826

La Soekris net4826, a differenza della net4801, ha la memoria Compact Flash saldata sulla scheda madre, è stato quindi necessario utilizzare il protocollo PXE per installare il sistema operativo. PXE è l'abbreviazione per Pre-Boot Execution Environment, si pronuncia pixie, ed è uno dei componenti delle specifiche di Intel WfM. Questo permette alle workstation di fare il boot da un server su una rete.

La procedura consiste nell'avere un server dhcp ed un server tftp che lavorano insieme. Il server dhcp passa delle opzioni particolari alla soekris che da tftp scarica una particolare immagine che installerà linuxino nella sua compact flash.

Nel server dhcp va configurata una sezione speciale per ogni macchina che si vuole configurare, dove vanno indicati i parametri per l'installazione del suo sistema operativo. Nel nostro server linux del laboratorio abbiamo configurato il server (editando il file `/etc/dhcpd.conf`).

```
host soekris1 {
    hardware ethernet 00:00:24:C5:C9:E4;
    fixed-address 192.168.100.80;
    option subnet-mask 255.255.255.0;
    option broadcast-address 192.168.100.255;
    option routers 192.168.100.1;
    next-server 192.168.100.1;
    filename "pxelinux.0";
}

host soekris2 {
    hardware ethernet 00:00:24:C5:CA:58;
    fixed-address 192.168.100.81;
    option subnet-mask 255.255.255.0;
    option broadcast-address 192.168.100.255;
    option routers 192.168.100.1;
    next-server 192.168.100.1;
    filename "pxelinux.0";
}
```

A questo punto la Soekris farà una richiesta tftp al *next-server* per il file `pxelinux.0`. PxeLinux è una derivazione del progetto SYSLINUX, per fare il boot

di un sistema operativo linux via rete usando una memoria ROM conforme allo standard Intel PXE (Pre-Execution Environment).

Nella directory root del nostro tftpserver è stata già preparata una cartella che si chiama pxelinux.cfg, in questa si trova il file con le istruzioni per scaricare il root file system di linuxino. Il nome di questo file è il MAC address della scheda soekris:

```
bash-3.00$ ls -l -h /tftpboot/pxelinux.cfg/
total 20K
-rwxrwxrwx 1 tftpd tftpd 213 Jun 20 17:25 01-00-00-24-c5-c9-d8*
-rwxrwxrwx 1 tftpd tftpd 213 Jun 13 10:35 01-00-00-24-c5-c9-e4*
-rwxrwxrwx 1 tftpd tftpd 213 Jun 20 17:59 01-00-00-24-c5-ca-54*
-rwxrwxrwx 1 tftpd tftpd 213 Jun 16 16:36 01-00-00-24-c5-ca-58*
-rwxrwxrwx 1 tftpd tftpd 213 Jun 20 18:00 01-00-00-24-c5-ca-6c*
bash-3.00$
```

Il contenuto di questi files (uno per ogni differente scheda da configurare) è il seguente:

```
SERIAL 0 9600
DEFAULT ramdisk
LABEL ramdisk
kernel linuxino
APPEND initrd=linuxino.img ramdisk_size=8192 root=/dev/ram init=/linuxrc
console=ttyS0,9600 rw
```

L'ultimo passo consiste nel download da parte della soekris dell'immagine linuxino.img, ovvero del root file system del sistema operativo linux che installerà la nostra distribuzione. Questa immagine verrà caricata in memoria e automaticamente scaricherà dalla rete la partizione di root da installare nella flash. In pratica svolgerà i seguenti passi in modo automatico:

- Cancellerà la tabella delle partizioni con fdisk la compat flash mettendo una partizione linux
- Formatterà con ext2 la partizione appena creata
- Scaricherà dalla rete la distribuzione Linuxino (compressa in tgz linuxino.tgz)
- Compatterà la distribuzione nella partizione

- Installerà il loader (lilo) per far partire linuxino automaticamente
- Si riavvierà automaticamente facendo partire il sistema operativo appena installato

Una volta preparate le Soekris con schede 802.11a/b/g Atheros abbiamo fatto una spiacevole scoperta. Il driver di MadWiFi non supporta il tag 802.1Q. Sebbene il protocollo di VLAN sia nello standard e dovrebbe essere implementato su ogni NIC ethernet, il driver di MadWiFi, essendo ancora in versione beta non lo implementa. Contattati gli sviluppatori ci hanno informato che il completo supporto per l'802.1Q sarà disponibile con la prima release stable del driver. È stato quindi possibile solo provare RASTA, che ha però dato ottimi risultati autoconfigurando la rete e pesando correttamente i rami dello spanning tree.

Conclusioni

Le reti wireless sono una importante forma di rete di accesso per molte attività, soprattutto per le imprese. Il mercato dei dispositivi wireless è in crescita. Reti sono presenti negli aeroporti, nelle università, nei parchi pubblici delle grandi città.

L'obiettivo di questo lavoro di tesi è stato presentare un'implementazione di rete mesh conforme allo standard IEEE 802.11, scalabile ad elevato numero di nodi, che offrisse qualità di servizio per diverse categorie di traffico e realizzata con software open source.

