[16] Jouni Malinen, *Host AP driver for Intersil Prism2/2.5/3 and WPA Supplicant*, 2001-2004, `http://hostap.epitest.fi`, *jkmaline@cc.hut.fi*.

[17] Stefano Lucetti, *STATS homepage*, TLC NetGroup, Università di Pisa, February 2004, `http://netgroup-serv.iet.unipi.it/stats/`.

[18] Brian Adamson, Hal Greenwald, *MGEN User's and Reference Guide*, 2004, `http://mgen.pf.itd.nrl.navy.mil/mgen.html`, *adamson@itd.nrl.navy.mil*, *hgreenwald@itd.nrl.navy.mil*.

[6] Bert Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, P. Larroy, *Linux Advanced Routing & Traffic Control HOWTO*, 2000-2003, `http://lartc.org`, *HOWTO@ds9a.nl*.

[7] David Hinds, *Linux PCMCIA Information Page*, 2000-2003, `http://pcmcia-cs.sourceforge.net`, *dahinds@users.sourceforge.net*.

[8] *PLUTO project - Italian Linux Documentation Project*, 1994-2003, `http://it.tldp.org/Guida-ILDP/howto.html`.

[9] *Linux Documentation Project HOWTO Index*, 1994-2004, `http://www.linux.org/docs/ldp/howto/HOWTO-INDEX/`.

[10] Daniele Giacomini, *Appunti di informatica libera*, 2000-2004, `http://a2.pluto.it/a21.html`, *daniele@swlibero.org*.

[11] Daniele Giacomini, *Differenze tra i file*, 2004, `http://a2.swlibero.org/differenze_tra_i_file.html`, *daniele@swlibero.org*.

[12] Jean Tourrilhes, *Wireless LAN resources for Linux*, 1996-2003, `http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/`, *jt@hpl.hp.com*.

[13] Mark Lamb, Alexey Kuznetsov, *iproute2+tc notes*, 1999, `http://snafu.freedom.org/linux2.2/iproute-notes.html`.

[14] Martin Devera, *HTB Linux queuing discipline manual - user guide*, May 2002, `http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm`, *devik@cdi.cz*.

[15] Chuck Semeria, *Supporting Differentiated Service Classes: Queue Scheduling Disciplines*, Juniper Networks, December 2001, `http://www.juniper.net/solutions/literature/white_papers/200020.pdf`.

Indice

Bibliografia

[1] Lars Wischhof, John W. Lockwood, *Packet Scheduling for Link-Sharing and Quality of Service Support in Wireless Local Area Networks*, Technical Report WUCS-01-35, Applied Research Laboratory, Washington University in St. Louis, November 2001, `http://wsched.sourceforge.net`, *wischhof@ieee.org*.

[2] M. Heusse, F. Rousseau, G. Berger-Sabbatel, A. Duda, *Performance Anomaly of 802.11b*, Proc. of IEEE Infocom 2003, San Francisco, April 2003, `http://www.ieee-infocom.org/2003/papers/21_01.PDF`, *heusse@imag.fr*.

[3] Martin Devera, *Hierarchical token bucket theory*, May 2002, `http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm`, *devik@cdi.cz*.

[4] Stefano Lucetti, *Performance evaluation of SIP-based multimedia services over infrastructured IEEE 802.11b networks*, VICOM Project - Work Package 4 - Section 1.1.3, UdR CNIT-Pisa, 2003, *s.lucetti@iet.unipi.it*.

[5] M. G. Arranz, R. Agüero, L. Muñoz, P. Mähönen, *Behavior of UDP-Based Applications over IEEE 802.11 Wireless Networks*, Proceedings of the 12th IEEE International Symposium on Personal Indoor and Mobile Radio Communication, PIMRC 2001, San Diego, September 2001.

LLC	Logical Link Control
MAC	Medium Access Control
MGEN	Multi-GENerator
MS	Mobile Station
PC	Personal Computer
PCMCIA	Personal Computer Memory Card International Association
PCMCIA-CS	PCMCIA Card Services
PLCP	Physical Layer Convergence Protocol
QoS	Quality of Service
RTS	Request To Send
SIFS	Short InterFrame Space
SLA	Service Level Agreement
SNR	Signal-to-Noise Ratio
TBF	Token Bucket Filter
TC	Traffic Control
TWCM	Time-based Wireless Channel Monitor
TWHTB	Time-based Wireless Hierarchical Token Bucket
UDP	User Datagram Protocol
WChMon	Wireless Channel Monitor
WHTB	Wireless Hierarchical Token Bucket
WLAN	Wireless Local Area Network

Acronimi

ACK	ACKnowledgement
AP	Access Point
ARP	Address Resolution Protocol
ARQ	Automatic Repeat reQuest
BER	Bit Error Rate
CBQ	Class-Based Queuing
CTS	Clear To Send
CW	Contention Window
DCF	Distributed Coordination Function
DIFS	Distributed InterFrame Space
DRR	Deficit Round Robin
FCS	Frame Check Sequence
FER	Frame Error Rate
FIFO	First-In, First-Out
H-FSC	Hierarchical Fair Service Curve
HTB	Hierarchical Token Bucket
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISP	Internet Service Provider
LAN	Local Area Network

Introduzione

Con il crescente sviluppo delle Wireless Local Area Networks (WLAN) sono aumentate le possibilità di accedere ad Internet da terminali mobili. A causa del contemporaneo sviluppo di applicazioni con requisiti stringenti in termini di ritardo e capacità trasmissiva, in altre parole di **Qualità del Servizio** (QoS), si prospetta la necessità per gli Internet Service Provider (ISP) di suddividere le risorse della rete wireless in modo controllato e differenziato fra i clienti.

Tuttavia gli algoritmi di scheduling attuali, che siano *classless* (come la semplice coda **FIFO** o il **TBF**) o *gerarchici*, cioè dotati di classi di traffico strutturate ad albero (come il **CBQ** o l'**HTB**), non vedono le ritrasmissioni delle trame ed i meccanismi di adattamento del rate implementati ai livelli inferiori della pila protocollare per ridurre gli effetti del degrado della qualità del canale di trasmissione: dunque gli scheduler tradizionali **non tengono conto delle caratteristiche del canale wireless**. Ciò porta ad un'**anomalia nelle prestazioni** della Wireless LAN, perché se un Access Point deve trasmettere dati a due destinazioni mobili, una vicina ed una lontana, la cattiva qualità del canale verso la destinazione lontana determina una riduzione della capacità trasmissiva verso quella che si trova a minore distanza, benché la qualità del canale verso di essa sia costante e buona. La causa di questo fenomeno è che la stazione che si trova in posizione di cattiva ricezione consuma più risorse trasmissive di quante gliene spetterebbero.

Per evitare un tale scenario, lo scheduler deve poter tenere conto della

quantità di risorse di rete richiesta per poter fornire uno specifico servizio all'utente. Questo permetterebbe ad un ISP di stipulare un *Service Level Agreement* (SLA) basato non solo sui dati effettivamente ricevuti dall'utente ma specificando anche le condizioni da soddisfare al fine di ottenere la banda concordata. Ad esempio, ad un utente vengono garantiti 64 kbit/s finché il rapporto segnale-rumore (SNR) si mantiene sopra un dato valore: al di sotto di tale soglia, lo scheduler dell'Access Point ridurrà la frequenza di trasmissione dei pacchetti verso l'utente al fine di limitarne il consumo di risorse della rete. In ogni caso, questa limitazione risulta opportuna solo in una situazione di scarsità di banda.

Oggetto di questo lavoro di tesi è l'analisi dell'anomalia ed il suo superamento tramite due diversi approcci.

- Nel primo capitolo si definisce che cosa si intende per **anomalia** della Wireless LAN 802.11b e si evidenziano gli inconvenienti che questa anomalia comporta ai fini della Qualità del Servizio.

- Il secondo capitolo descrive la struttura di uno scheduler che tenga conto della qualità del canale: a questo scopo viene introdotto il *Wireless Channel Monitor*, un importante strumento per la stima della capacità trasmissiva ottenibile verso una determinata destinazione a partire dalla conoscenza del **rapporto segnale-rumore**. Il capitolo 2 spiega inoltre come sia possibile modificare lo scheduler HTB, già implementato all'interno del kernel del sistema operativo Linux, in modo che consenta alle classi di traffico con buona qualità del canale di ricevere la quantità di dati cui hanno diritto, indipendentemente dalla presenza di altre classi associate a destinazioni con canali di qualità peggiore. I risultati sperimentali per lo scheduler **WHTB**, fondato su questo approccio, sono riportati nel capitolo 3.

- Infine, nel quarto capitolo viene presentata l'implementazione di un'altra tecnica di stima, più complicata dal punto di vista implementativo


```
                    for (i=0; i < 6; i++) capa[i] = '0';
                    i = 0;
                }
                else {
                    capa[5] = capa[4];
                    capa[4] = capa[3];
                    capa[3] = capa[2];
                    capa[2] = capa[1];
                    capa[1] = capa[0];
                    capa[0] = c;
                }
            }
            for (i = 5; i >= 0; i--) putchar (capa[i]);
            putchar ('\n');
    }
}
```

```
                    while ((c = getchar()) != ' ') putchar (c);
                    putchar ('.');
                    for (i=0; i < 6; i++) putchar ('0');
                    putchar(c);
                    /* "edison kernel: channel 00 02 2D 20 6E 8E: level" */
                    for (i=16; i < 64; i++) getchar();
                    while ((c = getchar()) != ' ') putchar (c);
                    while ((c = getchar()) != '\n');
                    putchar (c);
        }
}
```

Programma che elimina le parole dal file di debug, in modo da estrarre l'ultimo valore numerico della riga (in questo caso, il goodput stimato dal TWCM): `eliminaparole4.c`

```
#include <stdio.h>

main() {
        char c, capa[6];
        int i;
        while ((int)(c = getchar()) != -1)
        {
                    for (i = 1; i < 7; i++) getchar();
                    while ((c = getchar()) != ' ') putchar (c);
                    putchar ('.');
                    for (i = 0; i < 6; i++) putchar ('0');
                    putchar(c);
                    /* registra numeri fino a '\n' */
                    for (i = 0; i < 6; i++) capa[i] = '0';
                    i = 0;
                    while ((c = getchar()) != '\n') {
                            if (c < '0' || c > '9') {
```


(in quanto comporta la modifica del driver di una scheda di rete) ma più accurata nella stima della capacità trasmissiva, basata sulla misura dell'**intervallo temporale** fra due pacchetti diversi consecutivi anziché sull'SNR. I risultati sperimentali per lo scheduler **TWHTB** e sullo stimatore **TWCM**, implementati in quest'ultima fase della tesi, sono riportati nel capitolo 5.

```
                                        default: exit(1);
                                }
                        }
                        numero = num * 0.5;
                        printf (" %02g\t", numero);
                        while ((c = getchar()) != 'x');
                        num = 0;
                        while ((c = getchar()) != '\n' && (int)c != -1) {
                                num *= 16;
                                switch (c) {
                                        case '0'...'9': num += c - '0';
                                                        break;
                                        case 'a'...'f': num += c - 'a' + 10;
                                                        break;
                                        default: exit(1);
                                }
                        }
                        numero = num * 0.000001;
                        printf ("%f\n", numero);
        }
}
```

Programma che elimina le parole dal file di debug, in modo da estrarre la qualità del canale secondo per secondo: `eliminaparole3.c`

```
#include <stdio.h>

main() {
        char c;
        int i;
        while ((int)(c = getchar()) != -1)
        {
                        for (i=1; i < 7; i++) getchar();
```

```
                printf ("%d0\t%d\n", i, tabella [i]);
        }
}
```

Programma per l'interpretazione di dati estratti dalla *Prism2 Monitoring Header*, in particolare per trasformare i valori esadecimali del `rate` e del `MAC time` in valori reali, più facilmente interpretabili: `shh2sff.c`

```
/* Questo programma converte righe del tipo:
 * STRINGA 0xESADECIMALE 0xESADECIMALE
 * in righe del tipo:
 * STRINGA FLOAT FLOAT
 * (che rappresentano: tipo di trama, rate nominale (Mbit/s), istante (s)) */

#include <stdio.h>

main() {
        char c;
        unsigned long num;
        double numero;
        while ((int)(c = getchar()) != -1)
        {
                        do putchar (c);
                        while ((c = getchar()) != ' ');
                        while ((c = getchar()) != 'x');
                        num = 0;
                        while ((c = getchar()) != ' ') {
                                num *= 16;
                                switch (c) {
                                        case '0'...'9': num += c - '0';
                                                        break;
                                        case 'a'...'f': num += c - 'a' + 10;
                                                        break;
```

Capitolo 1

L'anomalia della Wireless LAN 802.11b

1.1 Schema di funzionamento dell'802.11b

In una Wireless LAN 802.11b, la trasmissione di una trama consiste di 5 fasi che si susseguono temporalmente (supponendo di trasmettere con continuità, il che avviene a condizione che il trasmettitore abbia sempre dati da inviare):

1. Trasmissione dei dati
2. Attesa di uno *Short InterFrame Space* (SIFS)
3. Ricezione di un *Acknowledgement* (ACK) oppure attesa di un *ACK_ Timeout*
4. Attesa di un *Distributed InterFrame Space* (DIFS)
5. Attesa di un *backoff* di durata casuale, prima della trasmissione della trama successiva

Lo standard 802.11b prevede che il destinatario, quando riceve una trama, controlli la *Frame Check Sequence* (FCS) per verificare che non ci siano errori, ed in caso di esito positivo spedisca al mittente un ACK. Se la trama viene

trasmessa con successo, la sorgente può iniziare la trasmissione di eventuali nuove trame (derivanti da nuovi pacchetti passati al MAC layer dagli strati protocollari superiori). Se invece la stazione trasmettitrice non riceve l'ACK entro un *ACK_ Timeout*, ritrasmette la stessa trama, supponendo che la mancata ricezione dell'ACK sia dovuta ad errori sul canale radio. Il numero massimo di tentativi fatti dalla sorgente è determinato da un parametro, detto *Retry_ Limit*, il cui valore di default è 4 (vedere anche l'appendice A; il valore massimo stabilito dallo standard è 7).

Va precisato che nelle WLAN 802.11b, a causa delle lunghe attese dovute in particolare al backoff, oltre che per il consistente overhead, non ci si può aspettare di avvicinarsi al rate nominale della scheda, ovvero 11 Mbit/s, come evidenziato negli articoli [4] e [5]. Il rapporto fra la quantità di dati effettivamente trasmissibili a livello IP ed il rate nominale cresce all'aumentare della lunghezza media dei pacchetti.

Dato che lo studio del comportamento delle Wireless LAN con pacchetti di dimensione variabile va oltre gli scopi di questa tesi, da qui in avanti saranno considerati sempre pacchetti da 1 KB. Come risulta dalle tabelle 4.2 e 4.3 il tempo necessario, nelle migliori condizioni, per la trasmissione di un datagramma UDP da 1 KB (1024 byte di dati, 1052 byte comprese le intestazioni IP e UDP) è mediamente di 1.602 ms; ne consegue che il massimo goodput a livello IP è dato dal rapporto fra lunghezza del pacchetto e durata della trasmissione, ovvero

$$R_{MAX} = \frac{1052byte \cdot 8bit/byte}{1.602 \cdot 10^{-3}s} \cong 5Mbit/s$$

come confermato da [4] e [5].

1.2 Link-sharing gerarchico nelle Wireless LAN

Nelle reti locali basate sugli standard IEEE 802.x, tipicamente i meccanismi di gestione di classi di servizio differenziate (link-sharing gerarchico, fair


```
                while ((c = getchar()) != ' ');
                n = 0;
                while ((c = getchar()) != '\n') {
                        n *= 10;
                        n += (int)(c - '0');
                }
                if (n < 256) printf(" %d", tabella[n]);
                putchar (c);
        }
}
```

Programma che estrae da ciascuna riga di un file un valore numerico e ne fa l'istogramma su tutto il file (su 10000 intervalli): `istoparser.c`

```
#include <stdio.h>

main() {
        char c;
        int migli, centin, decine, unita, i;
        int tabella [10000];
        for (i = 0; i < 10000; i++) tabella [i] = 0;
        while ((int)(c = getchar()) != -1)
        {
                        while ((c = getchar()) != ' ');
                        getchar();
                        getchar();
                        getchar();
                        migli = (c = getchar()) >= '0' && c <= '9' ? c-'0' : 0;
                        centin = (c = getchar()) >= '0' && c <= '9' ? c-'0' : 0;
                        ++tabella [((migli*10+centin)*10+decine)*10+unita];
                        while ((c = getchar()) != '\n' && (int)c != -1);
        }
        for (i = 0; i < 10000; i++) {
```

```
        184010, 272346, 354042, 401921, 423819, 480759, 516107, 552194, 553785, 564682,
        569708, 573908, 578784, 586438, 589950, 593455, 596960, 600465, 603970, 607475,
        610980, 614485, 617990, 621495, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000,
        625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000, 625000 };

main() {
        char c;
        unsigned int n;
        while ((int)(c = getchar()) != -1)
        {
                do putchar (c);
```


queuing...) sono implementati **all'interno del sistema operativo** o al più nel *device driver*. Ciò accade perché di solito il livello *data link* (MAC + LLC) non espleta funzioni di suddivisione controllata della capacità trasmissiva disponibile.

Il vantaggio dell'implementazione ad alto livello di un meccanismo di scheduling (quale può essere il CBQ o l'HTB) è, oltre alla sua configurabilità semplice e flessibile, l'indipendenza da uno specifico hardware di rete.

Tuttavia, occorre osservare che, implementandolo al di sopra del livello MAC, lo scheduler non si rende conto delle ritrasmissioni a tale livello e perciò schedula, di fatto, il **goodput atteso** anziché le **risorse di banda utilizzate** (paragrafo 1.2.1). Poiché in una Wireless LAN il rapporto tra goodput e risorse necessarie per ottenerlo non è uguale per tutti, lo scheduling di uno stesso goodput a due diverse stazioni mobili può portare a frazioni completamente differenti di risorse trasmissive consumate. Se questo in taluni casi può persino essere desiderabile (ad esempio, per compensare un utente per la cattiva qualità del canale), in altri casi un tale fenomeno risulta inefficiente o in generale non voluto [1].

1.2.1 Il goodput relativo

In questa tesi, il termine *goodput* viene usato per definire la **quantità di dati visti al di sopra del livello MAC**.

In una LAN cablata, ad esempio una Fast Ethernet, dove la probabilità di errore è bassa e tutte le stazioni percepiscono la stessa qualità, l'assunzione che il goodput sia costante non compromette la *fairness*. Ciò non risulta più vero nel caso di una Wireless LAN: la probabilità di errore su un collegamento wireless è significativamente più alta, rendendo necessarie **ritrasmissioni** in caso di errore, nonché tecniche di **modulazione adattativa**.

Ritrasmissioni. Come si può facilmente osservare, le ritrasmissioni (sezione 1.1) allungano il tempo necessario per trasmettere un pacchetto IP, in

quanto per trasmetterne uno è necessario inviare più di una trama, a causa dei tentativi di trasmissione falliti.

Modulazione adattativa. Quando una stazione è costretta a ritrasmettere molto spesso le trame che invia verso una determinata destinazione, abbassa il rate per diminuire il tasso di errore. Quando, viceversa, viene trasmesso con successo, senza ritrasmissioni, un certo numero di trame, il rate viene alzato per aumentare l'efficienza nell'utilizzo delle risorse di rete. Il rate nominale può pertanto variare fra quattro possibili valori: 1 Mbit/s, 2 Mbit/s, 5.5 Mbit/s, 11 Mbit/s. Il fatto che si adotti una velocità trasmissiva inferiore a quella massima (11 Mbit/s) abbassa il goodput in maniera significativa, come vedremo più approfonditamente nel capitolo 4.

Con l'espressione *goodput relativo* ($g(t)$) si intende il rapporto fra il goodput ottenibile in un dato istante verso una determinata destinazione e il goodput massimo del link, che è il goodput ottenibile nelle migliori condizioni, ovvero quando il rate nominale usato è di 11 Mbit/s e non si commettono errori trasmissivi sul canale.

In formule, il goodput relativo è

$$g(t) = \frac{R(t)}{R_{MAX}}$$

dove $R(t)$ è il goodput all'istante t verso la destinazione considerata e R_{MAX} è il goodput massimo.

1.2.2 Scenario esemplificativo

Lo scenario preso in considerazione nelle misure, che verrà poi usato anche nei capitoli successivi per analizzare le prestazioni dello scheduler HTB ed evidenziare vantaggi e limiti degli approcci seguiti nella tesi, è il seguente: un Internet Service Provider (ISP) offre accesso al backbone IP utilizzando un **Access Point** (AP) dotato di scheda wireless compatibile con lo standard


```
        {
                if (c == '\n') righe++;
        }
        printf("%d righe.\n", righe);
}
```

Anche i programmi di cui si riporta qui il codice sorgente vanno compilati con il comando seguente:

`gcc -o file_eseguibile file_sorgente.c`

Programma che inserisce un punto dopo il secondo carattere di ciascuna riga (per convertire il tempo in secondi, dato un file in cui tutte le righe iniziano con un valore temporale espresso in microsecondi): `punto.c`

```
#include <stdio.h>

main() {
        char c;
        int i;
        while ((int)(c = getchar()) != -1)
        {
                        putchar(c);
                        putchar(getchar());
                        putchar ('.');
                        while ((c = getchar()) != '\n') putchar (c);
                        putchar (c);
        }
}
```

Programma che converte le informazioni di debug sulla qualità del canale in goodput stimabile a partire da tali informazioni: `level2cap.c`

```
#include <stdio.h>

unsigned int tabella[256] = { 5000, 5000, 8067, 26490, 64579, 113262,
```

```
        prism2_callback(local, PRISM2_CALLBACK_RX_START);
        stats = hostap_get_stats(dev);
***************
*** 2012,2017 ****
--- 2026,2038 ----
                goto rx_dropped;
        }

+ #ifdef CONFIG_NET_WCHMON_DRIVER_MODULE
+       if (rxdesc.frame_control == UDP)
+               for (i = 0; i < 6; i++) prevaddr[i] = rxdesc.addr1[i];
+       /* signal to WChMon (with MAC address of previous data (UDP) frame */
+       if (rxdesc.frame_control == ACK)
+               wchmon_driver_transmission_status(dev1->wsched_chmon, prevaddr,
(int) rxdesc.time);
+ #endif
        len = le16_to_cpu(rxdesc.data_len);
        hdr_len = sizeof(rxdesc);
        status = le16_to_cpu(rxdesc.status);
```

Piccoli programmi di utilità

Per contare le righe di un file, ecco l'utilissimo programmino `contarighe.c`, da compilare col comando `gcc -o cr contarighe.c` per dare origine al file eseguibile `cr` (la cui sintassi è `cr < nomefile`).

```
#include <stdio.h>

main() {
        int righe = 0;
        char c;
        while ((int)(c = getchar()) != -1)
```


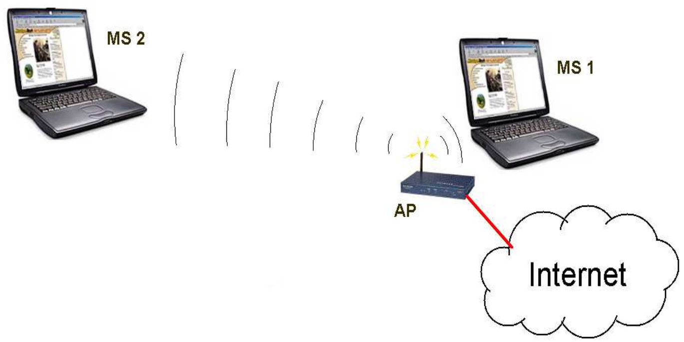

Figura 1.1: Testbed per le misure.

IEEE 802.11b; la capacità trasmissiva totale dichiarata per il link è quella calcolata in funzione della dimensione media dei pacchetti [4] (sicuramente inferiore agli 11 Mbit/s nominali, come già detto a pag. 12). L'AP viene usato per servire **due diversi utenti**: l'utente 1, il cui contratto prevede che gli venga assegnata una frazione α della banda, e l'utente 2, che invece ha diritto al restante $1-\alpha$. Per semplicità, si assume che ciascuno dei due utenti accede all'AP solo con una stazione mobile per volta, rispettivamente MS1 ed MS2. La corretta suddivisione della capacità del link viene garantita dall'Access Point, che utilizza uno **scheduler**.

A causa del multipath, o più semplicemente della diversa distanza di ciascuna MS dall'AP, una stazione può vedere un link di buona qualità mentre l'altra può vederlo di qualità scadente. Perciò, per le due stazioni mobili, il valore del goodput relativo è diverso, ad esempio g_1 per la prima e g_2 per la seconda; uno scheduler ben progettato per una rete cablata fa il possibile per inviare i dati a MS1 e MS2 in proporzione α:$(1-\alpha)$, il che si riflette però in un differente rapporto fra le risorse trasmissive T_1 e T_2 necessarie per

l'effettiva trasmissione di tali dati:

$$T_1 = \frac{\alpha \cdot R}{g_1}$$

mentre

$$T_2 = \frac{(1-\alpha) \cdot R}{g_2}$$

(in entrambe le formule, R è il goodput complessivo verso MS1 e MS2), quindi

$$\frac{T_1}{T_2} = \frac{\alpha}{1-\alpha} \cdot \frac{g_2}{g_1} \neq \frac{\alpha}{1-\alpha}$$

Tenendo conto che $T_1 + T_2 = R_{MAX}$ (costante), se ad esempio $\frac{T_1}{T_2}$ diminuisce è perché T_1 diminuisce e T_2 aumenta, e questo crea **unfairness**, visto che il contratto dell'utente 1 stabilisce che esso utilizzi una quantità di risorse pari a $T_1 = \alpha \cdot R_{MAX}$, almeno finché MS1 si trova a breve distanza dall'AP.

In particolare, quando MS2 si trova in cattiva posizione, $g_2 << 1$ (mentre $g_1 \cong 1$) e il goodput di MS1 diminuisce notevolmente rispetto al caso ideale. L'anomalia della Wireless LAN 802.11b consiste proprio nel fatto che il goodput dell'utente in buona posizione dipende anche dalla posizione degli altri utenti: la situazione problematica, che si cerca di affrontare con vari approcci in questa tesi, è quella in cui **la cattiva qualità di una stazione mobile degrada anche le prestazioni dell'altra**, benché questa abbia a disposizione un canale di buona qualità.

Esaminando il problema da due diversi punti di vista, l'effetto indesiderato può essere visto come:

1. La posizione dell'utente 2 influenza drasticamente la banda disponibile per la trasmissione dei dati verso l'utente 1.