Sfruttando il canale radio anche per il distribution system, non solo la tecnologia wireless eviterà all'utente di essere vincolato al cavo, ma anche l'installazione di una rete wireless sarà più facile ed economica per gli installatori. La diffusione di questa tecnologia diventerà ancora più massiccia di quanto sia stata fin ora.

Il risultato raggiunto ci vede molto vicini all'obiettivo che ci eravamo posti. Il software sviluppato è ancora in versione beta e necessita del passaggio per uno stadio di ottimizzazione, ma nel complesso è funzionante e permette l'instaurazione automatica e dinamica del *link* radio, caratteristica che la rende scalabile ad un elevato numero di nodi.

Inoltre, anche se ancora non è chiaro quale sia la metrica passiva migliore per caratterizzare il canale radio, abbiamo già il codice che ci permette di pesare uno spanning tree con una metrica diversa da quella convenzionale, e soprattutto con una metrica radio aware.

Anche per la qualità di servizio, sebbene con dei problemi dovuti alla implementazione parziale dei drivers dei dispositivi wireless per quello che concerne lo standard 802.1Q, è stato sviluppato un codice che è portabile facilmente su molti dispositivi diversi e che permette di gestire il problema della qualità di servizio direttamente a livello MAC.

La scelta di implementare restando fedeli allo standard IEEE 802.11 ci ha dato notevoli vantaggi, dalla disponibilità di documentazione e codice GPL in rete, alla garanzia che il nostro lavoro non andasse in conflitto con i protocolli nei layer superiori. Inoltre la documentazione ed il codice prodotti durante questo lavoro sono un ottima risorsa per tutti coloro che producono software GPL per tecnologie

CONCLUSIONI

wireless. A mio parere infatti l'aspetto più importante è il contributo dato alla comunità open source nello sviluppo di software libero.

In conclusione sono state gettate le basi per una buona implementazione open source che mira a risolvere i problemi di interoperabilità, definendo un'architettura e protocolli che supportino la consegna di pacchetti broadcast/multicast e unicast, usando delle metriche che tengono conto della qualità del canale radio, tutto su una topologia multi-hop autoconfigurante.

Bibliografia

- IEEE Std. 802.11; “Wireless LAN Media Access Control (MAC) and Physical Layer (PHY) Specification”; R2003
- Gautam Kulkarni, Alok Nandan, Mario Gerla e Mani Srivastava; “A Radio Aware Routing Protocol for Wireless Mesh Networks”; UCLA Electrical Engineering, UCLA Computer Science Los Angeles, CA
- Richard Draves, Jitendra Padhye e Brian Zill; “Comparison of Routing Metrics for Static Multi-Hop Wireless Networks”; Microsoft Research
- OpenWRT - <http://openwrt.org>
- Multiband Atheros Driver for WiFi - <http://madwifi.org>
- Librerie uClibc - <http://www.uclibc.org/>
- Compilatore mipsel GPL Linksys -
ftp://ftp.linksys.com/opensourcecode/wrt54gl/4.30.0-EU/WRT54GL_v4.30.0_ESI.tgz
- Embedded Linux/Microcontroller Project - <http://www.uclinux.org/>
- Wireless Community di Roma - <http://www.ninux.org>
- Progetto Mesh della Wireless Community di Roma -
<http://tuscolomesh.ninux.org>
- Progetto TWELVE - <http://twelve.unitn.it/>

BIBLIOGRAFIA

- Università degli studi di Roma Tor Vergata – <http://www.uniroma2.it>
- Tropos Networks - <http://www.tropos.com>
- Firetide Instant Mesh Networks – <http://www.firetide.com>
- Ministero delle Comunicazioni - <http://www.comunicazioni.it>
- Ethereal home page - <http://www.ethereal.com>
- Kismet home page - <http://www.kismetwireless.net>
- Wiviz home page - <http://devices.natetrue.com/wiviz/>
- Broadcom 4301 Linux Driver Project -
<http://sourceforge.net/projects/linux-bcom4301>
- Broadcom replacement firmware -
<http://sourceforge.net/projects/newbroadcom>
- IEEE Std. 802.1Q-2003, Virtual Bridged Local Area Networks; ISBN 0-7381-3662-X.
- IEEE 802.1 P,Q - QoS on the MAC level -
<http://www.tml.tkk.fi/Opinnot/Tik-110.551/1999/papers/08IEEE802.1QosInMAC/qos.html>
- Freifunk Firmware - <http://freifunk.net/>
- Freifunk olsr experiments – <http://olsrexperiment.de>
- Sveasoft Alchemy e Talisman Firmwares - <http://www.sveasoft.com/>
- AODV - <http://moment.cs.ucsb.edu/AODV/aodv.html>

BIBLIOGRAFIA

- OLSR - <http://olsr.org>
- Linuxino - (Vito Ammirata) vito.ammirata@tin.it
- Soekris Engineering – <http://www.soekris.com>
- Gruppo reti dell’Università degli studi di Roma Tor Vergata -
<http://netgroup.uniroma2.it/>