2. L'utente 2 consuma più risorse di rete rispetto alla frazione $1-\alpha$ che gli è stata assegnata, sebbene ci sia un altro utente che ha contemporaneamente bisogno di tali risorse, e che in questo modo si vede sottrarre una parte della sua frazione α.


```
  #ifndef final_version
--- 164,173 ----
  static int prism2_get_ram_size(local_info_t *local);
  #endif /* PRISM2_DOWNLOAD_SUPPORT */

! #ifdef CONFIG_NET_WCHMON_DRIVER_MODULE
! extern u32 wchmon_driver_transmission_status( struct wsched_channel_monitor *mon,
!                                              char *dst, unsigned int status );
! #endif


  #ifndef final_version
***************
*** 1974,1984 ****
--- 1982,1998 ----
  static void prism2_rx(local_info_t *local)
  {
        struct net_device *dev = local->dev;
+       struct net_device *dev1 = dev;
        int res, rx_pending = 0;
        u16 len, hdr_len, rxfid, status, macport;
        struct net_device_stats *stats;
        struct hfa384x_rx_frame rxdesc;
        struct sk_buff *skb = NULL;
+       static u8 prevaddr[6];
+       int i;
+
+       do dev1 = dev1->next;
+       while (strcmp(dev1->name, "wlan1") && strcmp(dev1->name, "eth1") && dev1!=dev)
```

2 in modalità *Monitor* si chiama invece `wlan0` ed è quella identificata dalla variabile `dev`.

2. Registrare nella variabile `prevaddr` l'indirizzo MAC dell'ultima trama dati ricevuta (ovvero trasmessa dall'interfaccia `dev1` e sniffata da `dev`). La trama dati si riconosce dal campo Frame Control di valore pari a 8.

3. Chiamare il TWCM, ed in particolare la funzione `wchmon_driver_transmission_status()`, quando viene ricevuta una trama di Acknowledgement, con il campo Frame Control pari a 212. In tal caso, al TWCM viene passato l'istante di ricezione dell'ACK (`rxdesc.time`) e l'indirizzo MAC della trama dati immediatamente precedente (`prevaddr`).

```
*** linux/drivers/net/wireless/hostap_hw.c      Fri May 28 13:21:57 2004
--- /home/valori/tx/hostap_hw.c Fri May 28 13:25:52 2004
***************
*** 142,147 ****
--- 142,150 ----
  #define HFA384X_INIT_TIMEOUT (HZ / 2) /* 500 ms */
  #define HFA384X_ALLOC_COMPL_TIMEOUT (HZ / 20) /* 50 ms */

+ /* Frame Control words */
+ #define ACK 212
+ #define UDP 8

  static void prism2_hw_reset(struct net_device *dev);
  static void prism2_check_sta_fw_version(local_info_t *local);
***************
*** 161,167 ****
  static int prism2_get_ram_size(local_info_t *local);
  #endif /* PRISM2_DOWNLOAD_SUPPORT */

!
```


1.3 Prove sperimentali dell'anomalia

Una conferma di queste considerazioni viene dalle misure effettuate con schede Lucent *Orinoco* basate sullo standard 802.11b.

Per semplicità, sono stati usati pacchetti UDP di **lunghezza costante** (1024 byte, che diventano 1052 byte se si aggiunge l'overhead dei protocolli UDP e IP). Pertanto, il massimo goodput ottenibile su una rete 802.11b è noto a priori [4].

Il goodput che una stazione mobile ottiene se è l'unica a ricevere varia invece con la qualità del canale, ovvero con il rapporto segnale-rumore (SNR). Se la qualità è **buona** (SNR $\geq$ 18 dB per schede *Orinoco*) esso coincide col goodput massimo, quindi il goodput relativo è unitario. Se, al contrario, la stazione si trova **fuori portata** rispetto all'Access Point (SNR $\leq$ -1 dB per le *Orinoco*), il goodput, e di conseguenza il goodput relativo, diventa nullo. Si possono individuare, con le schede *Orinoco*, fino a cinque casi di qualità distinguibili fra loro:

- **buona:** SNR $\geq$ 18 dB, $g(t) = 1$;
- **media:** SNR $\cong$ 12 dB, $g(t) \cong 0.8$;
- **cattiva:** SNR $\cong$ 8 dB, $g(t) \cong 0.5$;
- **pessima:** SNR $\cong$ 4 dB, $g(t) \cong 0.1$;
- **fuori portata:** SNR $\leq$ -1 dB, $g(t) = 0$.

1.3.1 Scheduler CBQ con rapporto 4:1

La prima prova è stata fatta nello scenario descritto a pag. 15, con $\alpha = 0.8$: una classe di traffico ha diritto all'80% della banda, mentre all'altra viene assegnato il restante 20%. È stato usato lo scheduler CBQ (che esegue le operazioni di link-sharing in modo molto preciso), assegnando 4 Mbit/s alla classe di traffico corrispondente alla destinazione MS1 e 1 Mbit/s alla classe corrispondente a MS2.

In figura 1.2 si può osservare l'andamento del goodput delle due stazioni

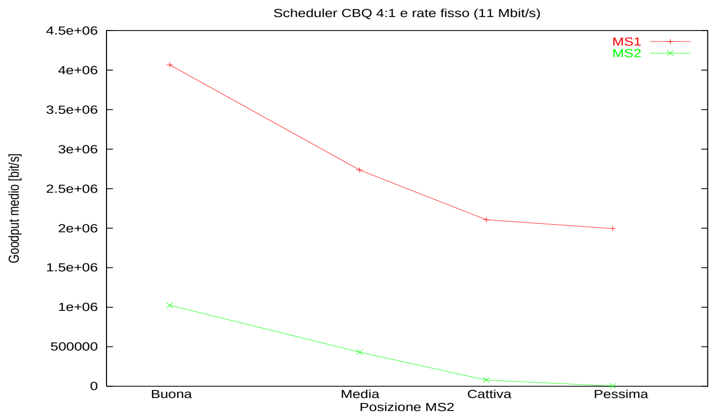

Figura 1.2: Curva prestazioni-qualità dello scheduler CBQ con rapporto 4:1.

mobili al variare della posizione di una di esse, MS2: si nota come, sebbene la stazione MS1 non si muova, il goodput che essa ottiene **diminuisce al peggiorare della qualità del canale fra AP e MS2**.

Questo fenomeno avviene perché lo scheduler trasmette dati alle due destinazioni mobili in proporzione 4:1, senza tenere conto della diversa qualità del canale. Quando il canale è buono (figura 1.3), le due destinazioni ottengono quanto effettivamente si aspettano, cioè rispettivamente 4 Mbit/s e 1 Mbit/s. Quando invece il canale è buono per MS1 ma medio o cattivo per MS2 (figura 1.4), lo scheduler CBQ trasmette sempre i pacchetti in rapporto 4:1, ma per poter trasmettere con successo un pacchetto a MS2 è costretto a volte a ripetuti tentativi di trasmissione, il che comporta un maggior consumo delle risorse di rete da parte di MS2 a danno del goodput di MS1, che così rimane più basso dei 4 Mbit/s cui avrebbe invece diritto.

Il motivo per cui il goodput di MS1 non va mai a zero, anche quando MS2 non riceve nulla (figura 1.5), sta nel fatto che l'AP, in condizioni di qualità pessima, tenta ugualmente di trasmettere verso MS2 (a causa del-


```
            if (status > before + MIN_LENGTH || status < before) {
                duration = status > before ? status - before : SECOND;
                DEBUG(DEBUG_CALIBRATE, "duration = %d\n", duration);
                before = status;
                cap = CONST_LENGTH * SECOND / duration;

                capacity[i] = cap; /* capacity[i] = (3 * capacity[i] + cap) >> 2; */
                if( capacity[i] < capacity_map[0] )
                    capacity[i] = capacity_map[0];
                if( capacity[i] > capacity_map[(CAPACITY_MAP_MAX_LEVEL-CAPACITY_MAP_MIN_LEVE
                    capacity[i] = capacity_map[(CAPACITY_MAP_MAX_LEVEL-CAPACITY_MAP_MIN_LEVE
            }
        FCT_OUT;
        return 0;
        }

        FCT_OUT;
        return capacity[i];
}

EXPORT_SYMBOL (wchmon_driver_transmission_status);
```

Modifiche al driver *HostAP*

Il driver *HostAP*, quando riceve una trama (ricordiamo che nel nostro caso la scheda lavora in modalità *Monitor*, perciò vede sia le trame dati che gli ACK), chiama la funzione `prism2_rx`. Le modifiche apportate a questa funzione sono essenzialmente le tre seguenti:

1. Trovare l'interfaccia `dev1` che trasmette i dati, fra tutte le interfacce nella lista delle `net_device`: questa interfaccia si chiama `eth1` nel caso di scheda *Orinoco*, `wlan1` nel caso di scheda *Prism 2*. La scheda *Prism*

```
 *       is an exponential average over the last status updates _for this channel_.
 */
u32 wchmon_driver_transmission_status( struct wsched_channel_monitor *mon, char *dst,
                                        unsigned int status )
{
  static unsigned int before = 0;
  unsigned int duration;
  static u32 capacity[3] = {6000, 6000, 6000};
  u32 cap = 4000;
  int i = 0;

  FCT_IN;

  if (!mon){
        printk (KERN_DEBUG "No Wireless Channel Monitor!!!\n");
        return capacity_map[0];
  }

  if (dst[5] == 0xFFFFFF8E) i = 1;
  if (dst[5] == 0x6F) i = 2;

  if( dst_indicator == DST_MAC )
    DEBUG( DEBUG_CALIBRATE, "channel %02X %02X %02X %02X %02X %02X: time %d, MS%d\n",
          dst[0], dst[1], dst[2], dst[3], dst[4], dst[5], status, i );

  if (status) {
  /*
   * map status (time) to expected capacity
   * Note: An averaging mechanism here could help to dampen the influence
   * on the scheduler's transmission rate.
   */
```


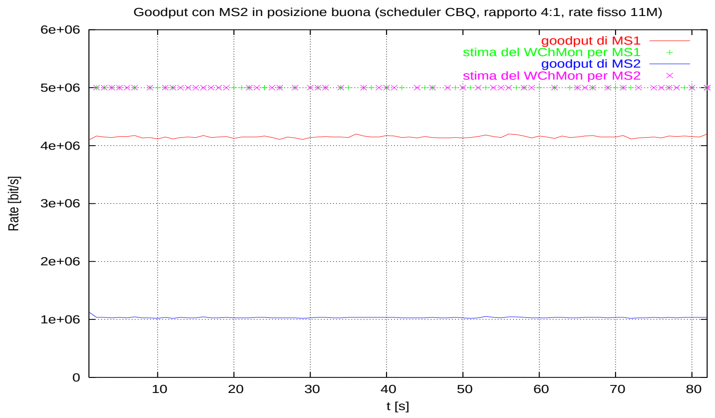

Figura 1.3: Andamento temporale del goodput con entrambe le MS in posizione buona e rate fissato a 11 Mbit/s.

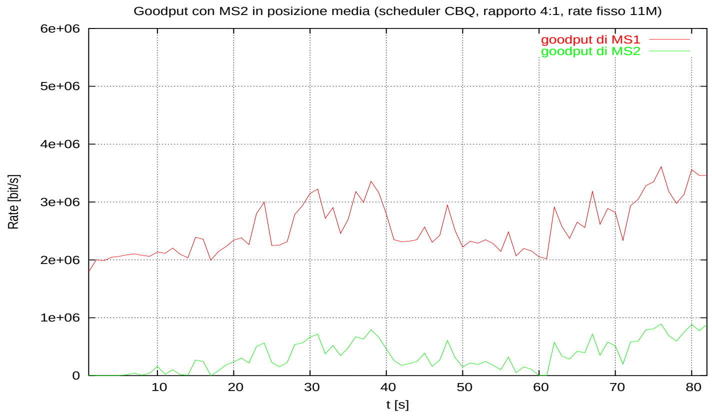

Figura 1.4: Andamento temporale del goodput con MS1 in posizione buona e MS2 in posizione intermedia. Il rate nominale è fissato sempre a 11 Mbit/s.

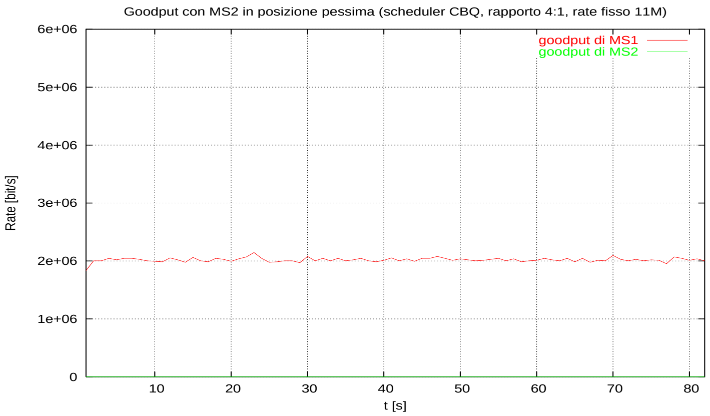

Figura 1.5: Andamento temporale del goodput con MS2 in posizione pessima e rate fissato a 11 Mbit/s.

l'ARP statica, come spiegato nell'appendice A) ma desiste dopo 4 tentativi falliti (a causa del *Retry_ Limit*, sezione 1.1), consentendo pertanto a MS1 di trasmettere comunque i propri dati.

Se invece esaminiamo la figura 1.6, che rappresenta l'andamento della quantità di dati **trasmessi** a MS1 e a MS2 (anziché ricevuti, come in figura 1.2), si vede che il rapporto è sempre 4:1 anche quando il canale verso MS2 è pessimo.

1.3.2 Scheduler CBQ con rapporto 1:1

In questo caso $\alpha = 0.5$, quindi a ciascuna delle due classi viene assegnato il 50% della banda.

Le figure 1.7 e 1.8 mostrano un andamento del goodput decrescente per entrambe le stazioni mobili, benché una sola di esse, MS2, si allontani dall'Access Point.


```
 *
 * Return an estimated value for the available bandwidth (goodput) for
 * the specified destination.
 *
 * Note: Right now, in case we do not have any information we simply
 *       return the minimum capacity - it could be beneficial to be
 *       able to signal this fact to the scheduler.
 */

u32 wchmon_driver_get_capacity( struct wsched_channel_monitor *mon,
  struct wsched_dest *dst )
{
  u32 capa;

  FCT_IN;
  capa = wchmon_driver_transmission_status ( mon, dst->addr, 0 );

  if( dst_indicator == DST_MAC )
    DEBUG( DEBUG_CALIBRATE, "channel %02X %02X %02X %02X %02X %02X: capacity %d\n",
          dst->addr[0], dst->addr[1], dst->addr[2], dst->addr[3], dst->addr[4],
          dst->addr[5], capa );

  FCT_OUT;
  return capa;
}

/*
 * notify channel monitor about the result of a transmission
 *
 * !!! can be called in interrupt context !!!
 *
 * note: The long-term level
```

`wchmon_driver_transmission_status()`, che viene invocata dal driver *HostAP* ogni volta che riceve un ACK.

In questo caso non utilizziamo il formato con le righe di differenza tra il vecchio ed il nuovo file: le due funzioni qui riportate non sono semplici modifiche delle omonime funzioni realizzate in [1], ma sono state completamente riscritte, perciò non avrebbe avuto senso riportare un elenco di differenze.

Mentre `wchmon_driver_get_capacity()` non fa altro che chiamare `wchmon_driver_transmission_status()` con il parametro `status` nullo, il vero lavoro è svolto da quest'ultima funzione. Essa può ricevere due tipi di chiamate:

1. **Informazione** dell'avvenuta ricezione di un ACK da parte del driver: in questo caso il parametro `status` indica l'istante, espresso in μs, della ricezione dell'ACK; facendo la differenza fra questo e l'istante in cui era arrivato l'ACK precedente (variabile `before`), il TWCM calcola il goodput stimato, dividendo la lunghezza del pacchetto per l'intertempo (`cap = CONST_LENGTH * SECOND / duration;`) ed eventualmente filtrando con un IIR passa-basso (attualmente non implementato: `capacity[i] = cap;`).

2. **Richiesta** del goodput stimato da parte della funzione `wchmon_driver_get_capacity()` invocata dallo scheduler: qui `status` vale zero, per distinguere questo caso dal precedente; la funzione si limita a restituire il valore della capacità del canale verso la destinazione mobile di cui è fornito l'indirizzo MAC.

Da notare come, alla fine, il nome della funzione `wchmon_driver_transmission_status()` debba essere **esportato** come simbolo globale, visibile dagli altri moduli del kernel, in quanto il driver *HostAP* deve poterlo utilizzare per chiamare la funzione.

```
/*
 * wchmon_driver_get_capacity
```


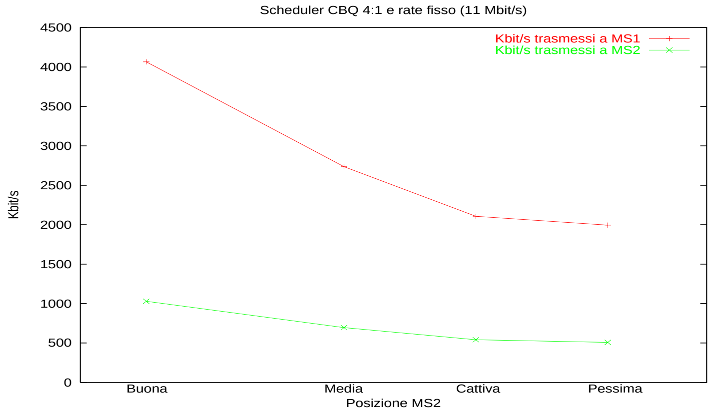

Figura 1.6: Curva prestazioni-qualità dello scheduler CBQ basata sui pacchetti trasmessi con rate 11 Mbit/s e rapporto 4:1.

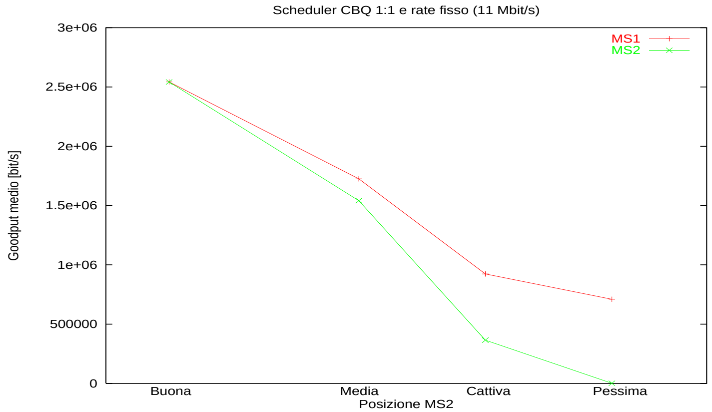

Figura 1.7: Curva prestazioni-qualità dello scheduler CBQ con rapporto 1:1.

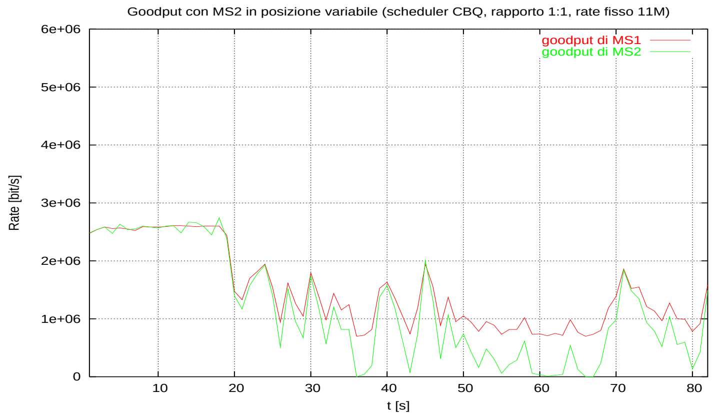

Figura 1.8: Andamento temporale del goodput per le due stazioni mobili (scheduler CBQ con rapporto 1:1), quando la qualità del canale verso MS2, inizialmente buona, diventa media e poi cattiva, in quanto MS2 si allontana dall'AP.

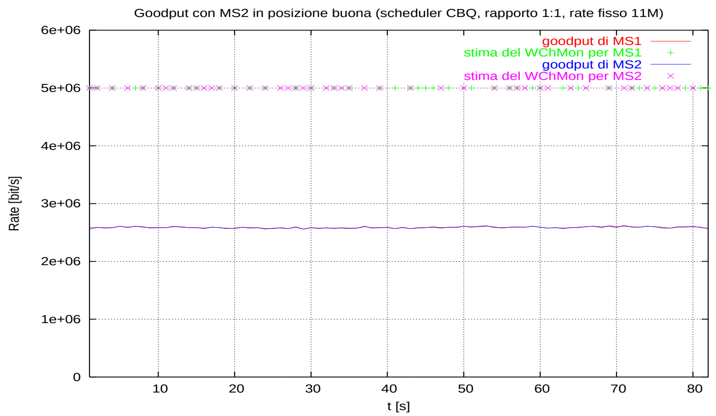

Figura 1.9: Andamento temporale del goodput per le due stazioni mobili entrambe in buona posizione (scheduler CBQ con rapporto 1:1).


```
                                        sch->q.qlen--;
                                        sch->flags &= ~TCQ_F_THROTTLED;
***************
*** 1193,1198 ****
--- 1206,1216 ----
          for (i = 0; i < TC_HTB_NUMPRIO; i++)
                  INIT_LIST_HEAD(q->drops+i);

+         if ( wsched_wchmon_add( sch->dev, "driver" ) != 0)
+           {
+                printk("htb: Unable to add wireless channel monitor to device!\n");
+                return -EINVAL;
+         }
          init_timer(&q->timer);
          skb_queue_head_init(&q->direct_queue);

***************
*** 1402,1407 ****
--- 1420,1426 ----

          htb_destroy_filters(&q->filter_list);
          __skb_queue_purge(&q->direct_queue);
+         wsched_wchmon_del( sch->dev );
          MOD_DEC_USE_COUNT;
  }
```

Modifiche al *Wireless Channel Monitor*

In questa sezione vediamo l'implementazione di due funzioni del TWCM: `wchmon_driver_get_capacity()`, chiamata dal WHTB/TWHTB con l'istruzione `capacity = mon->get_capacity (mon, &cur_dst);`

```
+               }
                if ((cl->un.leaf.deficit[level] -= skb->len) < 0) {
                        HTB_DBG(4,2,"htb_next_cl oldptr=%p quant_add=%d\n",
                                level?cl->parent->un.inner.ptr[prio]:q->ptr[0][prio],
cl->un.leaf.quantum);
***************
*** 1006,1011 ****
--- 1016,1024 ----
                if (!cl->un.leaf.q->q.qlen)
                        htb_deactivate (q,cl);
                htb_charge_class (q,cl,level,skb->len);
+               if(mon) {
+                       mon->start_transmit(mon, &cur_dst, skb->len);
+               }
        }
        return skb;
  }
***************
*** 1072,1078 ****
                while (m != (int)(-1)) {
                        int prio = ffz (m);
                        m |= 1 << prio;
!                       skb = htb_dequeue_tree(q,prio,level);
                        if (likely(skb != NULL)) {
                                sch->q.qlen--;
                                sch->flags &= ~TCQ_F_THROTTLED;
--- 1085,1091 ----
                while (m != (int)(-1)) {
                        int prio = ffz (m);
                        m |= 1 << prio;
!                       skb = htb_dequeue_tree(sch,q,prio,level);
                        if (likely(skb != NULL)) {
```


Quando la posizione di MS2 è buona come quella di MS1 (figura 1.9) lo scheduler CBQ assegna esattamente la stessa banda ad entrambe. Quando la posizione di MS2 è invece cattiva (figura 1.10), lo scheduler CBQ cerca

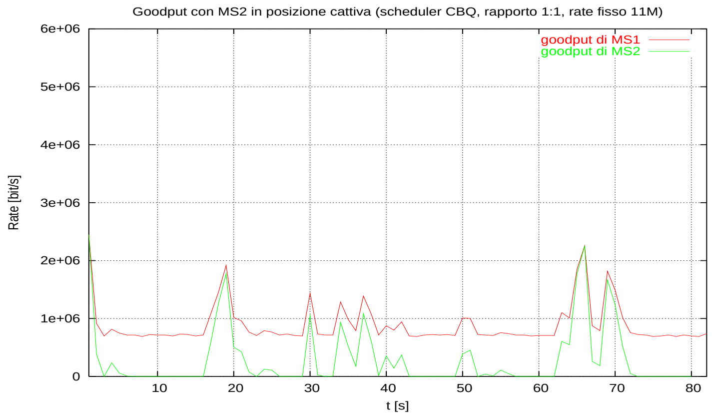

Figura 1.10: Andamento temporale del goodput per le due stazioni mobili con MS2 in cattiva posizione (scheduler CBQ con rapporto 1:1).

comunque di assegnare la stessa banda ad entrambe, ma dal momento che l'AP impiega più del dovuto a trasmettere i pacchetti destinati a MS2, utilizza in realtà **più risorse trasmissive** di quante ne usa per trasmettere verso MS1 (si veda poi il paragrafo 1.3.5). Pertanto riesce a trasmettere meno dati a MS1, in quanto ad essa riserva ora meno del 50% della banda, a cui MS1 avrebbe invece diritto.

La differenza di goodput fra MS1 ed MS2 è dovuta unicamente alla perdita di alcuni pacchetti (nel caso di qualità cattiva, la maggior parte) destinati a MS2, perché l'Access Point, raggiunto il *Retry_Limit* (paragrafo 1.1), scarta il pacchetto e passa al successivo. Se infatti esaminiamo la figura 1.11, che rappresenta l'andamento della quantità di dati **trasmessi** a MS1 e a

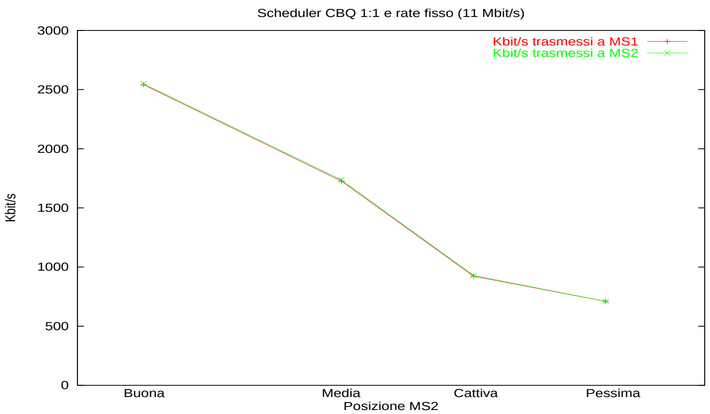

Figura 1.11: Curva prestazioni-qualità dello scheduler CBQ basata sui pacchetti trasmessi con rate 11 Mbit/s e rapporto 1:1.

MS2 (anziché ricevuti, come in figura 1.7), si vede che il rapporto è sempre paritario anche quando il canale verso MS2 è pessimo.

1.3.3 Effetto del rate adattativo

Le misure precedenti erano state eseguite fissando il rate nominale a 11 Mbit/s. Ciò è stato fatto per isolare l'effetto delle ritrasmissioni e verificarne l'impatto negativo sulle prestazioni.

In realtà spesso si lascia che il rate vari in maniera adattativa, a seconda del tasso di errore riscontrato. In generale, in questo modo si riesce ad ottenere un goodput più alto (per SNR non eccessivamente bassi), grazie alla diminuzione della BER causata dall'abbassamento del rate. Nella figura 1.12 si vede come la MS2 tragga vantaggio dal rate adattativo: infatti, quando la qualità del canale è intermedia (cioè tale da provocare errori occasionali ma non da compromettere totalmente la ricezione), essa riceve molti pacchetti in più rispetto al caso in cui il rate rimane fisso a 11 Mbit/s.


```
*** 949,955 ****
  /* dequeues packet at given priority and level; call only if
     you are sure that there is active class at prio/level */
  static struct sk_buff *
! htb_dequeue_tree(struct htb_sched *q,int prio,int level)
  {
        struct sk_buff *skb = NULL;
        struct htb_class *cl,*start;
--- 950,956 ----
  /* dequeues packet at given priority and level; call only if
     you are sure that there is active class at prio/level */
  static struct sk_buff *
! htb_dequeue_tree(struct Qdisc *sch,struct htb_sched *q,int prio,int level)
  {
        struct sk_buff *skb = NULL;
        struct htb_class *cl,*start;
***************
*** 995,1000 ****
--- 996,1010 ----
        } while (cl != start);

        if (likely(skb != NULL)) {
+               u32 capacity = 2222;    // if you see 2222 in /var/log/debug,
something isn't working...
+               struct wsched_dest cur_dst;
+               struct wsched_channel_monitor *mon=sch->dev->wsched_chmon;
+
+               if(mon) {
+                       mon->skb2dst (mon, skb, &cur_dst);
+                       capacity = mon->get_capacity (mon, &cur_dst);
+                       HTB_DBG(13,2,"channel capacity of %u byte/s for class %X\n",
capacity,cl->classid&7);
```

```
+         if ( wsched_wchmon_add( sch->dev, "driver" ) != 0)
+        {
+                printk("cbq: Unable to add wireless channel monitor to device!\n");
+                return -EINVAL;
+        }
         init_timer(&q->wd_timer);
         q->wd_timer.data = (unsigned long)sch;
         q->wd_timer.function = cbq_watchdog;
***************
*** 1749,1754 ****
--- 1764,1770 ----
         }

         qdisc_put_rtab(q->link.R_tab);
+        wsched_wchmon_del (sch->dev);
         MOD_DEC_USE_COUNT;
  }
```

Per lo scheduler HTB le modifiche sono le seguenti:

```
*** linux/net/sched/sch_htb.c   Tue May 11 16:58:25 2004
--- /home/valori/tx/sch_htb.c   Tue May 11 16:56:17 2004
***************
*** 94,99 ****
--- 94,100 ----
   10 - rate estimator
   11 - classifier
   12 - fast dequeue cache
+  13 - wireless channel monitor

   L is level; 0 = none, 1 = basic info, 2 = detailed, 3 = full
   q->debug uint32 contains 16 2-bit fields one for subsystem starting
***************
```


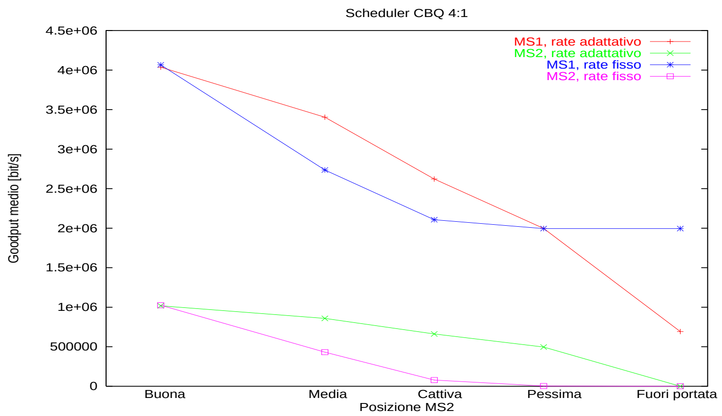

Figura 1.12: Confronto fra rate fisso e adattativo (scheduler CBQ con rapporto 4:1).

Tuttavia, occorre notare che invece il goodput di MS1 non risulta molto diverso nei due casi: se infatti, con un rate fisso, essa risente del fatto che MS2 ritrasmette più volte le sue trame, nel caso di rate adattativo il suo goodput viene danneggiato dall'aumento del tempo impiegato da MS2 a trasmettere una singola trama. Nel caso in cui MS2 sia fuori portata, la penalizzazione per MS1 è più forte quando il rate è adattativo: quella che lo scheduler vede come singola trasmissione di un pacchetto verso MS2 corrisponde in realtà a 4 trame, tante quanti sono i tentativi falliti di trasmissione (dopo i quali l'AP desiste e scarta il pacchetto) ma in un caso tali tentativi avvengono con rate fissato a 11 Mbit/s, nell'altro con il rate minimo possibile, 1 Mbit/s, ancora però non sufficiente a consentire una corretta ricezione da parte di MS2.

Nella figura 1.13 si conferma un crollo del goodput brusco ma spostato in avanti quando il rate è adattativo, rispetto ad una discesa più graduale nel caso di rate fisso, che comunque presenta una degradazione notevole già con un leggero peggioramento della qualità del canale.

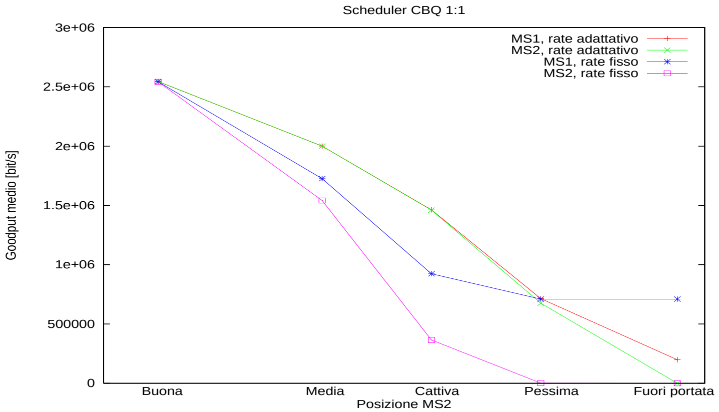

Figura 1.13: Confronto fra rate fisso e adattativo (scheduler CBQ con rapporto 1:1).

Analisi delle tracce dei pacchetti. È opportuno puntualizzare che la degradazione delle prestazioni per MS1 non è dovuta ad una riduzione del proprio rate istantaneo: in entrambi i casi **MS1 trasmette le proprie trame al rate massimo** (11 Mbit/s nominali). La traccia dei pacchetti (figura 1.14) dimostra quanto appena affermato: osservando le tre rette orizzontali (equidistanti, a 8 ms l'una dall'altra), si può notare come il numero di pacchetti fra la retta intermedia e quella superiore sia uguale al numero di pacchetti fra la retta inferiore e quella intermedia. Ciò equivale a dire che in entrambi i casi (rate fisso e rate adattativo) diversi pacchetti consecutivi vengono ricevuti da MS1 a distanza di circa 1.6 ms uno dall'altro, ovvero ad un **rate effettivo di 5 Mbit/s** (corrispondente ad 11 Mbit/s nominali, con i pacchetti da 1024 byte utilizzati. La diminuzione del goodput medio di MS1 è dovuta invece all'attesa che l'AP trasmetta il pacchetto spettante a MS2, attesa che risulta maggiore nel caso in cui il rate è adattativo.


```
                                        goto skip_class;

+                           u32 capacity = 2222;    // if you see 2222 in /var/log/debug,
something isn't working...
+                           struct wsched_dest cur_dst;
+                           struct wsched_channel_monitor *mon=sch->dev->wsched_chmon;
+
+                           if(mon) {
+                                   mon->skb2dst (mon, skb, &cur_dst);
+                                   capacity = mon->get_capacity (mon, &cur_dst);
+                                   printk(KERN_DEBUG "channel capacity of %u byte/s
for class %X\n",capacity,cl->classid&7);
+                           }
                            cl->deficit -= skb->len;
                            q->tx_class = cl;
                            q->tx_borrowed = borrow;
***************
*** 922,927 ****
--- 931,937 ----
                                    cl = cl->next_alive;
                                    cl->deficit += cl->quantum;
                            }
+                           if(mon) mon->start_transmit(mon, &cur_dst, skb->len);
                            return skb;

  skip_class:
***************
*** 1438,1443 ****
--- 1448,1458 ----
        q->link.minidle = -0x7FFFFFFF;
        q->link.stats.lock = &sch->dev->queue_lock;
```

```
+       }
        init_timer(&q->timer);
        skb_queue_head_init(&q->direct_queue);

***************
*** 1402,1407 ****
--- 1423,1429 ----

        htb_destroy_filters(&q->filter_list);
        __skb_queue_purge(&q->direct_queue);
+       wsched_wchmon_del( sch->dev );
        MOD_DEC_USE_COUNT;
  }
```

Modifiche agli scheduler CBQ e HTB per la stima del canale

Per confrontare i risultati, ottenuti con il WHTB e il TWHTB, con le prestazioni degli scheduler tradizionali, è utile inserire le chiamate al *Wireless Channel Monitor* anche in questi ultimi, per poter verificare come i cattivi risultati di questi scheduler siano fortemente correlati alla diminuzione del goodput ottenibile verso la destinazione in posizione peggiore.

Per lo scheduler CBQ le modifiche sono le seguenti:

```
*** linux/net/sched/sch_cbq.c   Fri Dec 21 18:42:06 2001
--- linux/net/sched/sch_cbq.c   Wed Jan 21 15:48:56 2004
***************
*** 903,908 ****
--- 903,917 ----
                        if (skb == NULL)
```


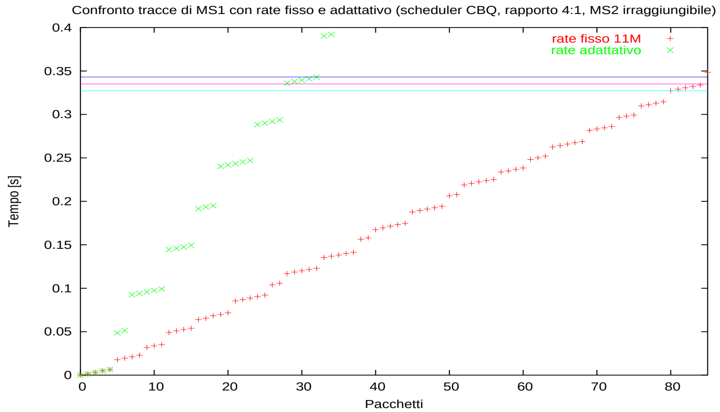

Figura 1.14: Confronto fra rate fisso e adattativo (tracce relative a MS1).

1.3.4 Scheduler FIFO

Anche in assenza di uno scheduler che cerchi di suddividere la capacità trasmissiva in maniera equa fra le classi, la stazione MS1, pur rimanendo ferma, risente della posizione variabile di MS2.

Lo scheduler FIFO non fa distinzioni di classe, ma trasmette i pacchetti nell'ordine in cui arrivano nell'unica coda. Il risultato è che il traffico viene smaltito approssimativamente nelle proporzioni in cui entra. La figura 1.15 mostra ciò che accade quando il traffico in ingresso è distribuito in modo equilibrato fra le due destinazioni: il leggero sbilanciamento a favore di MS1 è dovuto all'asimmetria del generatore di traffico (che trasmette deterministicamente i pacchetti destinati a MS2 subito dopo quelli destinati a MS1), mentre per il resto l'andamento è simile a quello dello scheduler CBQ. Anche qui il basso goodput di MS2 è accompagnato da una riduzione del goodput di MS1, in questo caso non perché lo scheduler ritenga di dover trasmettere lo stesso numero di pacchetti a ciascuna destinazione (come nel caso del CBQ con rapporto 1:1), ma semplicemente perché i pacchetti sono accodati

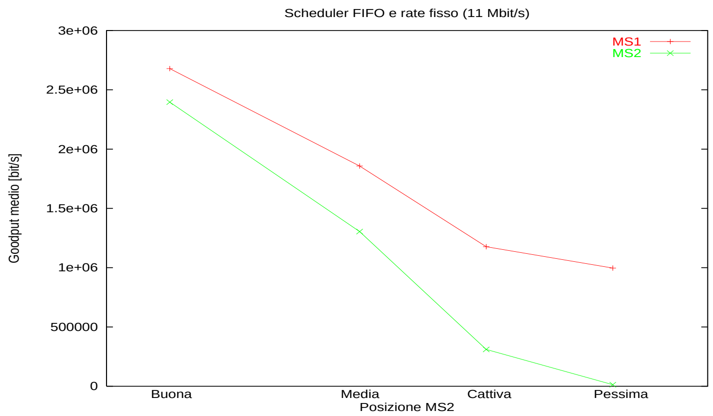

Figura 1.15: Curva prestazioni-qualità dello scheduler FIFO.

a seconda dell'intensità del traffico entrante e quindi vengono estratti dalla coda e trasmessi con quello stesso rapporto (molto vicino a 1:1) senza tener conto che quelli destinati a MS2 impiegano più tempo se la qualità del canale non è perfetta.

Nella figura 1.16 è mostrato un esempio di come varia il goodput nel tempo al variare della posizione di MS2, in assenza di uno scheduler con classi di traffico differenziate. Anche in questo caso possiamo verificare la dipendenza del goodput sul link da AP a MS1 dalla qualità del canale da AP a MS2, ad ulteriore conferma dell'anomalia della Wireless LAN 802.11b.

1.3.5 Risorse trasmissive utilizzate

Eseguendo infine un raffronto tra le risorse effettivamente consumate dalle due stazioni mobili, si osserva in tutti i casi come la stazione in cattiva posizione, MS2, utilizzi il canale più di quanto ne abbia diritto.

Dalle figure 1.17 e 1.18 emerge chiaramente l'effetto drastico della variazione della posizione di una stazione mobile sulla suddivisione delle risorse


```
!                       mon->start_transmit(mon, &cur_dst, skb->len);
!               }
        }
        return skb;
  }
***************
*** 1072,1078 ****
                while (m != (int)(-1)) {
                        int prio = ffz (m);
                        m |= 1 << prio;
!                       skb = htb_dequeue_tree(q,prio,level);
                        if (likely(skb != NULL)) {
                                sch->q.qlen--;
                                sch->flags &= ~TCQ_F_THROTTLED;
--- 1088,1094 ----
                while (m != (int)(-1)) {
                        int prio = ffz (m);
                        m |= 1 << prio;
!                       skb = htb_dequeue_tree(sch,q,prio,level);
                        if (likely(skb != NULL)) {
                                sch->q.qlen--;
                                sch->flags &= ~TCQ_F_THROTTLED;
***************
*** 1193,1198 ****
--- 1209,1219 ----
        for (i = 0; i < TC_HTB_NUMPRIO; i++)
                INIT_LIST_HEAD(q->drops+i);

+       if ( wsched_wchmon_add( sch->dev, "driver" ) != 0)
+         {
+               printk("htb: Unable to add wireless channel monitor to device!\n");
+               return -EINVAL;
```

```
!               struct wsched_dest cur_dst;
!               struct wsched_channel_monitor *mon=sch->dev->wsched_chmon;
!               int stretched = skb->len;
!
!               if(mon) {
!                       mon->skb2dst (mon, skb, &cur_dst);
!                       capacity = mon->get_capacity (mon, &cur_dst);
!                       HTB_DBG(13,2,"channel capacity of %u byte/s for class %X\n",
capacity,cl->classid&7);
!               }
!               stretched *= WIRELESS_MAX_CAPACITY / capacity;
!               if ((cl->un.leaf.deficit[level] -= (stretched < cl->un.leaf.quantum) ?
stretched : cl->un.leaf.quantum) < 0) {
                        HTB_DBG(4,2,"htb_next_cl oldptr=%p quant_add=%d\n",
                                level?cl->parent->un.inner.ptr[prio]:q->ptr[0][prio],
cl->un.leaf.quantum);
                        cl->un.leaf.deficit[level] += cl->un.leaf.quantum;
***************
*** 1005,1011 ****
                   gives us slightly better performance */
                if (!cl->un.leaf.q->q.qlen)
                        htb_deactivate (q,cl);
!               htb_charge_class (q,cl,level,skb->len);
        }
        return skb;
  }
--- 1018,1027 ----
                   gives us slightly better performance */
                if (!cl->un.leaf.q->q.qlen)
                        htb_deactivate (q,cl);
!               htb_charge_class (q,cl,level,stretched);
!               if(mon) {
```


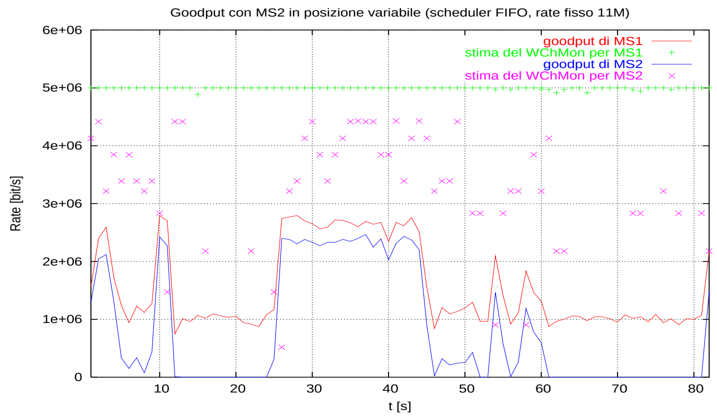

Figura 1.16: Andamento temporale del goodput con qualità variabile del canale fra AP e MS2 (scheduler FIFO): qualità buona fra t=25s e t=44s, cattiva fra t=11s e t=25s e per t>60s.

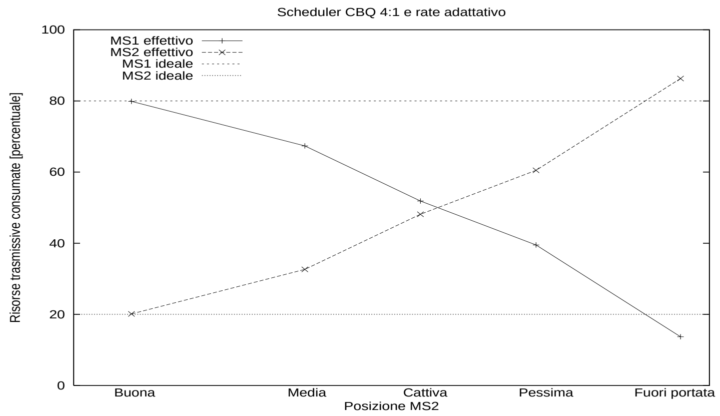

Figura 1.17: Percentuale di utilizzo del canale con rate adattativo, nel caso in cui sia presente uno scheduler CBQ con rapporto 4:1.

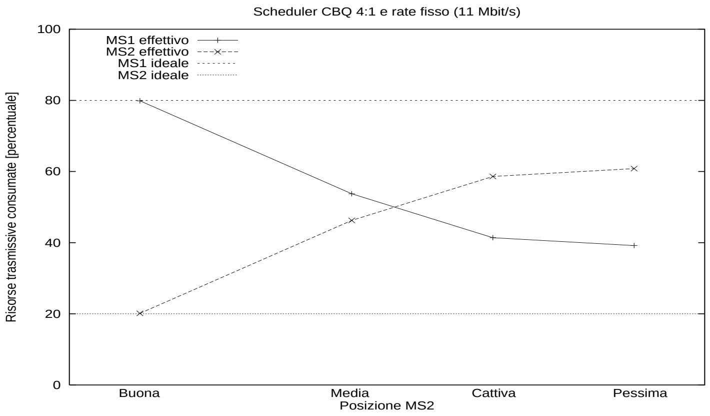

Figura 1.18: Percentuale di utilizzo del canale con rate fisso, nel caso in cui sia presente uno scheduler CBQ con rapporto 4:1.

trasmissive utilizzate: mentre le due MS, quando sono entrambe vicine all'Access Point, ricevono rispettivamente l'80% ed il 20% loro assegnati, quando MS2 si allontana si ha un significativo aumento delle risorse da essa usate, a scapito di quelle a disposizione di MS1, ferma in buona posizione. Addirittura, quando MS2 è in posizione pessima (e nel caso di rate fisso anche in posizione cattiva), le risorse trasmissive consumate da MS2 sono maggiori di quelle usate da MS1, benché il rate cui MS2 avrebbe diritto sia solo $\frac{1}{4}$ di quello assegnato a MS1.

L'ultima figura, la 1.19, mostra ciò che accade quando alle due stazioni mobili viene assegnato lo stesso rate. Mentre il rapporto fra i rispettivi goodput resta paritario al variare della qualità del canale fra AP e MS2, si nota come MS2, che (salvo nel caso di qualità buona per entrambe le stazioni) si trova in posizione peggiore rispetto a MS1, utilizzi l'AP in misura maggiore. In altre parole, MS2 consuma ben più del 50% delle risorse, quota a cui avrebbe diritto, ed è questo il problema di *fairness* che limita le prestazioni


```
***************
*** 949,955 ****
  /* dequeues packet at given priority and level; call only if
     you are sure that there is active class at prio/level */
  static struct sk_buff *
! htb_dequeue_tree(struct htb_sched *q,int prio,int level)
  {
        struct sk_buff *skb = NULL;
        struct htb_class *cl,*start;
--- 951,957 ----
  /* dequeues packet at given priority and level; call only if
     you are sure that there is active class at prio/level */
  static struct sk_buff *
! htb_dequeue_tree(struct Qdisc *sch,struct htb_sched *q,int prio,int level)
  {
        struct sk_buff *skb = NULL;
        struct htb_class *cl,*start;
***************
*** 995,1001 ****
        } while (cl != start);

        if (likely(skb != NULL)) {
!               if ((cl->un.leaf.deficit[level] -= skb->len) < 0) {
                        HTB_DBG(4,2,"htb_next_cl oldptr=%p quant_add=%d\n",
                                level?cl->parent->un.inner.ptr[prio]:q->ptr[0][prio],
cl->un.leaf.quantum);
                        cl->un.leaf.deficit[level] += cl->un.leaf.quantum;
--- 997,1014 ----
        } while (cl != start);

        if (likely(skb != NULL)) {
!               u32 capacity = 2222;
```

scheduler per trasmetterlo al driver dell'interfaccia di rete, chiede al *Wireless Channel Monitor* (puntatore `*mon`) quanto vale il goodput verso la destinazione `cur_dst` del pacchetto.

La funzione usata è `mon->get_capacity`, sviluppata nel *Wireless Channel Monitor* [1]. Il risultato di questa funzione, assegnato alla variabile `capacity`, viene usato per dividere la lunghezza reale del pacchetto per il goodput relativo, in modo da ottenere la lunghezza fittizia, come spiegato nella sezione 2.4: l'istruzione che esegue ciò è `stretched *= WIRELESS_MAX_CAPACITY / capacity;`

Di seguito si riporta la patch `htb-wireless.patch` per esteso:

```
*** linux/net/sched/sch_htb.c   Mon May 17 19:33:01 2004
--- /home/valori/tx/sch_whtb.c  Mon May 17 19:32:56 2004
***************
*** 68,73 ****
--- 68,74 ----
      one less than their parent.
  */

+ #define WIRELESS_MAX_CAPACITY 625000  /* maximum channel capacity (byte/s) */
  #define HTB_HSIZE 16  /* classid hash size */
  #define HTB_EWMAC 2   /* rate average over HTB_EWMAC*HTB_HSIZE sec */
  #define HTB_DEBUG 1   /* compile debugging support (activated by tc tool) */
***************
*** 94,99 ****
--- 95,101 ----
   10 - rate estimator
   11 - classifier
   12 - fast dequeue cache
+  13 - wireless channel monitor

   L is level; 0 = none, 1 = basic info, 2 = detailed, 3 = full
   q->debug uint32 contains 16 2-bit fields one for subsystem starting
```


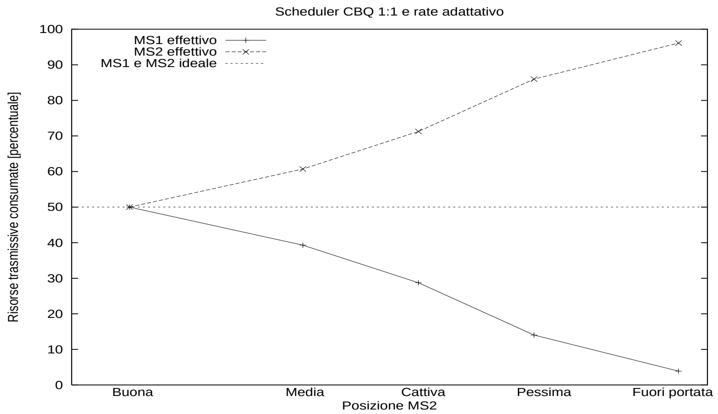

Figura 1.19: Percentuale di utilizzo del canale con rate adattativo, nel caso in cui sia presente uno scheduler CBQ con rapporto 1:1.

della WLAN 802.11b e il cui superamento con diversi approcci sarà oggetto dei prossimi capitoli.

Appendice B

Files di codice sorgente C implementato

In questa appendice sono riportate le modifiche al codice sorgente C del kernel di Linux, secondo lo standard delle differenze tra file (*patches*) che questo sistema operativo usa. Le *patches* sono state ottenute con il comando `diff -c old_file new_file` ed il significato delle indicazioni che compaiono è il seguente:

`***************` separazione di parti diverse del file

`***` n_1, n_2 `***` righe dalla n_1-esima alla n_2-esima di `old_file`

`---` n_1, n_2 `---` righe dalla n_1-esima alla n_2-esima di `new_file`

`+` questa riga viene aggiunta

`-` questa riga viene eliminata

`!` questa riga viene modificata

Modifiche allo scheduler HTB

Come già visto nel capitolo 2, lo scheduler WHTB ha bisogno di una stima del goodput verso una determinata destinazione. Allora esso, all'interno della funzione `htb_dequeue_tree()`, che estrae un pacchetto dalla coda dello

I file eseguibili `l2c`, `ep3` ed `ep4` derivano rispettivamente dai sorgenti `level2cap.c`, `eliminaparole3.c` ed `eliminaparole4.c`, listati nell'appendice B.

Capitolo 2

Lo scheduler HTB su Wireless LAN

2.1 Schema di uno scheduler *channel-aware*

L'obiettivo di questa fase della tesi è stato quello di sfruttare una **stima della capacità trasmissiva** per ridurre il degrado delle prestazioni dovuto alla presenza di una classe di traffico con cattiva qualità del canale.

La figura 2.1 mostra uno schema di principio di uno scheduler *channel-aware* come il WHTB, sviluppato in questa tesi, e l'H-FSC modificato, descritto in [1]: i principali elementi di novità rispetto ad uno scheduler tradizionale sono due, lo **stimatore di canale** e la **modifica dello scheduler**.

Lo stimatore di canale è, come gli scheduler ed i driver delle schede di rete, un modulo del kernel del sistema operativo dell'Access Point, che lavora parallelamente agli scheduler. Lo stimatore di canale non si sostituisce ad uno scheduler, ma lo affianca: la sua funzione è quella di ricevere dal driver dell'interfaccia wireless alcune informazioni utili per monitorare la qualità del canale di trasmissione verso ciascuna destinazione. Ciò che lo stimatore deve fornire allo scheduler, quando esso lo richiede, è una stima, quanto più accurata possibile, della qualità del canale verso una certa destinazione.

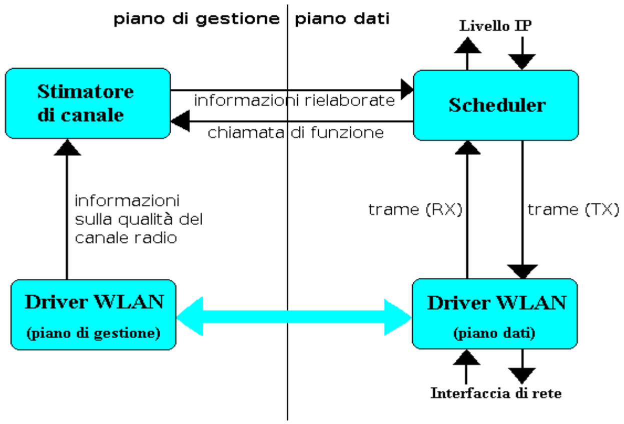

Figura 2.1: Schema a blocchi di uno scheduler che tenga conto della qualità del canale radio.

Quando lo scheduler deve estrarre un pacchetto dalla coda, ha infatti bisogno di conoscere la qualità del canale radio che collega l'AP alla destinazione del pacchetto, in modo da valutarne l'effettivo ammontare di risorse consumate. Di conseguenza, lo scheduler chiama una funzione, all'interno dello stimatore di canale, che restituisce delle informazioni sulla qualità verso la destinazione richiesta, rielaborate opportunamente in un formato che possa essere utilizzato dallo scheduler (ad esempio sotto forma di goodput stimato).

Nelle sezioni 2.2 e 2.3 saranno descritti i moduli che compongono la particolare implementazione realizzata in questa tesi. Essi sono il *Wireless Channel Monitor* (sezione 2.2) e lo scheduler HTB (sezione 2.3). Il sistema complessivo, denominato WHTB e presentato nella sezione 2.4, è rappresentato, con i moduli di cui effettivamente si compone, nella figura 2.15.

La seconda riga serve per verificare di non aver dimenticato di cancellare le prime due righe dai file `nome_misura` nelle cartelle `escher/` e `russell/`.

I file `nome_misura-cap2` e `nome_misura-cap4` si ottengono invece con il seguente comando:

`./cap debug/nome_misura grafici/nome_misura-cap2 grafici/nome_misura-cap4`

Lo script `cap` seleziona dal file `debug/nome_misura` (che è la copia di `/var/log/debug`, come spiegato al punto 4 della sezione precedente) le righe contenenti le informazioni sul goodput stimato verso MS1 e verso MS2, le mette in due file separati `tmp1` e `tmp2`, e con le utility `Conv` e `awk.abs2bit` li converte in due file `nome_misura-cap2` e `nome_misura-cap4`, nella cartella `grafici/`, nel formato standard a due colonne, utilizzabile da `gnuplot`. Nel caso del *Wireless Channel Monitor*, lo script `cap` è il seguente:

```
#!/bin/bash
# Usage: ./cap debug-file out-2 out-4
grep longterm $1 | grep "6E 8E" | ./ep3 > tmp1
./Conv tmp1 tmp2
./l2c < tmp2 | awk -f awk.abs2bit > $2
grep longterm $1 | grep -v "6E 8E" | ./ep3 > tmp1
./Conv tmp1 tmp2
./l2c < tmp2 | awk -f awk.abs2bit > $3
rm tmp1 tmp2
```

Nel caso del TWCM, lo script `cap` è il seguente:

```
#!/bin/bash
# Usage: ./cap debug-file out-2 out-4
./elaboradebug $1 | grep "6E 8E" | ./ep4 > tmp1
./Conv tmp1 tmp2
awk -f awk.abs2bit < tmp2 > $2
./elaboradebug $1 | grep -v "6E 8E" | ./ep4 > tmp1
./Conv tmp1 tmp2
awk -f awk.abs2bit < tmp2 > $3
rm tmp1 tmp2
```

Variazione temporale del goodput Il secondo tipo di grafici rappresenta il goodput istantaneo per ciascuna delle due stazioni mobili, MS1 e MS2, e la stima del *Wireless Channel Monitor* o del TWCM per il canale wireless fra l'AP ed ognuna di esse.

I comandi `gnuplot` sono i seguenti (file `grafici/neltempo.plt`):

```
set yrange [0:6000000]
set xlabel "t [s]"
set ylabel "Rate [bit/s]"
set grid
set title "Goodput con MS2 in posizione buona (scheduler TWHTB 4:1, rate adattativo)"
set t po eps enh c so
set o "../neltempo/TWHTB-buoni-auto.eps"
plot [1:82] "twhtb-buoni-escher" t "goodput di MS1" w l, "twhtb-buoni-cap2" t "s
tima del TWCM per MS1", "twhtb-buoni-russell" t "goodput di MS2" w l, "twhtb-buo
ni-cap4" t "stima del TWCM per MS2"
set t x11
quit
```

Anche qui si può fare il *copia e incolla*, facendo attenzione a non far spezzare le righe del file (altrimenti `gnuplot` protesta).

I file `nome_misura-escher` e `nome_misura-russell` si ottengono con lo script AWK `rate1-1024`, che calcola il rate su finestre di 1s, supponendo che i pacchetti abbiano dimensione media 1024 byte, a partire dai file ottenuti da MGEN in ricezione sulle stazioni mobili. Avendo precedentemente salvato tali file nelle cartelle `escher/` e `russell/`, il comando è del tipo:
`./rate1-1024 escher/nome_misura grafici/nome_misura-escher`
Lo script `rate1-1024` è fatto così:

```
#!/bin/bash
head $1 -n 1
./Conv $1 abs
awk -f awk.tethereal2abstime < abs | awk -f awk.aggrega t=1 size=1052 >$2
rm abs
```


2.2 Il *Wireless Channel Monitor*

2.2.1 Qualità del canale radio e stima del goodput

Uno strumento che si è rivelato molto utile per valutare la qualità del canale nella WLAN è il *Wireless Channel Monitor* [1].

Nella maggior parte dei *chipset* attualmente utilizzati nelle Wireless LAN, un Controllo Automatico del Guadagno (AGC) tiene sotto osservazione la potenza del segnale e adatta così il circuito RF della scheda. Questa informazione, insieme a quella relativa alla potenza del rumore, viene poi utilizzata dal driver per calcolare e memorizzare il valore in dB del rapporto segnale-rumore. La lettura della potenza del segnale e di quella del rumore avviene per ogni pacchetto ricevuto dalla macchina.

Esiste una forte correlazione tra il rapporto segnale-rumore (SNR) e la probabilità di errore sul bit, che si traduce in una correlazione altrettanto significativa fra l'SNR misurato dal driver e la probabilità di ricevere una trama con errori (*FER*, ovvero *Frame Error Rate*). Come visto nel primo capitolo, al degrado della qualità del canale corrisponde una diminuzione del goodput raggiungibile su tale canale.

Uno stimatore ideale dovrebbe tener conto del fatto che per la trasmissione di ogni pacchetto dall'Access Point ad una stazione mobile, nei casi in cui essa va a buon fine, l'AP deve trasmettere una trama e ricevere il relativo ACK: pertanto, è necessario che non si verifichino errori in entrambi i versi della comunicazione. Ne risulta che la qualità del canale è una combinazione dell'SNR della trama (misurato quindi dal ricevitore della destinazione mobile) e di quello dell'ACK (misurato dall'AP), e solo tenendo conto di entrambi è possibile una stima accurata del canale tramite la lettura dell'SNR. Poiché tuttavia in questa sede è necessario innanzitutto verificare la funzionalità di base del sistema nel suo complesso, non è necessario soffermarsi troppo su tecniche avanzate per la stima del canale, ma si può affrontare il problema con un primo approccio semplificativo.

Il cosiddetto *Wireless Channel Monitor* fa l'ipotesi di **simmetria** del canale: in altre parole, l'assunto è che l'SNR visto dall'Access Point non sia molto diverso da quello visto dal ricevitore posto sulla stazione mobile. Perciò si può supporre che l'Access Point, conoscendo il rapporto segnale-rumore delle sole **trame ricevute** da una determinata stazione mobile, possa stimare con sufficiente approssimazione il goodput che deve aspettarsi trasmettendo verso quella stazione mobile.

In questo tipo di approccio, il driver, dopo aver calcolato l'SNR, passa questa informazione al *Wireless Channel Monitor*. Quest'ultimo, tramite una tabella, stima il goodput con cui l'Access Point si aspetta di poter trasmettere verso tale stazione mobile. Il *Wireless Channel Monitor* viene poi chiamato dallo scheduler ogni volta che esso ritiene necessario conoscere la capacità del canale verso una determinata stazione mobile.

2.2.2 Calibrazione del *Wireless Channel Monitor*

Le schede di rete e i driver non sono tutti uguali: benché debbano rispettare lo standard 802.11b, case costruttrici diverse implementano lo standard in modo differente, sia nell'hardware che per quanto riguarda il software. Può accadere che uno stesso driver supporti diversi hardware, o viceversa che uno stesso hardware possa essere gestito da diversi driver [12].

In particolare, per quanto riguarda lo scheduling, è importante notare come il valore numerico della qualità del canale di trasmissione possa variare da un costruttore all'altro. Infatti, può accadere che hardware diversi abbiano sensibilità diverse: perciò schede diverse possono avere diversa capacità trasmissiva a parità di SNR. Inoltre, alcuni costruttori (ad esempio Raylink) non calcolano il rapporto segnale-rumore ma forniscono solo l'informazione sulla potenza del segnale in dBm.

Pertanto si rende necessaria una **calibrazione** del *Wireless Channel Monitor*.

La calibrazione andrebbe poi ripetuta periodicamente, in quanto la re-

tasto sinistro del mouse, facendo poi clic col tasto centrale (o con sinistro e destro contemporaneamente, nel caso si usi il touchpad) all'interno della sessione `gnuplot`).

I file da graficare, in questo caso `escher_twhtb` e `russell_twhtb`, hanno la seguente struttura: due colonne, la prima con il valore della qualità in dB (col segno meno, in modo da avere qualità decrescente sull'asse orizzontale), la seconda con i bit/s ricevuti dalla stazione mobile considerata. Esempio (`escher_twhtb`):

```
-18     4011500
-12     3578900
-8      3669100
-4      3046625
```

Per produrre i dati da inserire nei file, le tecniche sono le seguenti:

• **dati ricevuti:** si contano le righe dei file ottenuti mettendo le stazioni mobili in ricezione con MGEN (con il comando `cr < nome_misura`, dove `cr` è il file eseguibile prodotto dalla compilazione di `contarighe.c`, listato nell'appendice B) in corrispondenza di una determinata qualità per MS2, e si moltiplica per 100 (grazie al fatto che la misura dura 81.329 s per pacchetti da 1024 byte); se sono più file, se ne fa una media;

• **dati trasmessi:** si legge il rapporto del Traffic Control che, alla fine di ogni misura, è opportuno salvare in un file con un comando del tipo `tc -s -d class ls dev wlan1 parent 1:`

• **risorse consumate:** dati i bit/s ricevuti dalle stazioni in posizione buona (dovrebbero coincidere con quelli trasmessi, dato che in posizione buona non ci sono perdite), dalla loro somma si ricava il goodput massimo ottenibile sul canale; la percentuale di risorse usate da MS1 (in posizione sempre buona) è semplicemente il rapporto fra il goodput di MS1 e il goodput massimo; la percentuale di risorse usate da MS2 è la differenza fra il 100% e le risorse utilizzate da MS1.

separate `escher/` e `russell/`. Il comando è:
`scp 192.168.32.2:percorso/* escher/` (analogamente per Russell)
Bisogna poi eliminare le **prime due righe** (`LISTEN` e `START`) da ciascuno di questi file, in modo che l'istante $t = 0$ coincida con la ricezione del primo pacchetto UDP (che avviene circa contemporaneamente per le due stazioni mobili).

A questo punto, si possono fare due tipi di grafici: quelli sullo **scenario complessivo** del goodput medio, con la qualità di MS2 che varia da buona a pessima (o fuori portata); quelli sulle singole misure, **nel tempo**, con la variazione del goodput e della stima fornita dallo stimatore di canale.

Curve prestazioni-qualità Il primo tipo di grafici rappresenta il goodput medio per le due stazioni al variare della qualità del canale wireless.

I comandi `gnuplot` necessari per produrre questo tipo di grafico sono i seguenti, raccolti nei file del tipo `scenario/*.plt` (si riporta qui il file `TWHTB_adattativo_4-1.plt`):

```
set label "Buona" at -18.6, -100000
set label "Media" at -12.6, -100000
set label "Cattiva" at -8.6, -100000
set label "Pessima" at -4.6, -100000
set noxtics
set title "Scheduler TWHTB 4:1 e rate adattativo"
set xlabel "Posizione MS2" 0, -0.8
set ylabel "Goodput medio [bit/s]"
set t po eps enh c so
set o "TWHTB_adattativo_4-1.eps"
plot [-20:-2] "escher_twhtb" t "MS1" w linesp, "russell_twhtb" t "MS2" w linesp
set t x11
quit
```

Volendo evitare di digitare a mano tutti i comandi, uno per uno, si può eseguire il *copia e incolla* selezionando tutto il contenuto del file `*.plt` col

lazione che sussiste fra qualità e goodput dipende anche, seppur in misura minore, dalle caratteristiche dell'ambiente in cui si trova l'Access Point, in particolare dalla presenza e dal numero di cammini multipli (*multipath*), dalla quantità di interferenze e dalla loro variabilità temporale. La calibrazione va ripetuta anche quando varia la lunghezza media dei pacchetti trasmessi e quando si utilizza una diversa tecnica di modulazione adattativa.

Per la calibrazione, è stato usato il *Wireless Channel Monitor Signal-to-Goodput Mapper* (*WchmonSigMap*) [1] : si tratta di un'applicazione Java per Linux che ha il compito di costruire, in base a misure sperimentali, una tabella che associa a ciascun valore di SNR la corrispondente capacità. Avendo due host A e B, collegati da un link wireless, si invia da A a B traffico UDP, in quantità sufficiente per saturare completamente la capacità del link: in questo modo, A ha sempre abbastanza pacchetti IP che attendono di essere inviati, perciò il livello MAC **trasmette con continuità**. Il *WchmonSigMap*, installato su B, misura il tempo τ impiegato per ricevere un certo numero n di pacchetti (ad esempio 100), e conoscendone la lunghezza L riesce a fornire una stima del goodput:

$$\hat{R} = \frac{n \cdot L}{\tau}$$

Associando tale misura al livello di SNR contemporaneamente letto dal driver, il *WchmonSigMap* ottiene una coppia di valori (SNR, goodput) ogni n pacchetti ricevuti. Da queste coppie esso ricava poi una tabella, in cui ad ogni valore di SNR corrisponde la media dei valori di goodput misurati in corrispondenza di quel determinato SNR.

2.2.3 Problemi implementativi e risultati della calibrazione

A causa di asimmetrie nel canale di trasmissione wireless, la calibrazione con pacchetti trasmessi dall'AP alla MS ha dato un risultato leggermente diverso dalla calibrazione effettuata trasmettendo i pacchetti in direzione opposta.

Bisogna tenere conto infatti che la trasmissione di ogni pacchetto, nei casi in cui va a buon fine, si compone in realtà di due fasi: la trasmissione della trama e la ricezione di un Acknowledgement (ACK). Quando il canale non è simmetrico, si può verificare il caso in cui la trama vera e propria trova un SNR abbastanza buono, mentre l'ACK ha più probabilità di essere corrotto dal rumore perché trova un canale cattivo (dovuto alla maggior presenza di rumore al trasmettitore della trama, cioè al ricevitore dell'ACK). Oppure, viceversa, la trama viene trasmessa con difficoltà mentre l'ACK trova un SNR migliore e viaggia senza difficoltà.

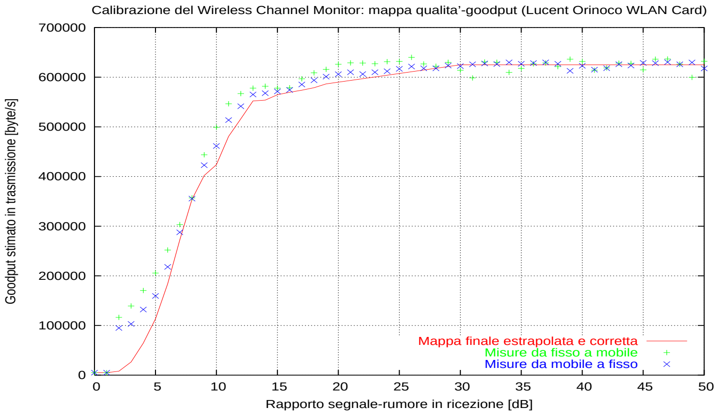

Figura 2.2: Risultati della calibrazione del *Wireless Channel Monitor*.

È stata fatta pertanto una calibrazione in entrambi i sensi e la tabella finale, da utilizzare poi per lo scheduling, è stata ricavata tramite operazioni di estrapolazione e *smoothing* (figura 2.2).

Data l'alta variabilità dell'SNR istantaneo letto dal driver (che si riferisce alla singola trama ricevuta e quindi risente di fluttuazioni a breve termine della potenza del segnale), è stato necessario introdurre nel *Wireless Channel Monitor* un filtro IIR passa-basso, che facesse dipendere il valore istantaneo

6 • Generazione dei pacchetti Il PC che genera i pacchetti deve innanzitutto sapere chi è il router in grado di inoltrarli verso la destinazione corretta, perciò prima di avviare MGEN si deve inserire nella tabella di routing di tale PC, se non già presente, una riga che dica che tutti i pacchetti indirizzati verso la rete wireless vanno inviati all'AP tramite la LAN (presumibilmente una Ethernet) che collega i due PC fissi. Il comando è il seguente:

```
route add -net 192.168.32.0/24 gw 131.114.53.96
```

Poi bisogna avviare MGEN con uno script di input:

```
./mgen input wischhof.mgn
```

dove il file `wischhof.mgn` conterrà, ad esempio, le seguenti righe:

```
0.000 ON 2 UDP DST 192.168.32.2/20000 PERIODIC [750 1024]
0.000 ON 4 UDP DST 192.168.32.4/40000 PERIODIC [750 1024]
81.329 OFF 4
81.329 OFF 2
```

In questo caso MGEN invia verso entrambe le destinazioni (192.168.32.2, corrispondente a MS1, e 192.168.32.4, corrispondente a MS2) 6 Mbit/s di traffico (750 pacchetti da 1024 byte ogni secondo) per una durata di 81.329 secondi. La durata è stata scelta apposta affinché il numero di pacchetti ricevuti da ciascuna destinazione alla fine della misura sia numericamente equivalente a 10 volte il goodput medio in Kbit/s: questo aiuta a capire in modo intuitivo, senza fare troppi calcoli, quanti dati sono stati trasmessi e ricevuti, e quindi trarre rapidamente qualche conclusione sul risultato della misura svolta.

Estrazione ed elaborazione dei dati misurati

Innanzituto può essere utile copiare i file contenenti i dati sulla ricezione dei pacchetti dalle due stazioni mobili al router, ad esempio in due cartelle

```
tc class add dev $ETH parent 1:0 classid 1:1 htb rate 5mbit burst 15k cburst 150
k
tc class add dev $ETH parent 1:1 classid 1:2 htb rate 4mbit burst 15k cburst 150
k
tc class add dev $ETH parent 1:1 classid 1:4 htb rate 1mbit burst 15k cburst 150
k
tc filter add dev $ETH parent 1:0 protocol ip prio 1 u32 \
        match ip dst 192.168.32.2 flowid 1:2
tc filter add dev $ETH parent 1:0 protocol ip prio 1 u32 \
        match ip dst 192.168.32.4 flowid 1:4

################################################################################

### drrpari (DRR/[T]WHTB con 2.5 Mbit/s sia verso 192.168.32.2 che verso 192.168.32.4) ##
#!/bin/sh
ETH=$1
tc qdisc del dev $ETH root
rmmod sch_htb
rmmod wchmon_driver
insmod wchmon_driver module_debug=3
cp /home/valori/cf/autosched-finale /proc/net/wchmon_gtr_map
tc qdisc add dev $ETH root handle 1:0 htb default 4 debug 0000120000200000
tc class add dev $ETH parent 1:0 classid 1:1 htb rate 5mbit burst 15k cburst 150
k
tc class add dev $ETH parent 1:1 classid 1:2 htb rate 2.5mbit burst 15k cburst 1
50k
tc class add dev $ETH parent 1:1 classid 1:4 htb rate 2.5mbit burst 15k cburst 1
50k
tc filter add dev $ETH parent 1:0 protocol ip prio 1 u32 \
        match ip dst 192.168.32.2 flowid 1:2
tc filter add dev $ETH parent 1:0 protocol ip prio 1 u32 \
        match ip dst 192.168.32.4 flowid 1:4
```


del goodput stimato non solo dall'SNR attuale ma anche da tutti i precedenti (naturalmente privilegiando i valori letti più recentemente):

$$\hat{R}(t) = \frac{3}{4} \cdot \hat{R}(t-1) + \frac{1}{4} \cdot \hat{R}[SNR(t)]$$

dove $\hat{R}[SNR(t)]$ è la stima del goodput, letta dalla tabella ottenuta come risultato della calibrazione, in corrispondenza di $SNR(t)$.

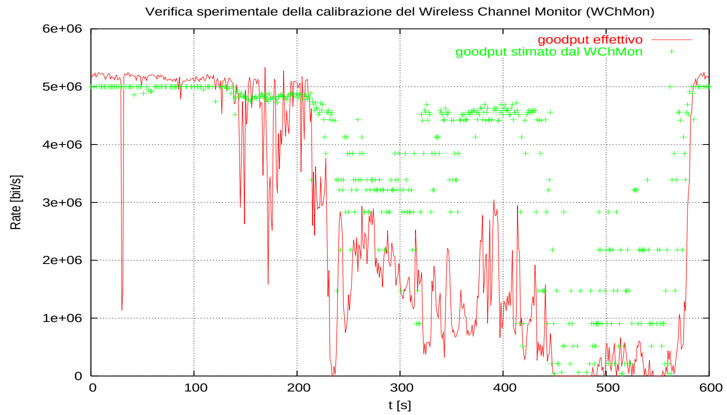

Figura 2.3: Confronto fra goodput effettivo e stima del *Wireless Channel Monitor*.

La figura 2.3 mostra come il *Wireless Channel Monitor*, una volta calibrato, basandosi solo sul valore del rapporto segnale-rumore delle trame ricevute da una determinata destinazione, fornisca un'approssimazione del goodput ottenibile verso tale destinazione.

2.3 Funzionamento dello scheduler HTB

Per utilizzare le informazioni provenienti dallo stimatore di canale in modo da ridurre il degrado delle prestazioni dovuto all'anomalia dell'802.11b, si è

scelto di modificare in questo senso lo scheduler HTB, che presenta una struttura relativamente semplice rispetto ad altri scheduler (quale il CBQ oppure l'H-FSC) dotati di analoga flessibilità nell'utilizzo delle classi di traffico.

2.3.1 Parametri principali dell'HTB

Lo scheduler **HTB** (***Hierarchical Token Bucket***) organizza le classi di traffico secondo una struttura ad albero. La figura 2.4 mostra come, nel nostro caso, si utilizzino due classi *leaf* (1:2 e 1:4) associate alle due diverse destinazioni, ed entrambe dipendano dalla classe *root* (1:1), legata direttamente alla disciplina di coda HTB.

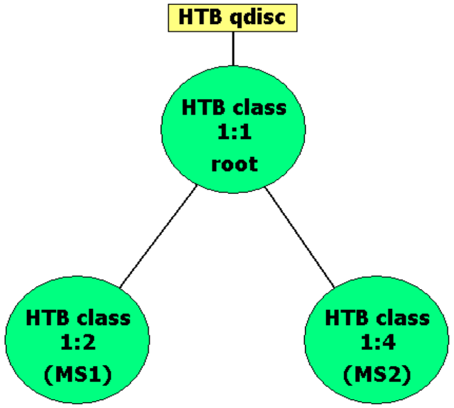

Figura 2.4: Classi di traffico utilizzate per lo scheduling su Wireless LAN.

A ciascuna classe viene assegnato un rate medio da garantire (parametro *rate*) ed un rate massimo da non superare mai (parametro *ceil*). Ogni classe, al suo interno, è regolata da un **token bucket**, da cui il nome dello scheduler. Il bucket viene riempito con un numero di token proporzionale ai byte che la classe ha il diritto di trasmettere, quindi proporzionale al prodotto fra tempo


```
### cbq (CBQ con 4 Mbit/s verso 192.168.32.2 e 1 Mbit/s verso 192.168.32.4) ###
#!/bin/sh
ETH=$1
tc qdisc del dev $ETH root
rmmod sch_cbq
rmmod wchmon_driver
insmod wchmon_driver module_debug=3
cp /home/valori/cf/autosched-finale /proc/net/wchmon_gtr_map
tc qdisc add dev $ETH root handle 1:0 cbq bandwidth 5mbit avpkt 1052\
        mpu 64 cell 8
tc class add dev $ETH parent 1:0 classid 1:1 cbq bandwidth 5mbit rate 5mbit\
        allot 1514 cell 8 prio 1 maxburst 20 avpkt 1052 bounded isolated
tc class add dev $ETH parent 1:1 classid 1:2 cbq bandwidth 5mbit rate 4mbit\
        allot 1514 cell 8 prio 2 maxburst 20 avpkt 1052 bounded
tc class add dev $ETH parent 1:1 classid 1:4 cbq bandwidth 5mbit rate 1mbit\
        allot 1514 cell 8 prio 2 maxburst 20 avpkt 1052 bounded
tc filter add dev $ETH parent 1:0 protocol ip prio 1 u32 \
        match ip dst 192.168.32.2 flowid 1:2
tc filter add dev $ETH parent 1:0 protocol ip prio 1 u32 \
        match ip dst 192.168.32.4 flowid 1:4

############################################################################

### drr (DRR/[T]WHTB con 4 Mbit/s verso 192.168.32.2 e 1 Mbit/s verso 192.168.32.4) ##
#!/bin/sh
ETH=$1
tc qdisc del dev $ETH root
rmmod sch_htb
rmmod wchmon_driver
insmod wchmon_driver module_debug=3
cp /home/valori/cf/autosched-finale /proc/net/wchmon_gtr_map
tc qdisc add dev $ETH root handle 1:0 htb default 4 debug 0000120000200000
```

considera non raggiungibile e non passa il pacchetto al proprio MAC layer. Nelle prove prestazionali di questa tesi, onde evitare che MS2 sia considerata irraggiungibile quando si trova in posizione pessima, si inserisce una entry statica nell'ARP cache, con il seguente comando:

```
arp -s 192.168.32.4 00:02:2D:27:43:6F
```

4 • Utilizzo del syslog Tutte le informazioni di debug, prodotte dai moduli del kernel con l'istruzione in linguaggio C `printk (KERN_DEBUG "...");`, vengono registrate nel file `/var/log/debug`. Poiché non ci interessa tutto il contenuto del file (che spesso è di svariati MB) ma solo ciò che è stato scritto durante la misura, si isola tale contenuto con i seguenti 4 passi:
`rm /var/log/debug` per eliminare tutto ciò che è stato scritto da $t = -\infty$ all'inizio della misura;
`kill -SIGHUP 227` (appena prima di iniziare la misura) per creare di nuovo il file e far ripartire il demone del syslog, in modo che i successivi messaggi di debug vengano scritti lì; per sapere qual è il processo da far ripartire, serve il comando `ps aux | grep syslogd`;
`mv /var/log/debug nome_cartella/debug_file` (appena finita la misura) per salvare il file in un posto sicuro;
`kill -SIGHUP 227` per far scrivere le future informazioni di debug (ormai inutili, dato che la misura è finita) non più sul file salvato, ma di nuovo su `/var/log/debug`.

5 • Configurazione del Traffic Control Indicazioni più dettagliate si possono trovare in [13], qui riportiamo solo, a titolo di esempio, gli script `cbq`, `drr` e `drrpari` utilizzati per il WHTB. Per indicare l'interfaccia di rete wireless su cui deve funzionare lo scheduler, se ne deve passare il nome come parametro.
Esempio: `drr wlan1` (attiva lo scheduler DRR/WHTB/TWHTB sull'interfaccia `wlan1` con 4 Mbit/s alla classe 1:2 e 1 Mbit/s alla classe 1:4)

trascorso e rate assegnato. Dal bucket vengono poi tolti i token corrispondenti ai byte trasmessi quando i pacchetti della classe vengono estratti dalla coda dello scheduler ed inviati agli strati inferiori dello stack protocollare. Poiché i rate assegnati a ciascuna classe sono due, in generale distinti, ogni classe ha bisogno di due token bucket: nel bucket associato al *rate* abbiamo i cosiddetti *tokens*, mentre in quello associato al *ceil* vengono immessi i cosiddetti *ctokens*.

Durante l'effettiva trasmissione dei pacchetti appartenenti alle diverse classi di traffico, ciascuna classe può trovarsi in un dato istante in tre possibili condizioni, convenzionalmente associate a tre colori:
• **verde:** la classe ha trasmesso meno di quanto abbia diritto a fare, perciò può trasmettere liberamente;
• **gialla:** la classe ha superato il *rate*, ma non il *ceil*, perciò può prendere in prestito la banda eventualmente non utilizzata da altre classi;
• **rossa:** la classe ha superato il tetto massimo del rate, perciò non può trasmettere.

Nei casi che sono oggetto di questa tesi, poiché ci si trova in una situazione di scarsità di banda, nessuna classe ha la possibilità di prestarne all'altra, quindi è opportuno che i parametri *rate* e *ceil* coincidano. Una classe quindi potrà essere, in un dato istante, solo verde o rossa: la discriminazione verrà fatta solo in base alla quantità di *ctokens* ancora a disposizione (mentre i *tokens* non costituiranno mai un fattore limitante).

2.3.2 Meccanismi di gestione delle classi nel caso in cui i parametri *rate* e *ceil* siano uguali

L'obiettivo fondamentale dello scheduler HTB è il seguente [3]: far sì che, in presenza di pacchetti da inviare, **ciascuna classe ottenga almeno il suo *rate*, ma non ecceda il *ceil***. Nel caso di nostro interesse, ciò si riduce ad ottenere esattamente il rate (medio e massimo) assegnato. Per raggiungere quest'obiettivo, lo scheduler HTB dà il diritto a trasmettere alle **sole classi**

verdi, secondo un algoritmo di tipo DRR (*Deficit Round Robin*).

Lo scheduler estrae dalla coda, uno per volta, i pacchetti appartenenti ad una certa classe. Ogni volta che un pacchetto viene trasmesso, il *deficit* viene decrementato della lunghezza *length* del pacchetto stesso; inoltre, il valore di *ctokens* viene decrementato di una quantità pari al rapporto fra la lunghezza del pacchetto ed il *ceil*, mentre viene incrementato di un valore pari al tempo trascorso dall'ultimo pacchetto inviato dalla classe.

I possibili motivi per cui lo scheduler decide di interrompere una classe, per assegnare ad un'eventuale altra il diritto a trasmettere, sono due:
• **Esaurimento del *deficit*:** quando il *deficit* viene decrementato, può accadere che si ottenga un risultato negativo. In tal caso il *deficit* viene aumentato di una quantità, detta *quantum*, proporzionale al *rate* (di default $\frac{1}{10}$ del *rate*), ma il diritto di trasmettere va alla successiva classe verde. Come caso particolare, se non ci sono altre classi verdi, la classe attualmente attiva può continuare a trasmettere i propri pacchetti.
• **Bucket underflow:** lo scheduler si accorge che la classe attualmente attiva sta superando il *ceil* dal fatto che nel token bucket di tale classe non ci sono abbastanza *ctokens*. L'intervallo di variazione di *ctokens* dipende da un parametro iniziale, il *cburst*. Il valore di *ctokens* deve essere sempre compreso fra *-cburst* e *cburst*: se scende sotto *-cburst*, si ha un underflow e la classe diventa rossa; se sale oltre *cburst*, tale valore viene semplicemente limitato superiormente (ovvero i *ctokens* in eccesso vengono buttati via).

Si può osservare come l'underflow sia prioritario rispetto al deficit: se si verifica underflow, infatti, indipendentemente dal deficit la classe viene interrotta; viceversa, se si esaurisce il deficit ma le altre classi sono ancora rosse, lo scheduler continua a far trasmettere la classe attiva (si limita ad aggiungerle un *quantum* e a proseguire).

sibili posizioni delle stazioni mobili:
• **buona:** il più possibile vicino all'Access Point (fra la scheda WLAN dell'AP e quella della MS non dovrebbero esserci più di un paio di metri in linea d'aria), con `iwspy` che indica una qualità intorno ai 30 dB o superiore;
• **media:** assenza di alterazioni nei ritardi del `ping` rispetto alla posizione buona (al più qualche ritardo superiore ai 2 ms, ma ogni tanto), qualità 10-12 dB segnalata da `iwspy`;
• **cattiva:** alterazioni, anche numerose, nei ritardi del `ping`, ma ancora senza perdite (se il rate è adattativo), qualità che si mantiene quasi sempre sotto i 10 dB, ma raramente scende sotto i 4 dB;
• **pessima:** distanza notevole (oltre 10 m, ma a volte si arriva anche a 30 m dall'Access Point), situazione limite tra ricezione e non ricezione dei dati, ritardi del `ping` regolarmente sopra 2 ms, alcuni `ping` senza risposta, qualità compresa fra -1 (`iwspy` segnalerà un poco credibile 255) e 6 dB;
• **fuori portata:** ci si potrebbe allontanare di 50 m, ma in realtà è sufficiente estrarre l'interfaccia WLAN dallo slot PCMCIA della MS ;-)

2 • Abilitazione del forwarding IP Un PC normalmente è un host, non un router, pertanto non è abilitato ad inoltrare i pacchetti IP non destinati ad esso. Poiché invece in questo testbed esiste un PC che funge da router, bisogna abilitare in esso il forwarding IP, con il comando

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

3 • ARP cache statica Nel comportamento normale di una LAN, ed in particolare di una Wireless LAN, ogni stazione mantiene una *ARP cache*, ovvero una tabella di corrispondenza tra indirizzi IP ed indirizzi MAC che viene aggiornata dinamicamente. Se una stazione vuole inviare un pacchetto verso un indirizzo IP e questo non è presente nella cache, essa deve prima mandare una *ARP request* ed aspettare che la destinazione risponda col suo indirizzo MAC. Se la destinazione non risponde, la stazione trasmettitrice la

qualsiasi, ad esempio `iproute2-2.2.4-now-ss001007.tar.gz`

• **Patch dell'HTB per il Traffic Control:** dal sito *HTB Home* [14], `htb3.6-020525.tgz`

• **Generatore di pacchetti MGEN:** dal sito del Naval Research Laboratory [18], `linux-mgen-4.0x5.tgz`

• **Wireless Tools:** dal sito di Jean Tourrilhes [12], `wireless_tools.26.tar.gz`

• **Codice sorgente del driver *HostAP*:** dal sito ufficiale dell'*HostAP* [16], `hostap-driver-0.2.0.tar.gz`

• **PCMCIA Card Services:** dal sito ufficiale su Sourceforge [7], `pcmcia-cs-3.2.7.tar.gz`

• **Patch per il Wireless Scheduling:** dal sito del progetto `wsched` [1], `wsched-0.5.2.tar.gz`

Sul PC che genera il traffico è sufficiente che siano installati Linux e MGEN.

Svolgimento delle prove prestazionali

1 • Predisposizione delle stazioni mobili MS1 e MS2 devono essere predisposte alla ricezione dei pacchetti. Per questo si usa il ricevitore di MGEN (o, in alternativa, TCPdump). Nel caso che si usi MGEN, il comando è:

`./mgen input receiver.mgn output nome_misura`

dove il file di input `receiver.mgn` indica su quale porta UDP il ricevitore deve mettersi in ascolto (per ricevere solo i pacchetti MGEN ad esso destinati):

`0.0 LISTEN UDP 20000` (riceve i pacchetti UDP sulla porta 20000)

Per monitorare manualmente la qualità del canale, si può dare il comando `iwspy` ogni uno/due secondi sulla MS, eventualmente dopo aver lanciato un `ping` verso l'AP (perché `iwspy` aggiorna il valore della qualità solo quando riceve una nuova trama) e leggere il parametro `Quality` o la differenza tra `Level` e `Noise`; ecco inoltre qualche suggerimento per distinguere fra le pos-

2.3.3 Casi pratici di scheduling HTB (non modificato) su Wireless LAN

Canale di buona qualità per entrambe le stazioni mobili. Quando le due stazioni mobili sono in posizione **buona**, cioè tale da rendere trascurabili gli errori sui bit e quindi le ritrasmissioni, ciascuna di esse, se fosse da sola, riceverebbe il massimo goodput possibile. Poiché nelle misure effettuate utilizziamo sempre pacchetti da 1024 byte, questo goodput risulta essere di circa 5 Mbit/s (come accennato a pag. 12). Come possiamo aspettarci da qualsiasi scheduler di tipo *classful*, la banda viene suddivisa fra le due classi secondo i rate assegnati (figure 2.5 e 2.6).

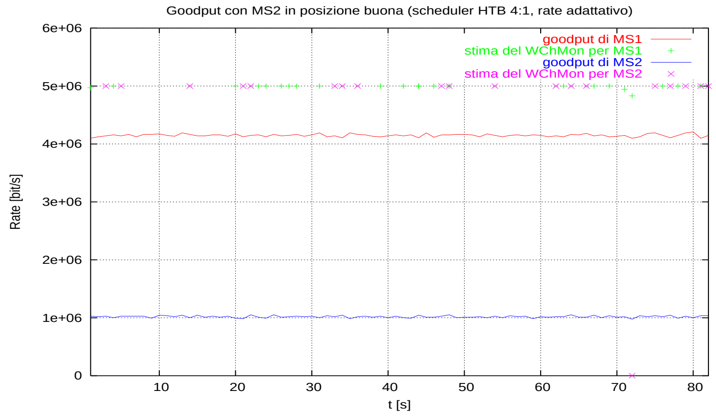

Figura 2.5: Comportamento nel tempo dello scheduler HTB, con 4 Mbit/s assegnati a MS1 e 1 Mbit/s assegnato a MS2, qualità buona per entrambe le stazioni mobili.

Il meccanismo che interviene in questo caso è il secondo (bucket underflow). Essendo il canale buono, il tempo tra un pacchetto e l'altro (*intertempo*) è pressoché costante e pari a circa 1.6 ms (varia solo il backoff, come

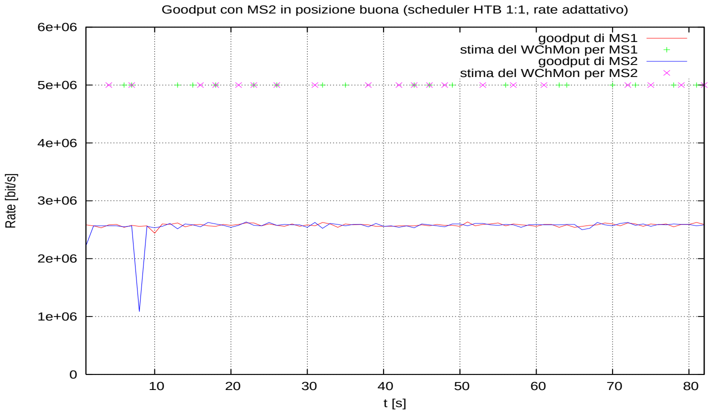

Figura 2.6: Comportamento nel tempo dello scheduler HTB, con 2.5 Mbit/s assegnati a ciascuna delle due stazioni mobili, entrambe in buona posizione.

mostrato nel capitolo 4 e in [5]). Ogni volta che viene estratto un pacchetto dalla coda, questo tempo viene sommato ai *ctokens*, sottraendo invece il rapporto fra la lunghezza del pacchetto ed il rate assegnato alla classe:

$$ctokens(t_2) = ctokens(t_1) + (t_2 - t_1) - \frac{length}{rate}$$

Dal momento che $rate < R_{MAX} = R$, dunque $\frac{length}{rate} > \frac{length}{R}$, la quantità sottratta è maggiore di quella sommata (l'intertempo $t_2 - t_1$ corrisponde infatti al tempo impiegato per trasmettere un pacchetto di lunghezza *length* utilizzando tutta la capacità R del link), perciò i *ctokens* diminuiscono. La diminuzione dei *ctokens* prosegue fino al bucket underflow: a quel punto, poiché non ci sono abbastanza *ctokens*, la classe diventa rossa e il diritto di trasmettere spetta all'altra classe. Quando invece una classe esaurisce il *deficit*, non perde il diritto di trasmettere (tabella 2.1) perché l'altra è rossa: in ogni istante, infatti, c'è una ed una sola classe verde, in quanto entrambe le classi stanno sfruttando al 100% la capacità trasmissiva assegnata loro

Appendice A

Realizzazione pratica del testbed

Installazione del testbed

Sono stati usati 4 calcolatori (2 desktop e 2 laptop) basati su processori Intel:

- *Granpa*: generatore di traffico, desktop Pentium III.
- *Edison*: Access Point (o, più propriamente, router IP), desktop Pentium III.
- *Escher*: MS1, stazione mobile in posizione costante buona, laptop Pentium II.
- *Russell*: MS2, stazione mobile in posizione tale da rendere variabile la qualità del canale radio, laptop Pentium II.

Sui calcolatori della rete wireless (tutti tranne il generatore di traffico) viene installato Linux (nel nostro caso Debian) con i seguenti pacchetti, scaricati da Internet o da CD-ROM e poi compilati e/o installati:

• **Codice sorgente di Linux:** nel nostro caso il kernel 2.4.24 (alcune prove sono state eseguite con il 2.4.20, ma non dovrebbe cambiare molto), quindi `linux-2.4.24.tar.gz`

• **Codice sorgente del Traffic Control di Linux:** va bene una versione

classe 1:2 (MS1)			classe 1:4 (MS2)		
deficit	Δt	*ctokens*	*deficit*	Δt	*ctokens*
770	1585	-17764			
32472	1447	**-20756**			
Bucket underflow →			5132	1830	2364
			4066	1599	-206
			3000	1497	-3148
			1934	1479	-6108
			868	1458	-9089
			32570	1484	-12044
			31504	1386	-15097
			30438	1393	-18143
			29372	1473	**-21109**
31406	1563	4043	← **Bucket underflow**		
30340	1495	1099			
29274	1468	-1872			
28208	...				

Tabella 2.1: Esempio di HTB con due canali buoni.

(quindi se c'è una classe verde, cioè sotto il *ceil*, l'altra è necessariamente sopra, perciò rossa).

Canale buono per una stazione mobile (MS1) e cattivo per l'altra (MS2). Le figure 2.7, 2.8 e 2.9 sembrerebbero indicare una certa differenza nel trattamento di classi corrispondenti a canali trasmissivi di qualità diversa. La figura 2.7 mostra un rapporto tra i goodput medi raggiunti dalle due

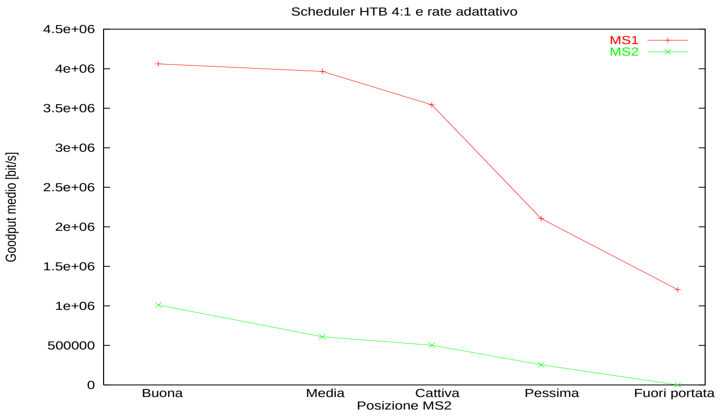

Figura 2.7: Curva prestazioni-qualità dello scheduler HTB con i parametri di default. Alla classe associata a MS1 (posizione costante buona) sono assegnati 4 Mbit/s, a MS2 (posizione variabile) 1 Mbit/s.

classi, in caso di canale cattivo per MS2, di 8:1 al posto del 4:1 assegnato: un rapporto che **penalizza la classe associata ad MS2**, la stazione mobile col link di qualità peggiore. Anche le figure 2.8 e 2.9 evidenziano il trattamento differente che subiscono le due classi, cui in questo caso viene assegnato lo stesso rate, pertanto sembra che lo scheduler HTB tenga conto dell'unica differenza fra esse, ovvero la qualità del canale: anche qui, **ad essere penalizzata è la classe associata ad MS2**, che riceve la metà dei

(pag. 84) può inoltre aprire la strada per lo sviluppo di un nuovo scheduler che tenga sempre vuota la coda a livello MAC, servendosi delle informazioni, fornite dal driver dell'interfaccia WLAN, sull'istante in cui si ha l'effettiva conclusione della trasmissione di un pacchetto nella rete.

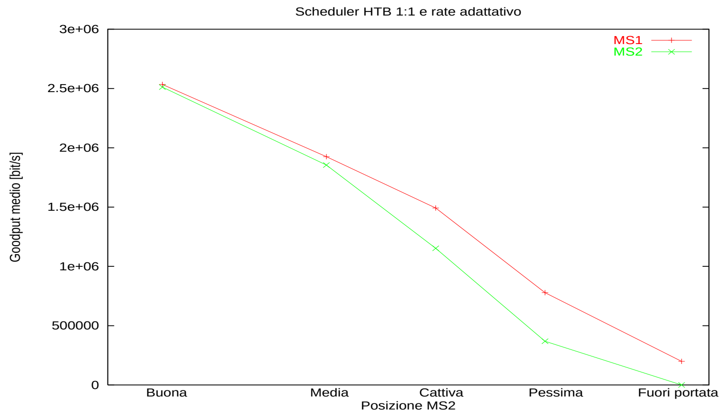

Figura 2.8: Curva prestazioni-qualità dello scheduler HTB, con 2.5 Mbit/s assegnati a ciascuna delle due stazioni mobili.

pacchetti rispetto alla stazione mobile in buona posizione. Tuttavia le cause di questo comportamento non sono legate direttamente alla conoscenza della qualità del canale, nota al driver ma non allo scheduler.

In questo caso il tempo di trasmissione di un pacchetto è **variabile**, oltre che **mediamente più lungo** (si veda la sezione 4.1 e in particolare la tabella 4.3). L'istante in cui lo scheduler viene avvertito che è possibile passare un altro pacchetto agli strati inferiori non coincide però con l'istante in cui il pacchetto precedente è stato definitivamente inviato a destinazione dal livello MAC. Esiste infatti una **coda a livello 2**, non accessibile allo scheduler, dove i pacchetti estratti dallo scheduler vengono accodati prima di essere fisicamente inviati.

Nel caso di scheduling verso due destinazioni con diversi SNR, è stato verificato sperimentalmente che, in presenza di una destinazione con SNR molto basso, con i parametri di default dell'HTB (in particolare *cburst*) gli intertempi più alti (in *corsivo* nella tabella 2.2) vengono osservati fra i pacchetti

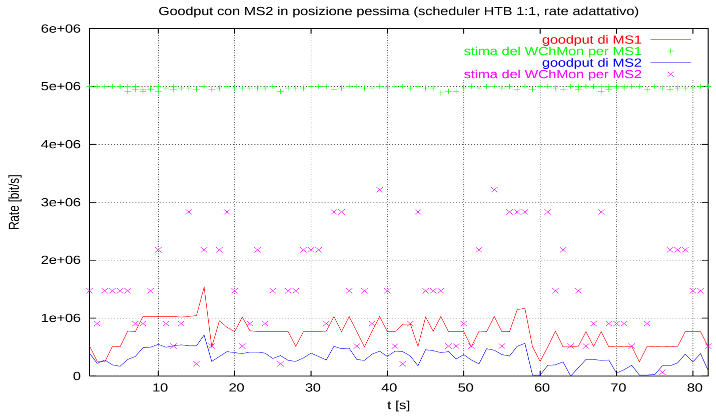

Figura 2.9: Comportamento nel tempo dello scheduler HTB, con 2.5 Mbit/s assegnati a ciascuna delle due stazioni mobili (MS1 in posizione buona e MS2 in posizione pessima).

come nel caso degli scheduler tradizionali. Si tratta probabilmente di un problema di calibrazione del *Wireless Channel Monitor*, dovuto all'asimmetria del canale di trasmissione: dato che tale asimmetria varia nel tempo, è possibile che le prestazioni migliorino nel caso che si esegua ogni volta una nuova calibrazione poco prima di iniziare la trasmissione vera e propria.

- ***Wireless Channel Monitor* e TWCM con qualità pessima:** Quando una delle due stazioni si trova all'estrema periferia dell'area di copertura dell'AP, si verificano due problemi, i cui effetti si sommano:

  - Alcuni pacchetti vengono persi in quanto il numero di tentativi eccede il *Retry_Limit*; né il *Wireless Channel Monitor* né il TWCM sono ben attrezzati per fornire una stima accurata del goodput anche in tali condizioni, perciò le prestazioni ne risentono.

  - Il funzionamento del DRR (paragrafo 2.3.4, [15]) comporta che, ogni volta che si esaurisce il *deficit* di una classe, venga trasmesso almeno un pacchetto dell'altra classe, in quanto la verifica che il *deficit* non sia esaurito si fa solo per i pacchetti successivi, in quanto si suppone che serva almeno un pacchetto per esaurire il *deficit*. Il problema è che ciò si traduce in un limite superiore al rapporto fra i dati trasmessi verso la stazione in posizione buona e quelli inviati alla stazione in posizione pessima. Una possibile soluzione da implementare, se non si vuole aumentare la dimensione del *quantum* (che causerebbe unfairness nel breve periodo), sarebbe quella di far saltare uno o più turni di trasmissione ad una classe che avesse un goodput relativo particolarmente basso.

In conclusione, i risultati ottenuti mostrano la possibilità concreta di tenere conto, in fase di scheduling, delle caratteristiche del canale wireless. I limiti evidenziati indicano come possibile ulteriore sviluppo la ricerca di tecniche più avanzate per la stima della qualità del canale radio. Il lavoro svolto in questa tesi, in particolare la modifica del driver *HostAP* per il TWCM

presenza faccia diminuire il goodput ricevuto dall'altra stazione, in posizione buona.

Il superamento dell'anomalia si basa dunque sulla stima del goodput sul canale, che è stata realizzata tramite due diversi approcci: il primo (*Wireless Channel Monitor*), già sviluppato in [1], che parte dalla conoscenza dell'SNR delle trame che l'AP riceve dalle diverse destinazioni; il secondo (TWCM), che invece calcola il goodput sulla base della conoscenza degli intervalli temporali fra i pacchetti, la cui misura è data, nella pratica, dalla differenza fra gli istanti in cui l'AP riceve gli ACK delle trame da esso inviate alle stazioni mobili.

Risultati ottenuti

Nonostante questo fosse solo un primo approccio per verificare la possibilità di uno scheduling *channel-aware*, i risultati iniziali sono buoni. Sia il WHTB che il TWHTB, infatti, come mostrato nei capitoli 3 e 5, sono capaci di tenere adeguatamente conto della differenza di qualità fra le diverse destinazioni mobili, limitando il consumo di risorse di rete da parte di una stazione in cattiva posizione rispetto all'AP, che invece per gli scheduler tradizionali è il problema all'origine dell'anomalia esaminata nel capitolo 1.

Le limitazioni degli scheduler implementati sono sostanzialmente legate ad una stima del canale molto approssimativa. Il funzionamento imperfetto del WHTB è causato fondamentalmente dal *Wireless Channel Monitor*, mentre le prestazioni del TWHTB sono limitate da un comportamento non sempre appropriato del TWCM. Si possono imputare i limiti dei due stimatori qui realizzati alle seguenti cause principali:

- ***Wireless Channel Monitor* con qualità media:** Quando una delle due stazioni si trova in posizione media, il *Wireless Channel Monitor* sovrastima la qualità e considera la posizione di tale stazione come se fosse buona; questo fa sì che anche il goodput della stazione in buona posizione decresca

classe 1:2 (MS1)			classe 1:4 (MS2)		
deficit	Δt	*ctokens*	*deficit*	Δt	*ctokens*
32740	1848	15904	→ 32346	2588	15904
31674	2042	13507	31280	2065	13530
30608	2160	11228	30214	1995	11086
29542	1927	8716	29148	1987	8634
28476	3552	7829	28082	2427	6622
27410	3527	6917	27016	2468	4651
26344	*27002*	*15904*	25950	1486	1698
25278	*7865*	*15904*	24884	3634	893
24212	1533	12998	23818	2296	-1250
23146	*14372*	*15904*	22752	2433	-3256
22080	*7393*	*15904*	21686	2380	-5315
21014	2987	14452	20620	2428	-7326
19948	2960	12973	19554	1976	-9789
18882	2992	11526	18488	2603	-11625
17816	3236	10323	17422	2025	-14039
16750	3591	9475	16356	1855	-16623
15684	*6201*	*11237*	15290	2537	-18525
14618	3059	9857	14224	2049	**-20915**
13552	3328	8746	**Bucket underflow**		
12486	3626	7933			
11420	3157	6651			
10354	*5413*	*7625*			
9288	3444	6630			
8222	2257	4448			
7156	2139	2148			
6090	2482	191			
5024	2379	-1869			
3958	2488	-3820			
2892	2297	-5962			
1826	2309	-8092			
760	1900	-10631			
Esaurimento *deficit* →					

Tabella 2.2: HTB con un canale buono e uno cattivo.

inviati verso l'altra destinazione, quella con un SNR migliore. La classe 1:2, col canale di qualità migliore, riesce quasi sempre, in questo caso, ad esaurire il suo *deficit*, perché i *ctokens* che perde con intertempi corti vengono spesso compensati da intertempi più lunghi dovuti alla destinazione con SNR basso. Al contrario, la classe 1:4, col canale di qualità peggiore, pur esaurendo a volte il *deficit* (ricordiamo che gli intervalli fra un pacchetto e l'altro sono in generale più lunghi), si interrompe anche per bucket underflow, in quanto mancano intertempi lunghissimi che facciano risalire il numero dei *ctokens*.

2.3.4 Uso dello scheduler HTB come un semplice DRR

L'effetto collaterale appena descritto può essere evitato: basta disabilitare il meccanismo del bucket underflow, riducendo il funzionamento dello scheduler HTB ad un puro **DRR** (***Deficit Round Robin***).

Non è necessario modificare lo scheduler per ottenere questo risultato: è sufficiente passare all'HTB, come parametro da riga di comando, un *cburst* molto grande (nel caso in esame, 150 KB è più che sufficiente). Con un token bucket così grande, di fatto è impossibile che si verifichi un bucket underflow. Perciò le due classi restano sempre di colore verde, quindi l'unico meccanismo che interviene nello scheduling è, a questo punto, il DRR (pag. 42). Esso si limita in questo caso a **trasmettere $\frac{quantum}{length}$ pacchetti per ciascuna classe**, per poi passare all'altra.

Risultato di questa semplificazione (da HTB a DRR) è che la differenza di qualità del canale fa diminuire in ugual misura il goodput di entrambe le classi. Il rapporto torna ad essere quindi quello assegnato (rispettivamente 1:1, figura 2.11, e 4:1, figura 2.12).

Come conseguenza, l'andamento discendente che assume la curva prestazioni-qualità del DRR (figura 2.12) è molto simile a quello della curva del CBQ vista nel capitolo 1 (figure 1.12 e 2.13). Per ottenere un significativo miglio-

Conclusioni

Obiettivi e metodologia

Lo scopo di questo lavoro di tesi è stato quello di analizzare la cosiddetta **anomalia della Wireless LAN 802.11b** e superarla tramite uno **scheduler basato sulla qualità del canale radio**.

L'anomalia, descritta nel primo capitolo anche con risultati sperimentali, deriva dal fatto che gli scheduler tradizionali, sviluppati per reti cablate, non tengono conto della diversa qualità del canale verso differenti destinazioni in una stessa rete wireless. Perciò tali scheduler, pur garantendo un determinato rapporto fra le quantità di dati trasmessi alle varie destinazioni, non possono garantire lo stesso rapporto fra le risorse da esse realmente consumate.

Ciò che in questa tesi si è cercato di fare, con risultati migliorabili ma incoraggianti, è uno scheduling che garantisca alle stazioni mobili un utilizzo delle risorse dell'Access Point suddiviso fra di esse in modo controllato. Per il raggiungimento di quest'obiettivo, si è introdotto il concetto di **goodput relativo**, così da pesare in maniera diversa i dati trasmessi verso destinazioni con qualità differente: se una stazione mobile è più lontana dall'AP rispetto ad un'altra, a parità di dati a livello IP che lo scheduler (presente sull'AP) le invia, essa consuma una quantità maggiore di risorse trasmissive perché il goodput relativo risulta minore. Pertanto lo scheduler deve tenere conto di questa differenza, riducendo di conseguenza la quantità di dati trasmessi verso la stazione in posizione cattiva, in misura tale da impedire che la sua

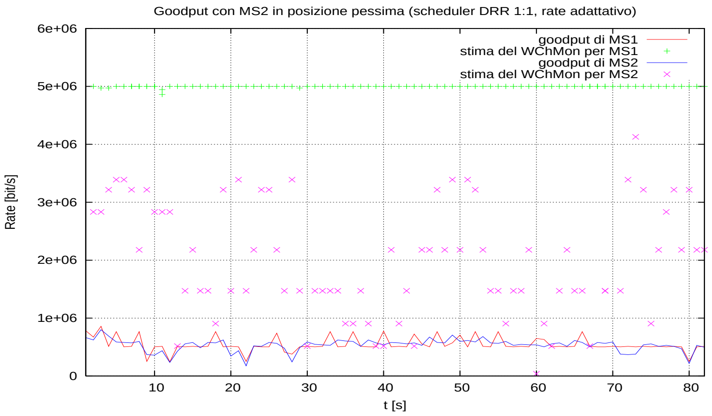

Figura 2.10: Comportamento nel tempo dello scheduler HTB quando il *cburst* è tanto alto da far degenerare l'HTB in un semplice DRR (MS1 in posizione buona e MS2 in posizione pessima).

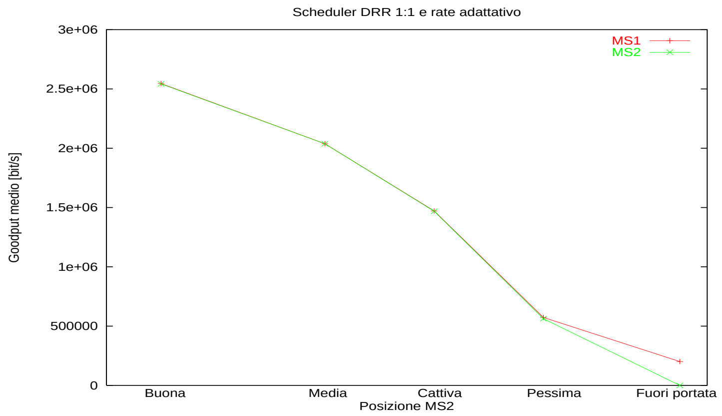

Figura 2.11: Curva prestazioni-qualità dello scheduler DRR derivante dall'HTB (rapporto 1:1).

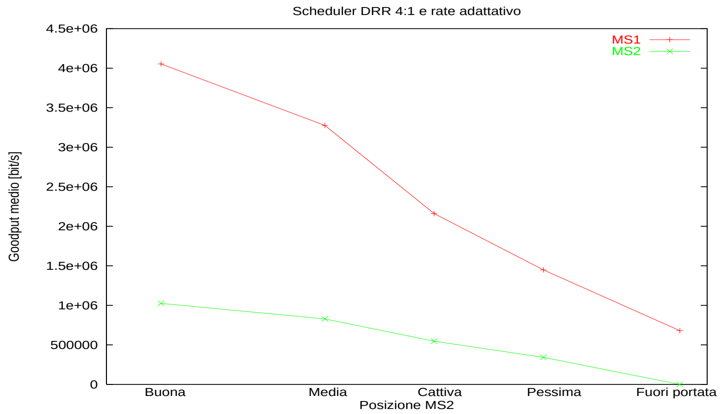

Figura 2.12: Curva prestazioni-qualità dello scheduler DRR derivante dall'HTB (rapporto 4:1).

tivi per illustrare l'effetto delle modifiche apportate al DRR in questa tesi. A questo scopo, la figura 5.9 mostra come, a fronte di un DRR in cui la classe relativa a MS1 viene danneggiata dalla cattiva qualità del canale verso MS2, entrambe le versioni modificate WHTB e TWHTB si comportano meglio perché riescono a limitare il danno. La figura 5.10 offre la chiave di lettura di questo miglioramento: il WHTB e, in misura leggermente minore, il TWHTB, impediscono a MS2 di appropriarsi della maggior parte delle risorse dell'Access Point che dovrebbero essere invece riservate alla trasmissione dei dati verso MS1.

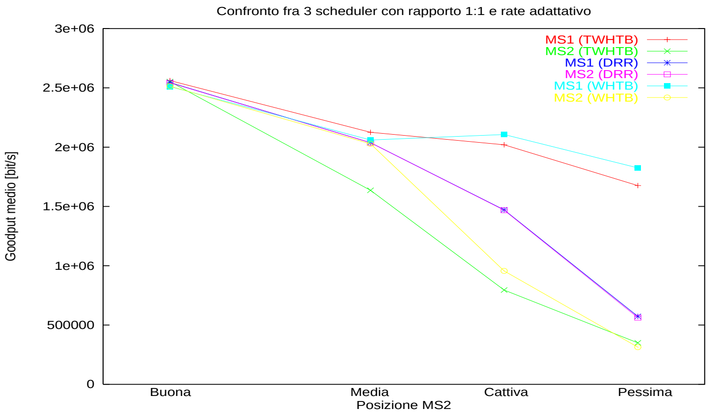

Figura 5.9: Confronto fra la curva prestazioni-qualità del DRR originario e quelle delle due versioni modificate WHTB e TWHTB, con 2.5 Mbit/s assegnati ad entrambe le classi di servizio.

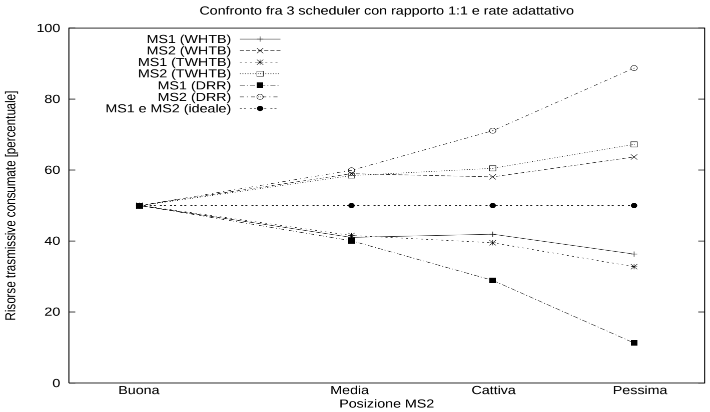

Figura 5.10: Percentuale di risorse trasmissive consumate dalle due classi, con lo stesso rate assegnato ad entrambe: confronto fra DRR, WHTB e TWHTB.

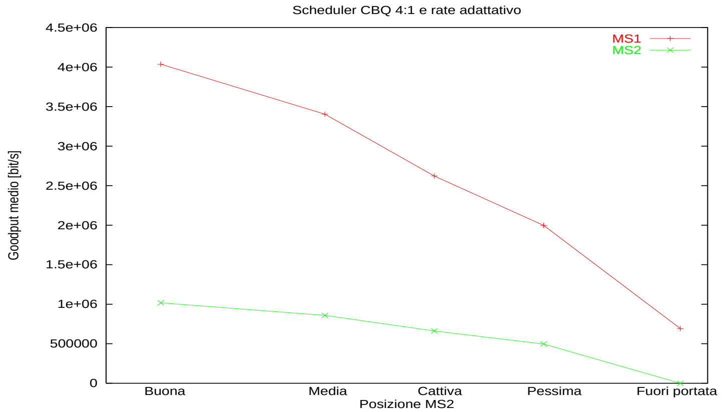

Figura 2.13: Curva prestazioni-qualità dello scheduler CBQ.

ramento delle prestazioni, occorrerà tenere conto della qualità del canale in fase di scheduling, utilizzando il *Wireless Channel Monitor*.

2.4 Lo scheduler HTB modificato per il canale wireless (WHTB)

Abbiamo visto che lo scheduler HTB, da solo, non è in grado di fare un buono scheduling in presenza di canale wireless. In particolare, quando si usa lo scheduler come un semplice Deficit Round Robin, disabilitando di fatto (grazie ad un *cburst* molto grande) il meccanismo di limitazione del rate istantaneo, il risultato è che la stazione mobile che si trova in posizione buona, pur rimanendo ferma, viene notevolmente danneggiata della cattiva qualità del canale dell'altra stazione mobile (figura 2.12).

2.4.1 Scheduling basato sulla qualità del canale

Il principio secondo cui modificare lo scheduler è il seguente: dato che l'interfaccia di rete wireless impiega, per spedire le trame verso una destinazione in posizione cattiva, un tempo mediamente maggiore rispetto ad una destinazione in posizione buona, si introduce una **lunghezza fittizia dei pacchetti** (denominata *stretched*) che cerca di rappresentarne l'allungamento temporale come se si trattasse invece di un vero e proprio aumento della lunghezza in bit.

La scheda di rete, in realtà, non impiega più tempo per trasmettere i pacchetti perché sono più lunghi, ma perché usa un rate più basso sul canale di cattiva qualità, inoltre a volte su tale canale è costretta a fare più di un tentativo di trasmissione. Questo si traduce, come descritto nel primo capitolo (paragrafo 1.2.1), in un abbassamento della capacità trasmissiva verso la corrispondente destinazione, abbassamento che è dunque fortemente correlato alla riduzione del rapporto segnale-rumore. Il *Wireless Channel Monitor* mette in relazione la qualità del canale con la capacità trasmissiva prevista, e lo scheduler calcola la lunghezza *stretched* dividendo la lunghezza effettiva del pacchetto per il **goodput relativo** (rapporto tra goodput atteso e goodput massimo, come esposto a pag. 13).

Nella tabella 2.3 (e, in forma grafica, nella figura 2.14) è riportata la mappa che il *Wireless Channel Monitor* usa per risalire dalla qualità del canale verso ciascuna destinazione (in questo caso l'SNR dei pacchetti ricevuti da essa) ad una stima del goodput ottenibile verso tale destinazione. Le misure sono state effettuate con il tool di calibrazione *WchmonSigMap* su schede Lucent *Orinoco* basate sul driver `wvlan_cs` (e naturalmente sullo standard 802.11b).

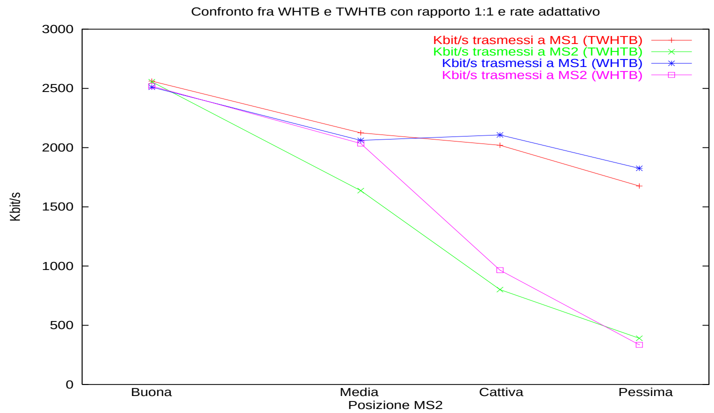

Figura 5.8: Confronto fra i due scheduler sviluppati in questa tesi, con 2.5 Mbit/s assegnati sia a MS1 che a MS2 (dati trasmessi in Kbit/s). Il rate è adattativo.

do a ciascuna delle due destinazioni viene assegnata metà della capacità trasmissiva.

Si nota un comportamento sostanzialmente simile fra le prestazioni dei due scheduler, pur se ottenute con due diverse metodologie di stima del goodput (*Wireless Channel Monitor* e TWCM), a parte il leggero vantaggio del TWHTB con MS2 in posizione media (dovuto ad una migliore stima del canale) ed invece una discesa leggermente più pronunciata del TWHTB con MS2 in posizione pessima (dovuta al fatto che, in caso di perdite di pacchetti, l'ACK non arriva mai e quindi il TWCM non riesce a stimare bene gli intertempi).

Comunque, in entrambi i casi (WHTB e TWHTB), la differenziazione tra le due classi è notevole, e ciò sta a significare che la stima della qualità del canale influisce positivamente sulle prestazioni degli scheduler in una Wireless LAN 802.11b.

A conclusione di quanto sopra esposto, di riportano due grafici compara-

mente minore quantità di dati che gli altri scheduler consentono di ottenere alla stessa stazione mobile in buona posizione. La differenza si nota soprattutto quando la posizione di MS2 è cattiva o pessima, tuttavia si può già constatare un leggero miglioramento rispetto agli scheduler tradizionali nel caso di qualità media.

In particolare, dalla figura 5.6 si vede chiaramente come, a differenza del DRR (curve rossa e verde) che trasmette ad entrambe le destinazioni la stessa quantità di dati, indipendentemente dalla qualità del canale radio, il TWHTB (curve blu e fucsia) tiene effettivamente conto di un diverso goodput relativo per le due stazioni mobili e riduce, di conseguenza, il numero di pacchetti che trasmette a MS2 quando questa si trova in posizione tale da consumare, a parità di goodput, più risorse rispetto a MS1.

5.2.2 Confronto con il WHTB basato sul rapporto segnale-rumore

Posizione MS2	MS	WHTB	TWHTB
	1	2511	2562
Buona	2	2514	2552
	1	2061	2125
Media	2	2028	1636
	1	2107	2020
Cattiva	2	956	795
	1	1825	1676
Pessima	2	313	350

Tabella 5.4: Confronto fra i due scheduler sviluppati in questa tesi, con 2.5 Mbit/s assegnati sia a MS1 che a MS2 (goodput medio in Kbit/s). Il rate è adattativo.

La tabella 5.4 e la figura 5.8 confrontano le due versioni dello scheduler HTB basato sulla qualità del canale radio, il WHTB ed il TWHTB, quan-

SNR (dB)	Goodput (byte/s)
0	5000
1	5000
2	8067
3	26490
4	64579
5	113262
6	184010
7	272346
8	354042
9	401921
10	423819
11	480759
12	516107
13	552194
14	553785
15	564682
16	569708
17	573908
18	578784
19	586438
20	589950
21	593455
22	596960
23	600465
24	603970
25	607475
26	610980
27	614485
28	617990
29	621495
$\geq$30	625000

Tabella 2.3: Mappa del goodput stimato dal *Wireless Channel Monitor* utilizzata per le prove sullo scheduler WHTB.

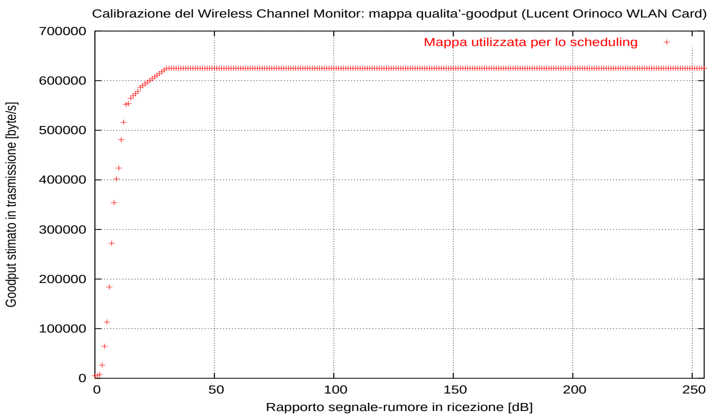

Figura 2.14: Mappa del goodput stimato dal *Wireless Channel Monitor* utilizzata per le prove sullo scheduler WHTB.

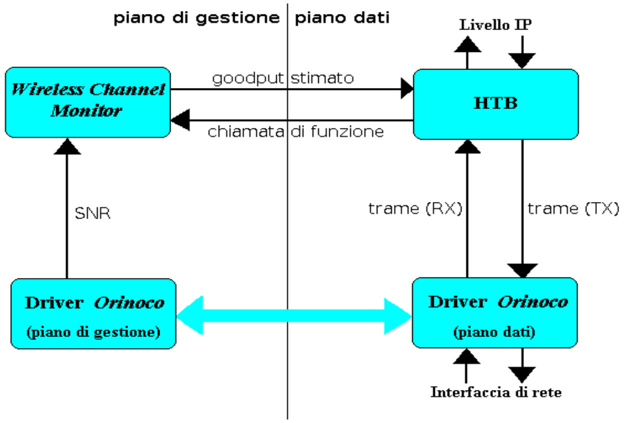

Figura 2.15: Schema a blocchi dello scheduler WHTB.

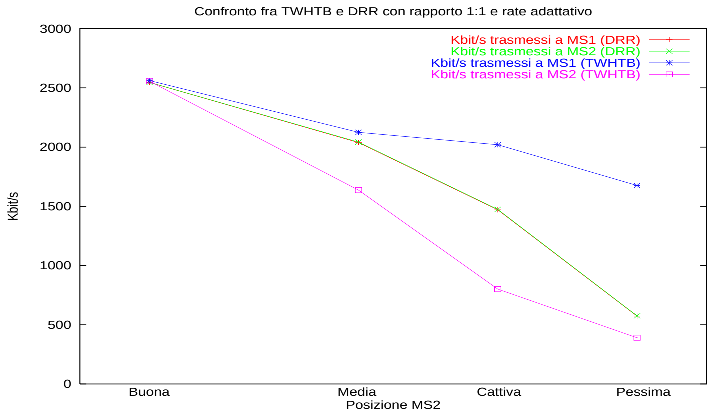

Figura 5.6: Confronto fra DRR e TWHTB in termini di dati trasmessi dallo scheduler alle due classi di servizio.

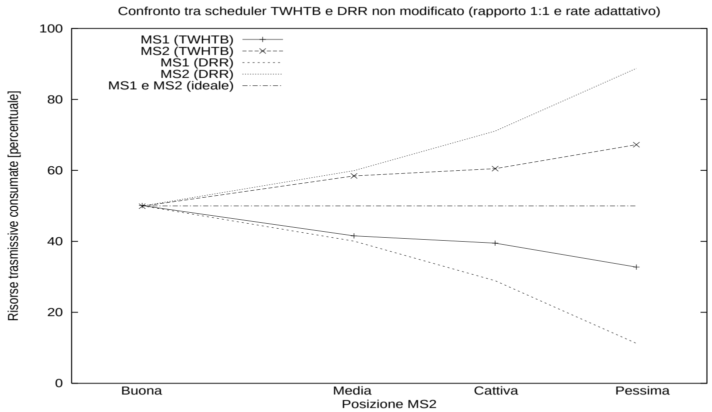

Figura 5.7: Percentuale di risorse trasmissive consumate dalle due classi, con lo stesso rate assegnato ad entrambe: confronto fra DRR e TWHTB.

durata della trasmissione di un pacchetto verso MS1 e verso MS2, e conseguentemente di trasmettere meno pacchetti a MS2, in quanto ciascuno di essi impiega più tempo ad essere trasmesso rispetto a quelli destinati a MS1. In termini di risorse utilizzate (figura 5.5), questo vuol dire che il TWHTB si discosta poco dalla retta ideale in cui entrambe le classi utilizzano l'Access Point al 50%: il consumo di risorse è limitato, per MS2, al 67% circa (in posizione pessima), il che è molto meglio di quanto accadeva ad esempio con il CBQ (figura 1.19), col quale MS2 utilizzava, sempre in posizione pessima, addirittura l'86% delle risorse totali, lasciando a MS1 solamente il 14%.

5.2.1 Confronto del TWHTB con gli scheduler tradizionali

Posizione MS2	MS	CBQ	DRR	TWHTB
Buona	1	2545	2546	2562
	2	2545	2543	2552
Media	1	2000	2038	2125
	2	2000	2037	1636
Cattiva	1	1462	1471	2020
	2	1460	1469	795
Pessima	1	714	573	1676
	2	676	562	350

Tabella 5.3: Confronto fra il TWHTB e gli scheduler tradizionali con 2.5 Mbit/s assegnati sia a MS1 che a MS2 (goodput medio in Kbit/s). Il rate è adattativo.

Anche nel caso in cui si assegni ad entrambe le classi il 50% della banda (in questo caso 2.5 Mbit/s), lo scheduler TWHTB ottiene prestazioni superiori agli scheduler tradizionali quando la qualità del canale verso MS2 non è buona. Osserviamo infatti la notevole differenza, sia nella tabella 5.3 che in figura 5.6, fra la quantità di dati trasmessi a MS1 dal TWHTB e la sensibil-

2.4.2 Il sistema WHTB sviluppato

In figura 2.15 si riporta lo schema effettivamente implementato sul testbed: come stimatore di canale si utilizza il *Wireless Channel Monitor*, mentre lo scheduler modificato è l'HTB, che a questo punto, dato che tiene conto della qualità del canale wireless, può essere identificato con l'acronimo **WHTB** (***Wireless Hierarchical Token Bucket***),

La modifica apportata al codice che implementa l'HTB si può sintetizzare in due operazioni sostanziali (il resto, riportato nell'appendice B, serve solo ad interfacciare lo scheduler con il *Wireless Channel Monitor*).

1. L'operazione più importante è quella che converte la lunghezza di ciascun pacchetto in una lunghezza fittizia:

$$stretched = \frac{length}{\hat{g}(t_0)} = \frac{length}{\frac{\hat{R}(t_0)}{R_{MAX}}} = length \cdot \frac{R_{MAX}}{\hat{R}(t_0)}$$

   dove $\hat{R}(t_0)$ è il goodput stimato dal *Wireless Channel Monitor* verso la destinazione del pacchetto all'istante t_0 in cui tale pacchetto viene estratto dalla coda, mentre R_{MAX} è il goodput ottenibile verso quella destinazione quando la qualità è massima.

2. La seconda operazione serve ad evitare che, quando si divide per un goodput relativo troppo basso, la lunghezza fittizia *stretched* diventi così alta da rendere il *deficit* negativo (cosa che non avrebbe senso). Perciò si pone un limite superiore:

$$stretched < quantum$$

Perciò le operazioni che fa lo scheduler WHTB si possono riassumere nella seguente formula:

$$stretched = \min(length \cdot \frac{R_{MAX}}{\hat{R}(t_0)}, quantum)$$

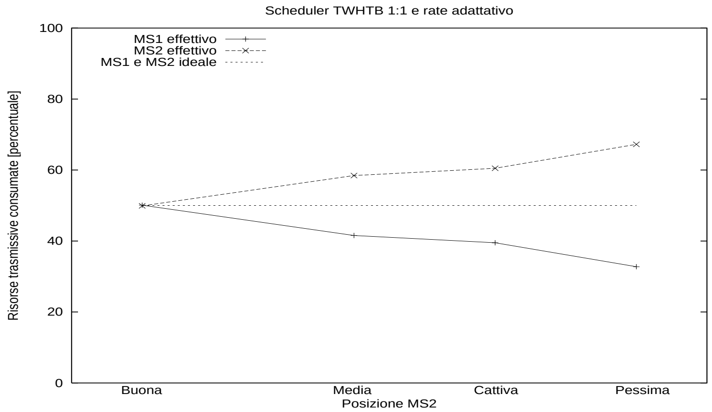

Figura 5.5: Percentuale di risorse trasmissive consumate dalle due classi in presenza dello scheduler TWHTB sull'Access Point, con lo stesso rate assegnato ad entrambe.

5.2 Test di funzionalità con rapporto 1:1

Le stesse misure, effettuate nella sezione precedente con rapporto 4:1, sono state eseguite con un rapporto paritario fra le classi: sia a MS1 che a MS2, nelle prove successive, sono stati assegnati 2.5 Mbit/s, corrispondenti a metà della capacità trasmissiva totale su una rete WLAN 802.11b con pacchetti UDP da 1024 byte.

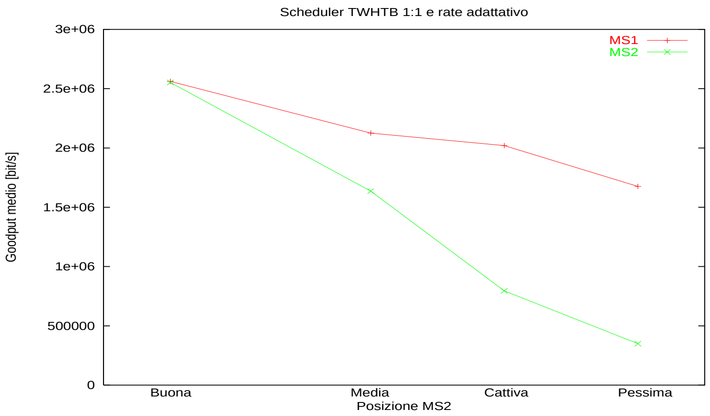

Figura 5.4: Curva prestazioni-qualità dello scheduler TWHTB con rapporto 1:1 fra i rate assegnati alle classi di servizio.

Le figure 5.4 e 5.5 riportano i risultati di tali misure. La prima osservazione che si può fare è che il goodput medio delle due stazioni mobili, pur variando per entrambe con l'allontanarsi di MS2 dall'AP, per MS1 decresce molto più lentamente che per MS2. In altre parole, lo scheduler TWHTB riesce a far ottenere a MS1 una quantità di dati decisamente maggiore di quella che riceve MS2, operando quindi una differenziazione fra le due classi in base alla diversa qualità del canale che esse vedono.

Lo scheduler TWHTB risulta pertanto capace di riconoscere la diversa

Capitolo 3

Risultati sperimentali per il WHTB

Presentiamo ora una serie di prove sperimentali che dimostrano le potenzialità dello scheduling basato sulla qualità del canale radio e le limitazioni del Wireless Hierarchical Token Bucket realizzato come descritto nel capitolo precedente. Le misure (e la successiva elaborazione dei dati) sono state eseguite secondo quanto indicato nell'appendice A.

3.1 Test di funzionalità con rapporto 4:1

Nella prima delle misure effettuate la classe 1:2, relativa alla stazione MS1, deve ricevere l'80% della banda, mentre alla classe 1:4, relativa alla stazione MS2, spetterebbe il 20% rimanente. Pertanto la configurazione del Traffic Control prevede che lo scheduler WHTB riservi 4 Mbit/s alla prima classe e 1 Mbit/s alla seconda. Come in tutte le prove sperimentali di questo capitolo, la posizione di MS2 varia da buona a fuori portata, mentre MS1 resta fissa in posizione buona (vicino all'AP).

Dalle prove svolte con lo scheduler WHTB risulta che il goodput di MS1 non scende mai sotto i 3 Mbit/s (figura 3.1, tabella 3.1): si tratta di un risul-

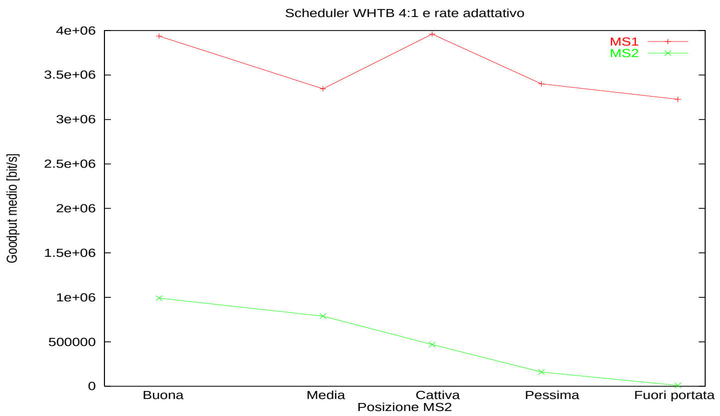

Figura 3.1: Curva prestazioni-qualità dello scheduler WHTB.

tato non ottenibile, con MS2 in posizione cattiva o pessima, dagli scheduler tradizionali (2.13; comunque, MS1 ottiene meno dei 4 Mbit/s assegnati alla classe corrispondente (escluso il caso in cui la qualità di entrambi i link sia buona), e ciò rappresenta sicuramente un limite ad una buona differenziazione del servizio fra le diverse classi di traffico.

La figura 3.2 (da confrontare con la figura 1.17) riporta invece le risorse consumate dalle due classi al variare della posizione di una delle stazioni mobili, MS2, quando sull'AP è presente lo scheduler WHTB. Si può osservare come MS2 in posizione non buona tenda ad utilizzare il canale per più del 20% ad essa assegnato, tuttavia non eccede mai il 35%, qualunque sia la qualità del canale che la collega via radio all'AP. Dal punto di vista di MS1, ciò significa che l'utente 1, pur subendo un certo degrado delle prestazioni a causa della non perfetta qualità del canale verso MS2, riceve comunque, a patto di rimanere vicino all'Access Point, almeno il 65% del goodput totale del link (rispetto all'80% richiesto), ovvero oltre l'80% di quanto gli era stato assegnato a priori.

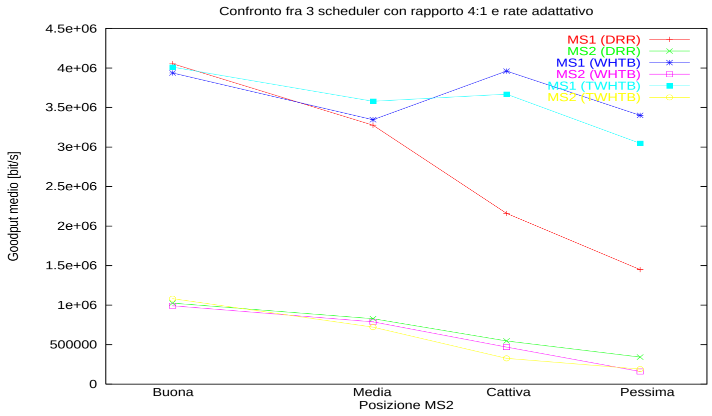

Figura 5.3: Confronto fra la curva prestazioni-qualità del DRR originario e quelle delle due versioni modificate WHTB e TWHTB, con 4 Mbit/s assegnati a MS1 e 1 Mbit/s a MS2.

5.1.2 Confronto con il WHTB basato sul rapporto segnale-rumore

Posizione MS2	MS	WHTB	TWHTB
Buona	1	3937	4011
	2	991	1081
Media	1	3344	3579
	2	788	721
Cattiva	1	3960	3669
	2	469	327
Pessima	1	3401	3047
	2	160	190

Tabella 5.2: Confronto fra i due scheduler sviluppati in questa tesi, con 4 Mbit/s assegnati a MS1 e 1 Mbit/s assegnati a MS2 (goodput medio in Kbit/s). Il rate è adattativo.

Se si confronta il TWHTB con il WHTB, che come stimatore di canale utilizza il *Wireless Channel Monitor*, si possono fare le seguenti considerazioni (tabella 5.2, figura 5.3):

- Entrambi gli scheduler, pur non garantendo a MS1 il goodput desiderato qualora la posizione di MS2 non sia buona, si comportano molto meglio degli scheduler CBQ e DRR, non basati sulla conoscenza del canale wireless.

- Non c'è molta differenza fra le prestazioni dei due scheduler implementati in questo lavoro di tesi. In particolare, quando il canale fra l'Access Point e la seconda stazione mobile è medio, MS1 ottiene un lieve miglioramento quando l'AP usa il TWHTB; quando, al contrario, il canale è cattivo o pessimo, è il WHTB che riesce a trasmettere più dati a MS1.

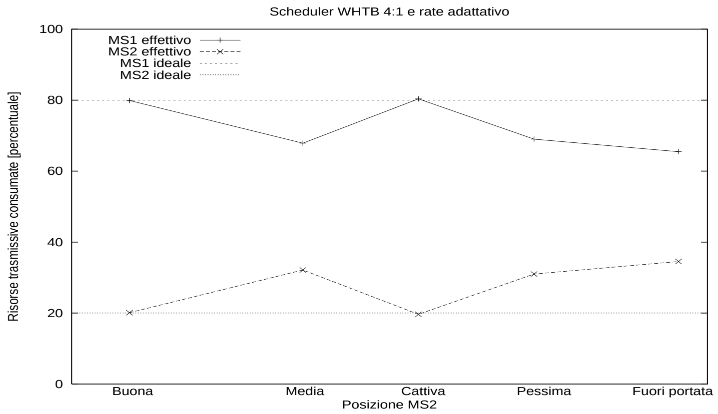

Figura 3.2: Percentuale di risorse trasmissive utilizzate dall'Access Point per trasmettere verso ciascuna delle due stazioni mobili.

3.2 Confronto delle prestazioni del WHTB con quelle degli scheduler tradizionali

3.2.1 Prestazioni con rate adattativo

Se si confrontano gli scheduler finora usati, con parametri di ingresso analoghi (in tutti gli scheduler si assegnano 4 Mbit/s a MS1 e 1 Mbit/s a MS2), si ottengono i seguenti risultati per le possibili posizioni di MS2 (tabella 3.1):
• **buona:** nessuna differenza, perché ogni scheduler assegna alle classi esattamente quanto esse si aspettano;
• **media:** con lo scheduler HTB standard (non modificato e con *cburst* di default) MS1 non perde quasi nulla rispetto al caso precedente, mentre con gli altri scheduler, compreso il WHTB, subisce un leggero peggioramento;
• **cattiva, pessima, fuori portata:** lo scheduler WHTB ha prestazioni migliori di tutti gli scheduler tradizionali utilizzati; in particolare, il goodput

Posizione MS2	MS	CBQ	HTB	DRR	WHTB
	1	4037	4061	4054	3937
Buona	2	1017	1011	1025	991
	1	3404	3966	3276	3344
Media	2	859	609	828	788
	1	2621	3544	2161	3960
Cattiva	2	662	503	546	469
	1	1996	2105	1449	3401
Pessima	2	497	255	342	160
	1	692	1205	680	3226
Fuori portata	2	0	0	0	0

Tabella 3.1: Confronto fra vari scheduler con 4 Mbit/s assegnati a MS1 e 1 Mbit/s assegnati a MS2 (goodput medio in Kbit/s). Il rate è adattativo.

della stazione MS1 si mantiene al di sopra dei 3 Mbit/s nonostante quello di MS2 decresca drasticamente; il CBQ e il DRR si comportano sostanzialmente in modo simile fra loro, con MS2 che influenza, riducendolo notevolmente, il goodput di MS1; lo scheduler HTB standard ha invece prestazioni intermedie fra CBQ/DRR e WHTB, tuttavia ciò è dovuto all'effetto collaterale discusso nel paragrafo 2.3.3.

Per quanto riguarda le risorse di banda effettivamente consumate, la figura 3.4 riporta un confronto fra lo scheduler DRR originale (HTB con *cburst* molto grande, paragrafo 2.3.4) e quello modificato (HTB con *cburst* molto grande ed inoltre con le modifiche descritte nella sezione 2.4).

3.2.2 Prestazioni con rate fissato a 11 Mbit/s

Osserviamo che cosa accade eliminando l'adattatività del rate, fissandolo quindi al valore massimo (11 Mbit/s nominali).

Bisogna notare che il *Wireless Channel Monitor*, progettato per funziona-

anche in questo caso dal confronto emerge il vantaggio, in termini di goodput ricevuto dalla stazione MS1 in buona posizione, dello scheduler che tiene conto della diversa qualità del canale fra l'AP e le due MS. In questo caso, si dimostra valido anche l'approccio allo scheduling basato sugli intertempi, in quanto il TWHTB permette a MS1 di ricevere sempre oltre 3 Mbit/s indipendentemente dalla posizione di MS2, al contrario di CBQ e DRR, che già in posizione cattiva fanno diminuire considerevolmente il goodput di MS1 rispetto ai 4 Mbit/s che le sono stati assegnati.

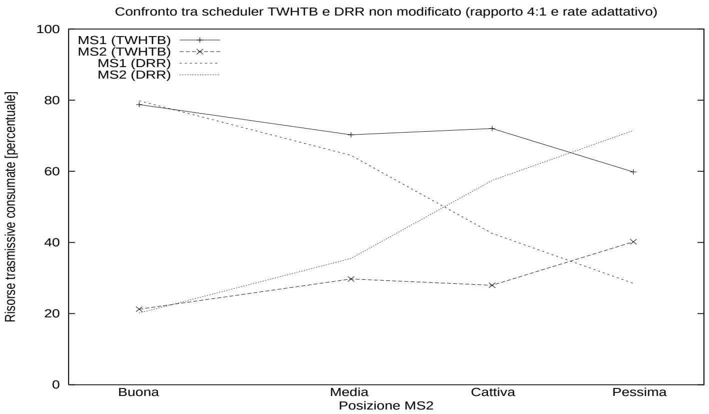

Figura 5.2: Confronto fra le risorse utilizzate dall'AP per trasmettere verso MS1 (in posizione buona) e MS2 (in posizione variabile) con scheduler DRR e TWHTB.

Se si osserva la figura 5.2, si può notare il netto miglioramento della fairness passando dal DRR al TWHTB. Infatti, con il DRR tradizionale la stazione MS2, in posizione cattiva o pessima, pur avendo diritto solo al 20% della banda, arriva a consumare più risorse rispetto a MS1, cui spetta invece l'80%; con il TWHTB, invece, l'Access Point dedica a MS1 circa il 60% delle risorse anche quando MS2 si trova in posizione pessima.

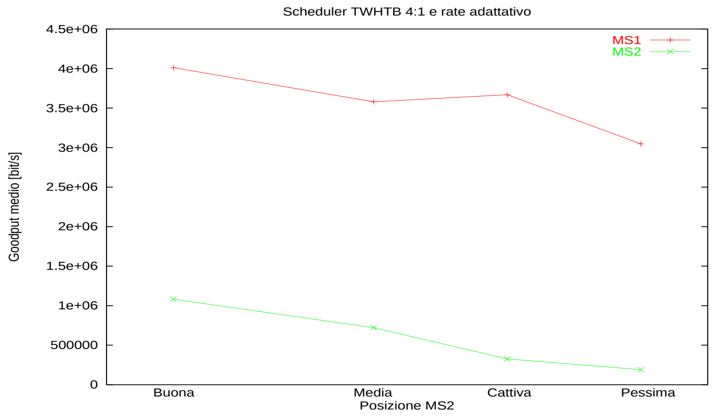

Figura 5.1: Curva prestazioni-qualità dello scheduler TWHTB con rapporto 4:1 fra i rate assegnati alle classi di servizio.

Posizione MS2	MS	CBQ	DRR	TWHTB
Buona	1	4037	4054	4011
	2	1017	1025	1081
Media	1	3404	3276	3579
	2	859	828	721
Cattiva	1	2621	2161	3669
	2	662	546	327
Pessima	1	1996	1449	3047
	2	497	342	190

Tabella 5.1: Confronto fra il TWHTB e gli scheduler tradizionali con 4 Mbit/s assegnati a MS1 e 1 Mbit/s assegnati a MS2 (goodput medio in Kbit/s). Il rate è adattativo.

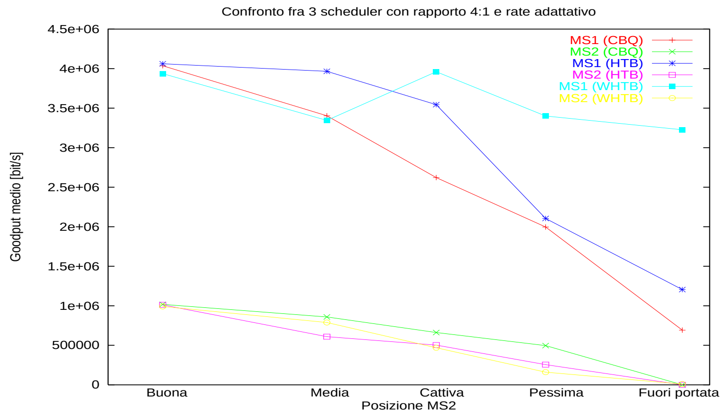

Figura 3.3: Prestazioni a confronto: scheduler CBQ, HTB (con *cburst* come da default), WHTB. In tutti e tre i casi, MS1 avrebbe diritto all'80% della banda.

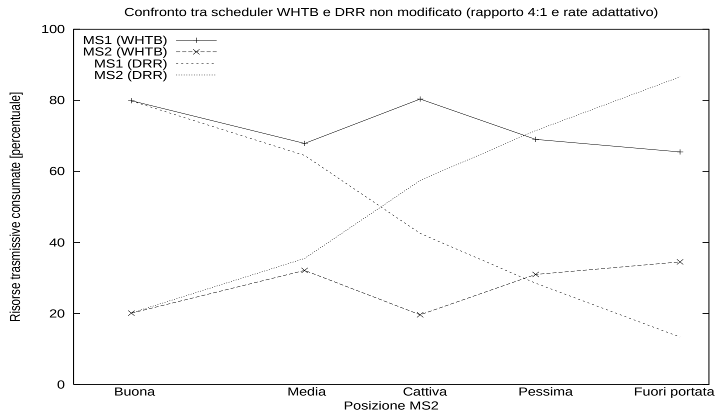

Figura 3.4: Confronto fra le risorse utilizzate dall'AP per trasmettere verso MS1 e MS2 con scheduler DRR e WHTB.

Posizione MS2	MS	Kbit/s ricevuti		Kbit/s trasmessi	
		CBQ	WHTB	CBQ	WHTB
	1	4066	3927	4066	3927
Buona	2	1025	983	1025	983
	1	2736	2865	2736	2865
Media	2	431	524	695	718
	1	2107	2817	2107	2817
Cattiva	2	79	113	541	440
Pessima o	1	1995	3517	1995	3517
fuori portata	2	0	0	508	263

Tabella 3.2: Confronto fra CBQ e WHTB con rate fisso (11M): 4 Mbit/s assegnati alla classe di MS1 e 1 Mbit/s assegnato alla classe di MS2 (goodput medio in Kbit/s).

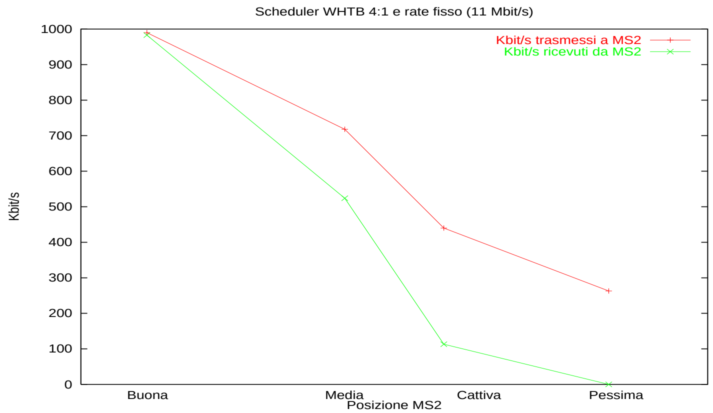

Figura 3.5: Quantità di dati trasmessi dallo scheduler WHTB e ricevuti dalle stazioni mobili quando si fissa il rate dell'Access Point a 11 Mbit/s.

Capitolo 5

Risultati sperimentali per il TWHTB

5.1 Test di funzionalità con rapporto 4:1

Come nel caso del WHTB (capitolo 3), la prima sessione di misure è stata svolta ripartendo la banda in proporzioni diseguali (rapporto 4:1), per mettere in risalto l'eventuale riduzione di goodput, per la stazione MS1 in buona posizione, al variare della qualità del canale fra l'Access Point e MS2.

Notiamo dalla figura 5.1 che, per quanto vi sia ancora una certa degradazione nelle prestazioni, si assiste ad un netto miglioramento del goodput di MS1, rispetto agli scheduler tradizionali, quando MS2 non si trova in posizione buona. Si confronti ad esempio la figura 5.1 con le figure 2.12 e 2.13, riferite rispettivamente al DRR ed al CBQ.

5.1.1 Confronto del TWHTB con gli scheduler tradizionali

La tabella 5.1 mostra un confronto del TWHTB con gli scheduler non basati sulla conoscenza della qualità del canale radio. Come già visto nel capitolo 3,

in quanto il driver dovrebbe distinguere fra la prima trama relativa ad un pacchetto e le successive ritrasmissioni.

Poiché i driver a nostra disposizione (`wvlan_cs`, `orinoco_cs` e `hostap_cs`), nella cosiddetta modalità *Ad-Hoc*, non vedono gli ACK, è necessario che l'AP sia dotato di **due interfacce WLAN**: una deve poter continuare a trasmettere i dati verso le stazioni mobili, e perciò deve essere posta in modalità *Ad-Hoc* [1]; l'altra deve poter vedere gli ACK destinati alla prima interfaccia, quindi va messa in modalità *Monitor*. Il driver da modificare è, di conseguenza, quello che vede gli ACK, ovvero il driver della scheda in modalità *Monitor*, che nel caso di questa tesi è l'*HostAP* [16].

La modifica, per i cui dettagli implementativi si rimanda all'appendice B, non presenta una particolare complessità dal punto di vista teorico: nella funzione `prism2_rx()`, attivata ad ogni trama ricevuta dal driver della scheda in modalità *Monitor*, si inserisce un'istruzione che, nel caso che si tratti di una trama dati, ne registra l'indirizzo di destinazione; quando invece il driver riceve un ACK, si limita a chiamare il TWCM, passandogli come parametri l'ultimo indirizzo registrato e l'istante attuale. Questo perché dal solo ACK (che non contiene l'indirizzo di sorgente) non è possibile capire chi l'abbia inviato, se non supponendo che esso sia riferito all'ultima trama dati trasmessa, cosa che di fatto avviene.

re con rate adattativo [1], è stato calibrato in quest'ultima condizione, dato che il tool di calibrazione non tiene conto delle perdite: il fatto che il rate venga fissato al valore massimo comporta invece una perdita di pacchetti non trascurabile (come si può riscontrare anche dalla differenza tra la curva dei dati trasmessi e quella dei dati ricevuti in figura 3.5), nonché una variazione nella relazione fra qualità del canale e tempi di trasmissione (come vedremo nel prossimo capitolo). Perciò i risultati ottenuti in questo paragrafo hanno un valore puramente indicativo.

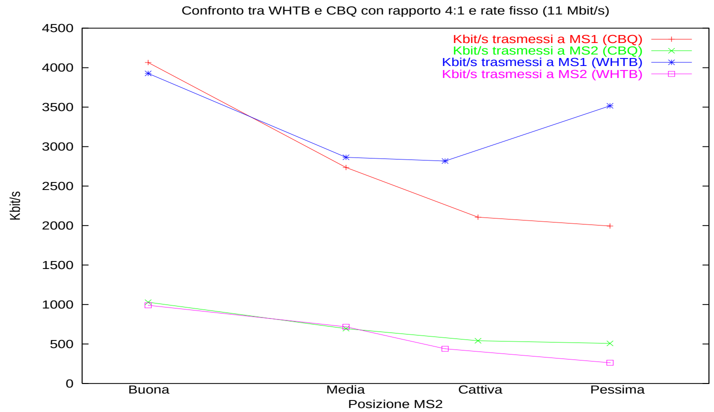

Figura 3.6: Confronto delle prestazioni dello scheduler WHTB con quelle del CBQ, in termini di dati trasmessi, con rate fisso.

Pur partendo da queste premesse, il comportamento del WHTB continua ad essere migliore degli altri scheduler, sebbene con le limitazioni, viste anche in precedenza, che il metodo preso in esame nei prossimi capitoli si propone di superare.

A titolo di esempio, la figura 3.6 evidenzia come il WHTB aumenti il rapporto fra i pacchetti trasmessi verso MS1 e quelli trasmessi verso MS2 quando quest'ultima è in cattiva posizione, mentre per il CBQ il rapporto

rimane costante (in questo caso 4:1) anche nel caso in cui l'Access Point sia costretto a ritrasmettere molte volte i pacchetti a MS2, consumando così ben più del 20% delle risorse.

3.3 Test di funzionalità con rapporto 1:1 (banda suddivisa a metà fra le due classi di servizio)

Posizione MS2	MS	CBQ	HTB	DRR	WHTB
Buona	1	2545	2534	2546	2511
	2	2545	2512	2543	2514
Media	1	2000	1925	2038	2061
	2	2000	1855	2037	2028
Cattiva	1	1462	1492	1471	2107
	2	1460	1153	1469	956
Pessima	1	714	778	573	1825
	2	676	369	562	313
Fuori portata	1	198	199	201	1878
	2	0	0	0	0

Tabella 3.3: Confronto fra vari scheduler con 2.5 Mbit/s assegnati sia a MS1 che a MS2 (goodput medio in Kbit/s). Il rate è adattativo.

Ancora più evidente è quanto accade nel caso in cui ad entrambe le stazioni mobili viene assegnato lo stesso rate. In questa serie di misure, a ciascuna delle due classi di traffico spettano 2.5 Mbit/s, ovvero metà del goodput massimo ottenibile con pacchetti da 1 KB. Anche qui, la classe associata a MS1 non riesce ad ottenere totalmente, nemmeno con il WHTB, quanto ad essa assegnato, tranne (come sempre) quando MS2 si trova in posizione buona come MS1. Ma, al contrario di quanto accade con gli scheduler tradizionali,

Con il nuovo TWCM realizzato come ora descritto, lo scheduler complessivo, derivante dall'HTB con le stesse modifiche eseguite nel secondo capitolo, è ora capace di tenere conto della qualità del canale radio in termini temporali anziché di SNR: lo si può pertanto denominare **TWHTB** (***Time-based*** *Wireless Hierarchical Token Bucket*).

4.2.2 La modifica del driver *HostAP*

Il TWCM, per poter conoscere gli intertempi, deve essere avvertito ogni volta che un pacchetto viene trasmesso (una sola volta per ogni pacchetto, quindi dopo tutte le eventuali ritrasmissioni necessarie).

Nel caso che le perdite a livello IP siano trascurabili, ovvero che ritrasmissioni e rate adattativo consentano di trasmettere la quasi totalità dei pacchetti con successo, si ha una tipica sequenza che si ripete: un certo numero di trame consecutive inviate dall'AP ad una MS (sono i tentativi di trasmettere uno stesso pacchetto, in numero variabile da 1 a *Retry_ Limit*), seguite da un ACK inviato dalla MS all'AP. Ne deriva che il tempo fra un pacchetto ed il successivo è proprio la distanza temporale fra due ACK. Per essere più esatti, dato che pacchetti diversi consecutivi non devono necessariamente essere trasmessi alla stessa destinazione, il tempo impiegato dall'Access Point per inviare un pacchetto, comprese le eventuali ritrasmissioni, è dato dalla **differenza fra l'istante di ricezione dell'ACK del pacchetto considerato e l'istante in cui era arrivato l'ACK del pacchetto precedente**.

Ciò presuppone che gli ACK abbiano durata istantanea, il che non è rigorosamente vero, ma si può ritenere l'approssimazione valida in quanto la durata di un ACK non è molto variabile, oltre ad essere generalmente breve rispetto al tempo totale di trasmissione del pacchetto (tabella 4.1).

In conclusione, il driver può notificare al TWCM l'avvenuta **ricezione di un ACK**, che è unica per ogni pacchetto, mentre sarebbe più complesso inserire la chiamata al TWCM nell'ambito della trasmissione di una trama,

di un pacchetto, rilevata tramite una scheda *Prism 2* in modalità *Monitor* posizionata sull'AP.

4.2.1 Stima del goodput a partire dalla conoscenza degli intertempi

Il dato di ingresso al TWCM è l'istante in cui termina la trasmissione di un pacchetto (vedere anche la sezione successiva), insieme all'indirizzo MAC della destinazione. Il TWCM deve poi produrre, come dato di uscita, una stima del goodput ottenibile verso ciascuna destinazione, da restituire allo scheduler quando esso la richiede.

Indicando con t_1 e t_2 gli istanti di trasmissione di due pacchetti consecutivi, il tempo necessario per trasmettere il secondo è

$$duration = t_2 - t_1$$

perciò, se tale pacchetto è lungo *length*, una stima (approssimativa ma ragionevole) del goodput verso la destinazione dell'ultimo pacchetto è

$$\hat{R} = \frac{length}{duration} = \frac{length}{t_2 - t_1}$$

Per ridurre l'effetto delle fluttuazioni a breve termine della stima del goodput (dovute ad errori occasionali ed all'aleatorietà del backoff stesso), si può applicare un filtro IIR passa-basso, come già visto a pag. 2.2.3:

$$\hat{R}(t_2) = \frac{3}{4} \cdot \hat{R}(t_1) + \frac{1}{4} \cdot \frac{length}{t_2 - t_1}$$

In alternativa, anziché sul goodput stimato, si può applicare il filtro direttamente alla durata; oppure si possono filtrare entrambe le grandezze. Nell'implementazione si è scelto di rimanere nel caso più semplice, lasciando aperta tuttavia la possibilità di intervenire facilmente sul codice nel caso che si ritenga di poter ottenere un miglioramento di prestazioni con l'utilizzo dei suddetti filtri.

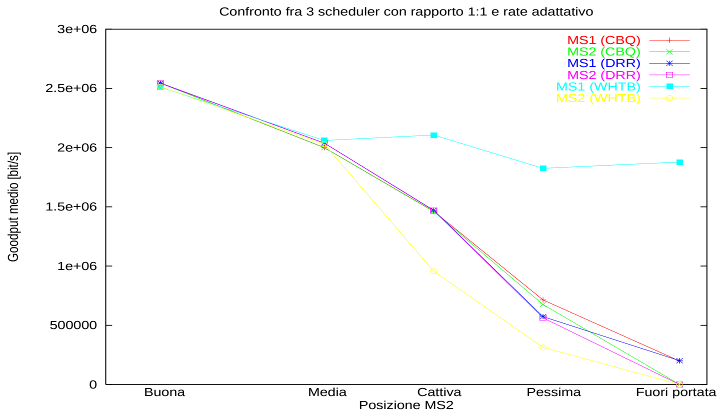

Figura 3.7: Curve prestazioni-qualità degli scheduler CBQ, DRR e WHTB con lo stesso rate assegnato alle due classi.

nel caso del WHTB si riesce a limitare il degrado delle prestazioni di MS1 ad una perdita di meno del 30% rispetto al caso ideale.

Si confronti la figura 3.8 con la 2.10, nella quale invece le due stazioni mobili ricevono sostanzialmente lo stesso goodput, benché MS1 sia in posizione molto migliore rispetto a MS2.

La causa di questa differenza emerge dalla figura 3.9: mentre il DRR, indipendentemente dalle condizioni del canale, trasmette sempre la stessa quantità di dati alle stazioni MS1 e MS2, il WHTB invece tiene conto delle informazioni che il *Wireless Channel Monitor* fornisce allo scheduler e pertanto riesce a fare una differenziazione tra le classi, impedendo a MS2 di appropriarsi della maggior parte delle risorse destinate a MS1.

La figura 3.10 mostra l'utilizzo delle risorse, da parte dell'AP, per trasmettere verso MS1 e MS2, in funzione della qualità del canale verso MS2. Si può notare che, mentre con il DRR la stazione mobile 2 tende a consumare sempre più risorse al peggiorare della propria posizione, arrivando fino al

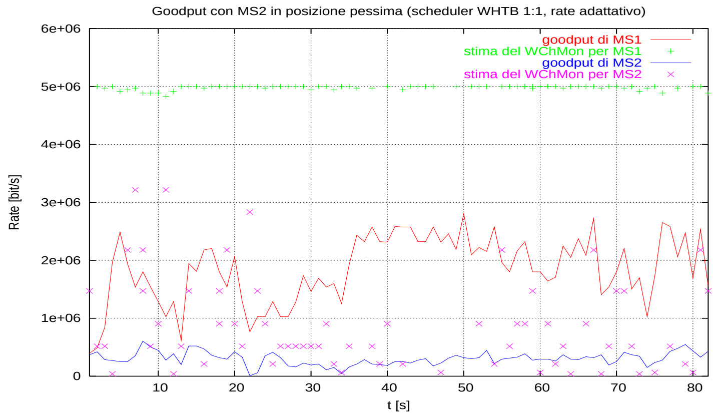

Figura 3.8: Comportamento nel tempo dello scheduler WHTB nel caso in cui ad entrambe le destinazioni venga assegnato lo stesso rate, ma con una (MS2) in posizione pessima.

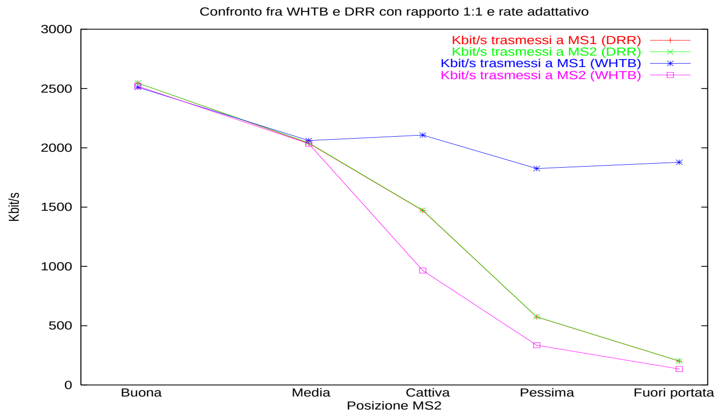

Figura 3.9: Confronto fra la quantità di dati trasmessi alle due classi dagli scheduler DRR e WHTB.

N° di tentativi	Intertempo teorico [ms]	Intertempo misurato [ms]
1	**1.602**	1.646
2	**3.524**	3.762
3	**6.086**	6.567
4	**9.928**	10.409

Tabella 4.4: Valori medi degli intertempi fra i pacchetti nel caso di rate fisso 11 Mbit/s.

4.2 Uno stimatore basato sugli intertempi

Come accennato all'inizio del secondo capitolo (pag. 34), lo stimatore può utilizzare diverse tecniche per valutare la qualità del canale radio fra l'Access Point e ciascuna delle stazioni mobili.

Una di queste tecniche, basata sulla conoscenza (unidirezionale) del rapporto segnale-rumore, è stata utilizzata da [1] per implementare il *Wireless Channel Monitor*, punto di partenza del WHTB sviluppato in questa tesi (sezione 2.4). Un altro possibile metodo di stima del canale è quello presentato in questa sezione: esso tiene conto direttamente del rallentamento della trasmissione dei pacchetti (sezione 4.1), dovuto alla modulazione adattativa e alle ritrasmissioni, in caso di cattiva qualità del link.

Per preservare la modularità del sistema, si è scelto di non modificare ulteriormente lo scheduler, ma di intervenire sul *Wireless Channel Monitor* inserendovi una nuova funzione che, ogni volta che un pacchetto viene trasmesso, ne calcola la durata ed aggiorna il valore del goodput stimato che, anche in questo caso, rappresenta il dato di uscita che lo scheduler richiede. Con questa nuova funzione, lo stimatore di canale agisce in modo diverso dal *Wireless Channel Monitor*, che si basava sull'SNR, perciò gli si può dare un nuovo nome, ***Time-based Wireless Channel Monitor*** **(TWCM)**.

Si è resa necessaria anche una modifica al driver *HostAP* [16], perché ad esso spetta il compito di avvertire il TWCM dell'avvenuta trasmissione

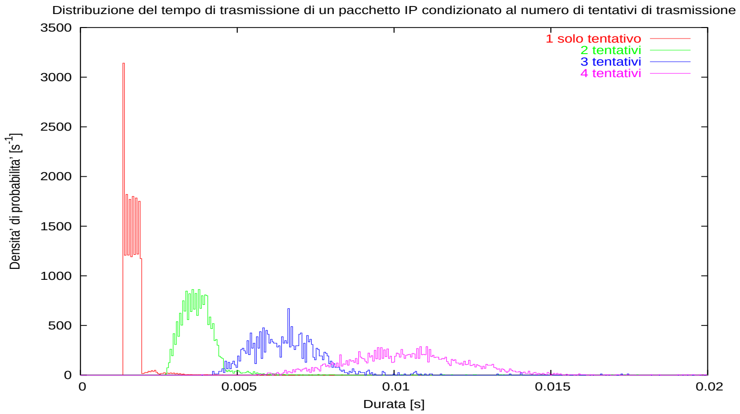

Figura 4.6: Densità di probabilità marginale del tempo di trasmissione di un pacchetto UDP condizionata al numero di tentativi necessari (MS in posizione cattiva, rate fissato a 11 Mbit/s).

L'ultimo grafico (figura 4.6) riporta, analogamente alla figura 4.4, la stima della densità di probabilità marginale del tempo di trasmissione di un pacchetto, condizionata al numero di tentativi di trasmissione necessari. In questa misura, tuttavia, il rate nominale è stato **fissato a 11 Mbit/s**, pertanto in ogni curva si osserva un solo intervallo temporale con un numero significativo di campioni. Ogni curva, secondo la teoria, è la convoluzione della precedente con una densità di probabilità uniforme di larghezza via via crescente. I valori medi delle curve sperimentali sono riportati nella tabella 4.4 e confrontati con i valori medi teorici (prima riga in grassetto nella tabella 4.3).

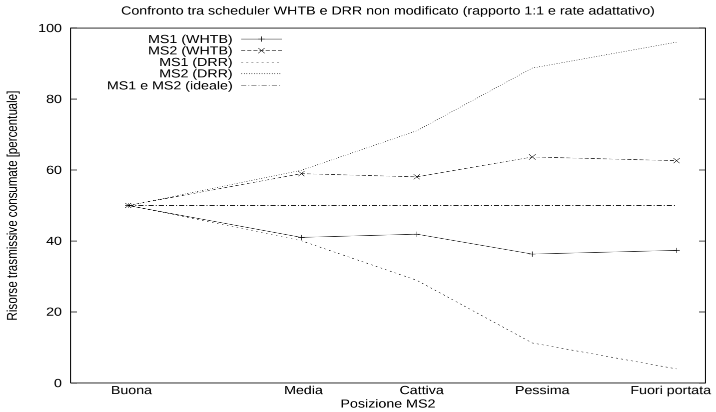

Figura 3.10: Percentuale di risorse trasmissive consumate dalle due classi in presenza dello scheduler DRR o WHTB sull'Access Point, con lo stesso rate assegnato ad entrambe.

96%, con il WHTB tale percentuale rimane sempre al di sotto del 64%. Il WHTB, pertanto, non garantisce ancora una totale fairness, dato che MS2 in posizione pessima consuma quasi il doppio delle risorse di MS1, però il fatto di tenere conto, almeno in parte, della qualità del canale radio, rappresenta già un primo miglioramento rispetto agli scheduler tradizionali.

possibili rate nominali (11, 5.5, 2 e 1 Mbit/s rispettivamente), in quanto il rate si adatta automaticamente alla qualità del canale in base al tasso di errore riscontrato (paragrafo 1.2.1).

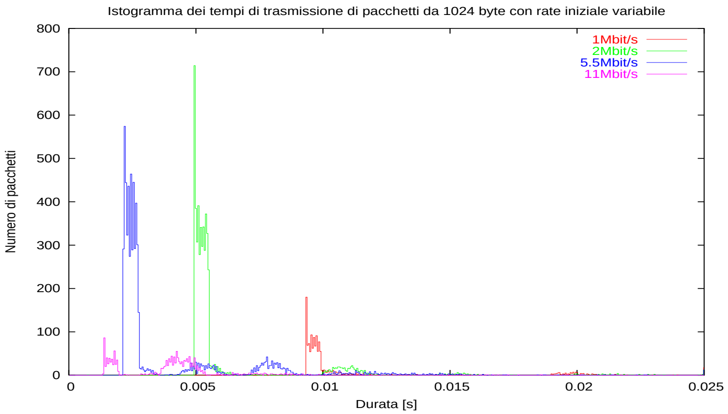

Figura 4.5: Istogramma del tempo di trasmissione di un pacchetto UDP condizionato al rate iniziale (MS in posizione cattiva, rate adattativo).

Anche nell'istogramma di figura 4.5 il rate è adattativo: il grafico, anzi, evidenzia la forte dipendenza del tempo di trasmissione dal rate utilizzato. Il conteggio degli intervalli temporali della durata corrispondente a quanto riportato sull'asse delle ascisse è presentato separatamente per i 4 diversi rate che l'AP ha usato nel primo tentativo di trasmissione: bisogna specificare che **ci si condiziona al rate del primo tentativo**, in quanto il rate può cambiare da una trama all'altra, anche se ciò corrisponde alla ritrasmissione dello stesso pacchetto. Anche in questo caso ogni curva presenta vari picchi, questa volta determinati dal diverso numero di tentativi (1, 2, 3 o 4) per un singolo pacchetto, ma si può notare la netta predominanza di due picchi, relativi ai rate 2M e 5.5M, in cui la trasmissione avviene con successo al primo tentativo.

stanzialmente uniforme (come appare nella figura 4.3 ottenuta con le nostre prove, praticamente identica a quella mostrata in [5]): il tempo necessario per l'invio del datagramma è di fatto il tempo per trasmettere una singola trama, operazione che avviene con successo al primo tentativo. Sull'asse orizzontale è indicato il tempo in μs, mentre l'asse verticale riporta il conteggio degli intertempi di durata corrispondente. Si può osservare come l'intervallo temporale assuma valori discreti, a causa della natura discreta della variabile aleatoria che determina la durata del backoff.

I grafici successivi sono stati ottenuti da una rilevazione su un canale di cattiva qualità, cioè SNR basso dovuto alla maggiore distanza della stazione mobile dall'Access Point.

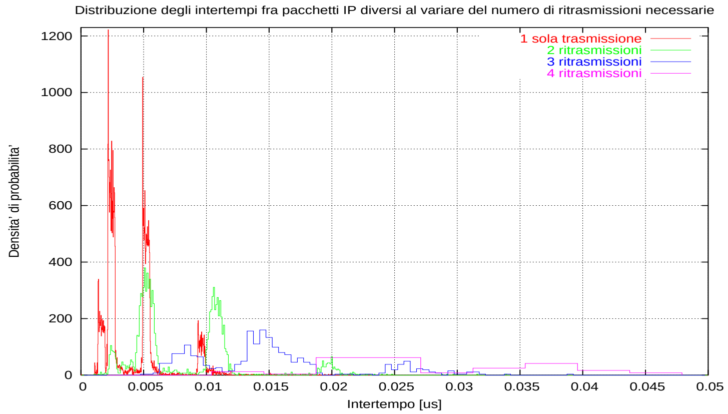

Figura 4.4: Densità di probabilità marginale del tempo di trasmissione di un pacchetto UDP condizionata al numero di tentativi necessari (MS in posizione cattiva, rate adattativo).

La figura 4.4 riporta una stima della densità di probabilità marginale del tempo di trasmissione di un pacchetto, condizionata al numero di tentativi di trasmissione necessari. Ciascuna curva presenta 4 picchi relativi ai 4 diversi

Capitolo 4

Scheduling basato sugli intertempi

Nei capitoli 2 e 3 si è notato come il limite principale del WHTB sia nella stima della qualità del canale radio: al vantaggio di avere a disposizione uno stimatore già implementato (il *Wireless Channel Monitor*) ha corrisposto, d'altra parte, l'inconveniente di una scarsa accuratezza nella stima. A questo punto, una volta modificato lo scheduler (sezione 2.4), l'obiettivo seguente diventa trovare un metodo migliore per stimare il goodput sul canale.

Presentiamo allora un approccio che si basa sull'**intervallo temporale fra i pacchetti**: quest'ultimo corrisponde, dato che si trasmette con continuità, alla durata della trasmissione di un pacchetto su una WLAN (come risulta più chiaro dalle figure 4.1 e 4.2).

Infatti, misurando la durata della trasmissione di un pacchetto, che corrisponde alla somma di tutti i tentativi necessari, si può ottenere una buona stima del goodput, dividendo la lunghezza media dei pacchetti per il tempo impiegato a trasmettere ciascuno di essi.

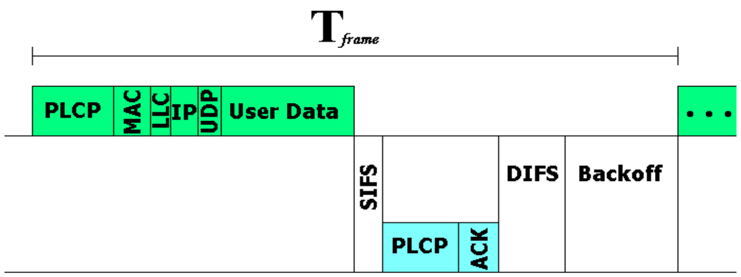

Figura 4.1: Schema generale delle parti di cui si compone la trasmissione di una trama su una rete Wireless LAN 802.11b.

4.1 Durata della trasmissione su una WLAN

4.1.1 Analisi teorica degli intervalli temporali fra i pacchetti

Trascurando i ritardi di propagazione, le 5 fasi di cui si compone la trasmissione di una trama in una Wireless LAN 802.11b (sezione 1.1, [4]) hanno le seguenti durate:

1. Trasmissione dei dati: T_{data}, che comprende il tempo per trasmettere i dati utili a livello utente, T_{user}, più il tempo per i diversi overhead (dal livello UDP al livello fisico), quindi

$$T_{oh} = T_{UDP} + T_{IP} + T_{LLC} + T_{MAC} + T_{PLCP}$$

2. Attesa di un SIFS di durata T_{SIFS}.

tabella 4.3 sono riportate **le loro somme**, che rappresentano i tempi necessari per inviare un pacchetto UDP da 1024 byte su una WLAN, rispettivamente con 1, 2, 3 o 4 tentativi di trasmissione (T_n in figura 4.2).

4.1.2 Misure sperimentali

Con una scheda *Prism 2* in modalità *Monitor* a breve distanza dall'Access Point, un semplice analizzatore di protocollo come Ethereal riesce a captare tutte le trame che l'AP scambia con le stazioni mobili, compresi i tentativi di trasmissione ripetuti e gli ACK ricevuti nel caso che i tentativi vadano a buon fine.

A partire dalle tracce ottenute in questo modo, si possono ricavare alcune distribuzioni statistiche relative alla durata della trasmissione di un datagramma UDP (comprensiva di tutte le ritrasmissioni necessarie) su una Wireless LAN.

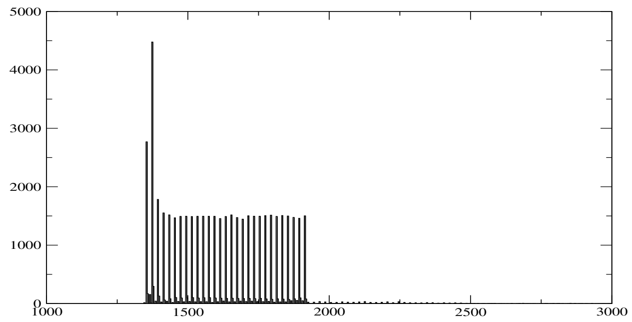

Figura 4.3: Istogramma del tempo di trasmissione di un pacchetto UDP con la MS in posizione buona rispetto all'AP.

Quando il canale è buono, ovvero l'SNR è alto, la distribuzione è so-

	T_1	T_2	T_3	T_4
11 Mbit/s	1 tentativo	2 tentativi	3 tentativi	4 tentativi
minimo	1.292	2.584	3.876	5.168
medio	**1.602**	**3.524**	**6.086**	**9.928**
massimo	1.912	4.464	8.296	14.688
5.5 Mbit/s	1 tentativo	2 tentativi	3 tentativi	4 tentativi
minimo	2.083	4.166	6.249	8.332
medio	**2.393**	**5.106**	**8.459**	**13.092**
massimo	2.703	6.046	10.669	17.852
2 Mbit/s	1 tentativo	2 tentativi	3 tentativi	4 tentativi
minimo	4.852	9.704	14.556	19.408
medio	**5.162**	**10.644**	**16.766**	**24.168**
massimo	5.472	11.584	18.976	28.928
1 Mbit/s	1 tentativo	2 tentativi	3 tentativi	4 tentativi
minimo	9.260	18.520	27.780	37.040
medio	**9.570**	**19.460**	**29.990**	**41.800**
massimo	9.880	20.400	32.200	46.560

Tabella 4.3: Tempo necessario (in ms) per trasmettere un datagramma UDP in una Wireless LAN 802.11b.

3. Ricezione di un ACK in un tempo $T_{ACK} + T_{PLCP}$ (oppure attesa di un *ACK_ Timeout*, che in questa trattazione sarà assimilata al caso di ricezione di un ACK, in quanto in entrambi i casi questa fase ha durata breve rispetto al tempo totale).

4. Attesa di un DIFS di durata $T_{DIFS} = T_{SIFS} + 2 \cdot T_{slot}$.

5. Attesa di un *backoff* di durata $T_{bo} = N \cdot T_{slot}$, dove N è una variabile aleatoria intera distribuita uniformemente fra 0 e la cosiddetta *Contention Window* (CW).

Parametro	Valore (11M)	Valore (5.5M)	Valore (2M)	Valore (1M)
T_{slot}	20 μs	20 μs	20 μs	20 μs
T_{SIFS}	**10** μs	**10** μs	**10** μs	**10** μs
T_{DIFS}	**50** μs	**50** μs	**50** μs	**50** μs
T_{UDP}	6 μs	12 μs	32 μs	64 μs
T_{IP}	15 μs	29 μs	80 μs	160 μs
T_{LLC}	6 μs	12 μs	32 μs	64 μs
T_{MAC}	20 μs	41 μs	112 μs	224 μs
T_{PLCP}	192 μs	192 μs	192 μs	192 μs
T_{oh}	**239** μs	**286** μs	**448** μs	**704** μs
T_{ACK}	56 μs	56 μs	56 μs	112 μs
$T_{ACK} + T_{PLCP}$	**248** μs	**248** μs	**248** μs	**304** μs
CW_1	31 (620 μs)	31 (620 μs)	31 (620 μs)	31 (620 μs)
CW_2	63 (1260 μs)	63 (1260 μs)	63 (1260 μs)	63 (1260 μs)
CW_3	127 (2540 μs)	127 (2540 μs)	127 (2540 μs)	127 (2540 μs)
CW_4	255 (5100 μs)	255 (5100 μs)	255 (5100 μs)	255 (5100 μs)
T_{user} (1KB)	**745** μs	**1489** μs	**4096** μs	**8192** μs

Tabella 4.1: Parametri della Wireless LAN IEEE 802.11b.

La tabella 4.1 riporta i valori dei tempi considerati e della CW_n relativa all'n-esimo tentativo di trasmissione (per i 4 tentativi possibili quando $Retry\_Limit = 4$).

Il tempo totale per trasmettere una trama è perciò dato da

$$T_{frame} = T_{user} + T_{oh} + T_{SIFS} + (T_{ACK} + T_{PLCP}) + T_{DIFS} + T_{bo}$$

dove solo T_{bo} è una variabile aleatoria, mentre il resto è deterministico.

	τ_1	τ_2	τ_3	τ_4
11 Mbit/s	1^o tentativo	2^o tentativo	3^o tentativo	4^o tentativo
minimo	1.292	1.292	1.292	1.292
medio	1.602	1.922	2.562	3.842
massimo	1.912	2.552	3.832	6.392
5.5 Mbit/s	1^o tentativo	2^o tentativo	3^o tentativo	4^o tentativo
minimo	2.083	2.083	2.083	2.083
medio	2.393	2.713	3.353	4.633
massimo	2.703	3.343	4.623	7.183
2 Mbit/s	1^o tentativo	2^o tentativo	3^o tentativo	4^o tentativo
minimo	4.852	4.852	4.852	4.852
medio	5.162	5.482	6.122	7.402
massimo	5.472	6.112	7.392	9.952
1 Mbit/s	1^o tentativo	2^o tentativo	3^o tentativo	4^o tentativo
minimo	9.260	9.260	9.260	9.260
medio	9.570	9.890	10.530	11.810
massimo	9.880	10.520	11.800	14.360

Tabella 4.2: Tempo necessario (in ms) per trasmettere una trama in una Wireless LAN 802.11b.

Nella tabella 4.2 sono calcolati i tempi minimi, medi e massimi per **ciascuno** dei primi 4 tentativi di trasmissione (τ_n in figura 4.2), mentre nella

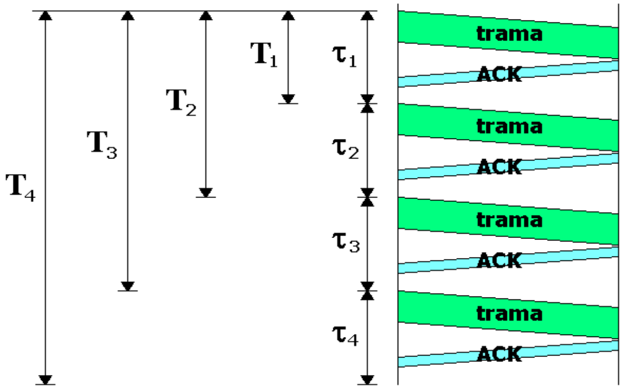

Figura 4.2: Notazioni per le durate dei singoli tentativi di trasmissione di un pacchetto e per la durata complessiva in funzione del numero di tentativi necessari. Si intende che l'unico ACK che realmente viene trasmesso dal destinatario e ricevuto dal mittente delle trame è l'ultimo ACK.