Capitolo 3

Sviluppo del simulatore

3.1 Il simulatore di reti NS-2

La simulazione cerca di costruire un modello del sistema reale. Consideriamo in particolare la simulazione ad eventi discreti: alcuni modelli sono caratterizzati dalla proprietà che le variabili di stato cambiano valore solo ad istanti discreti di tempo, il cambiamento di stato del sistema prende il nome di *evento* ed è caratterizzato da un istante di occorrenza (un evento non ha durata); al contrario l'attività rappresenta una condizione del sistema che perdura per un certo tempo ed è solitamente caratterizzata da un evento di inizio ed un evento di fine (ad esempio l'inizio e la fine della trasmissione di un pacchetto sono eventi, mentre la trasmissione stessa è un'attività); la simulazione ad eventi discreti sarà di fondamentale importanza per questo lavoro di tesi.

Il simulatore di reti utilizzato in questo lavoro di tesi è l' *NS-2* [23] [24], un software nato nel 1989 come variante del simulatore REAL, negli ultimi anni è cambiato molto ed è migliorato notevolmente, tanto da essere riconosciuto oggi, a livello internazionale, come uno fra i migliori della sua categoria. Dello sviluppo di NS si occupano i ricercatori del VINT Project, un progetto che vede la collaborazione di UC Berkeley, LBL, USC/ISI, e Xerox PARC, e che è supportato da DARPA (Defense Advanced Research Projects Agency). NS-2 è un simulatore di reti di telecomunicazioni *orientato agli oggetti* (*object-oriented*) che, in particolari istanti di tempo opportunamente schedulati, esegue determinati *eventi*: troviamo oggetti che implementano protocolli di trasporto come il TCP e l' UDP e protocolli per la gestione della congestione del traffico come RAP e TFRC, altri oggetti che implementano discipline di coda come DropTail, RED e CBQ, altri ancora che

implementano protocolli di routing, protocolli MAC e livelli fisici per reti LAN wired e wireless. La scelta di questo software è stata dettata anche dal fatto che è di tipo *Open Source* (è disponibile integralmente il codice sorgente di tutte le sue versioni, ciò permette di avere accesso completo al suo codice sorgente e di apportarvi qualsiasi tipo di modifica senza alcuna restrizione) e *Freeware* (il software viene scaricato liberamente da Internet e deve essere compilato prima di poterlo usare). La versione da noi utilizzata è la 2.1b9 che, benché non sia l' ultima disponibile, è altamente affidabile. NS-2 è implementato in due differenti linguaggi di programmazione: l' *OTcl* e il *C++*.

Il primo è un linguaggio *interpretato*, derivante dal Tcl e con l' estensione Object Oriented, adoperato sia per realizzare moduli nei quali non serve avere elevate velocità di esecuzione, sia come interfaccia ai moduli realizzati in C++: viene normalmente utilizzato nella stesura degli script che definiscono gli scenari di simulazione, che devono poter essere creati e modificati in maniera semplice e rapida.

Il secondo è un linguaggio *compilato*, con cui sono implementati in NS-2 la maggior parte dei modelli degli oggetti di rete utilizzati durante le simulazioni; esso ha lo svantaggio di essere più complesso e meno istantaneo nella scrittura, ma in compenso è più efficiente in termini di velocità di calcolo.

Tra i moduli realizzati in OTcl e quelli realizzati in C++ esiste una corrispondenza uno a uno, nel senso che gli oggetti sono visibili e accessibili in entrambi i linguaggi, ma ovviamente sono implementati solo in uno dei due. La mappatura a livello utente avviene attraverso comandi *tclcl* racchiusi, generalmente, in uno script; la struttura logica di NS-2 è visualizzata in figura 3.1.

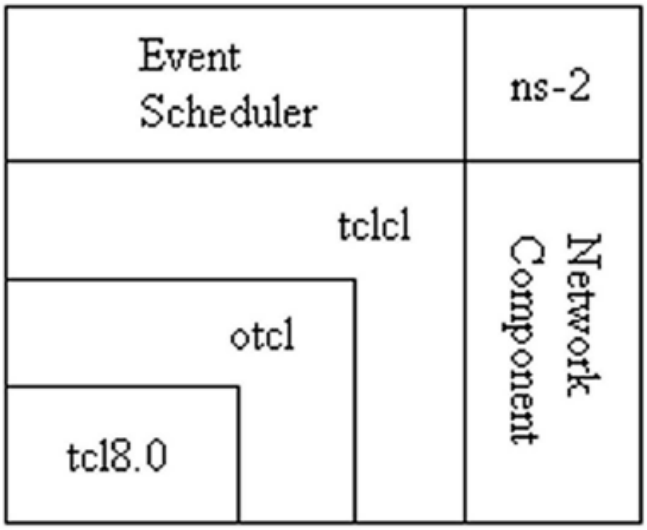

Figura 3.1: Organizzazione logica del simulatore NS-2

La figura 3.2 mostra una vista d'utente del Network Simulator:

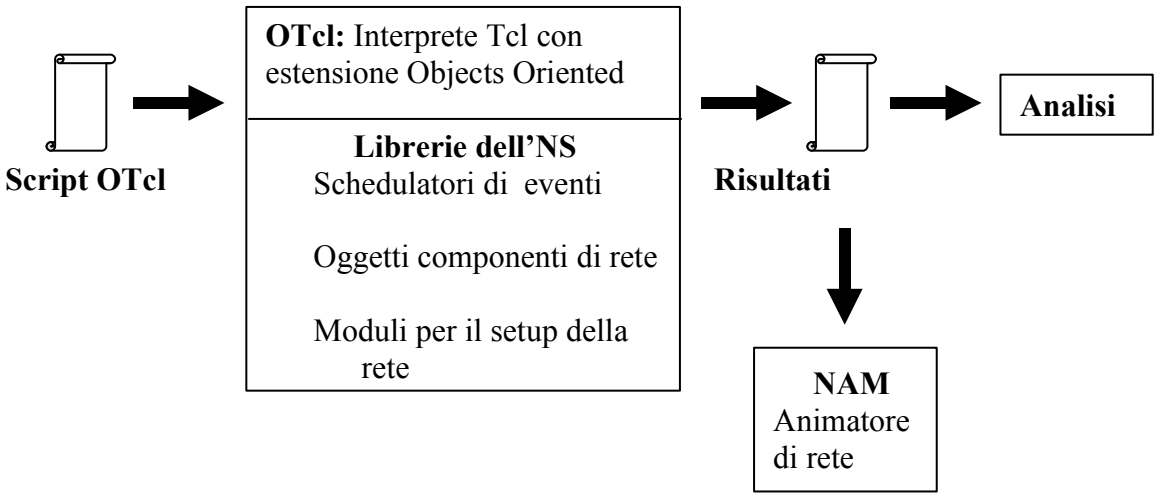

Figura 3.2Vista d'utente dell'NS.

per utilizzarlo bisogna dapprima scrivere un programma in linguaggio OTcl che deve contenere una schedulazione degli eventi (quando una sorgente deve iniziare o smettere di trasmettere, quando acquisire i parametri da analizzare, ecc.), la creazione della topologia della rete tramite gli oggetti OTcl presenti (classi e sottoclassi), e utilizzare le funzioni di libreria per la connessione degli oggetti (setup della rete).

Durante la simulazione, se specificato nel programma OTcl, NS produce in uscita dei file che permettono di effettuare l'analisi prestazionale, oppure di ripercorrere su display l'andamento della simulazione, tramite il Network AniMator (NAM) o XGRAPH.

Il NAM è nato nel 1990. E' un Network Animator che legge un nam trace file prodotto da NS e apre una finestra nella quale dà vita ad un'animazione della simulazione: mostra la topologia della rete, i flussi di pacchetti, i pacchetti in coda nei buffer e quelli dropped, i link che si guastano o che si aggiustano, e così via; attraverso una comoda interfaccia utente è possibile controllare diversi aspetti della simulazione. In figura 3.3 è possibile osservare un esempio di animazione di una simulazione mediante NAM.

XGRAPH è un programma che disegna grafici bidimensionali: legge i trace file prodotti da NS ed apre una finestra nella quale visualizza alcuni risultati della simulazione (come ad esempio lo sfruttamento della banda nel tempo). In figura 3.4 è possibile osservare un esempio di grafico bidimensionale tracciato da XGRAPH.

L' utente solitamente sarà interessato solo alle simulazioni, quindi scriverà uno script OTcl che definisce lo scenario di rete da simulare; tale script viene interpretato da NS, che produce come output i file contenenti i risultati della simulazione, che dovranno solitamente essere analizzati ulteriormente per trarne le informazioni di interesse per l' utente. Se invece si è interessati a modificare o aggiungere nuovi oggetti, bisogna scrivere e compilare in C++.

In NS la realtà è modellata come una successione di eventi: uno *scheduler* tiene traccia del tempo attuale di simulazione (ovviamente completamente incorrelato col tempo reale in cui si esegue il programma e con la sua durata di esecuzione), e della lista di eventi associati.

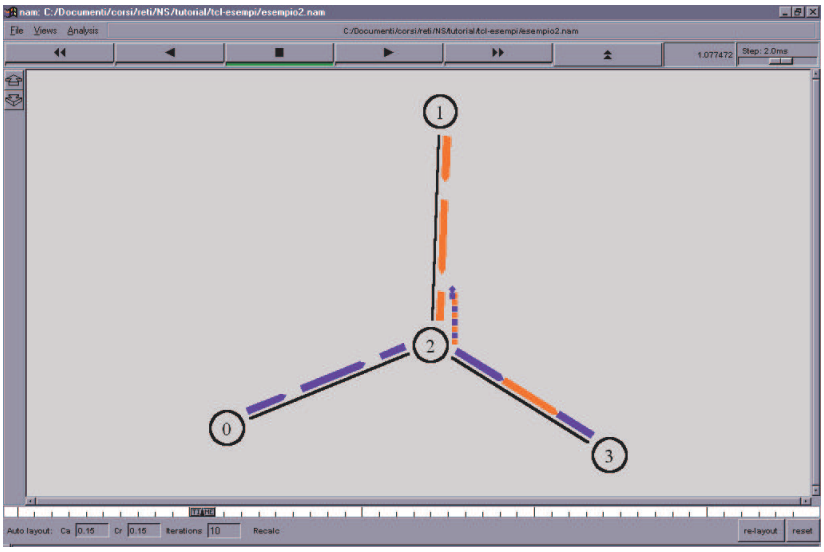

Figura 3.3: esempio animazione di una simulazione mediante NAM

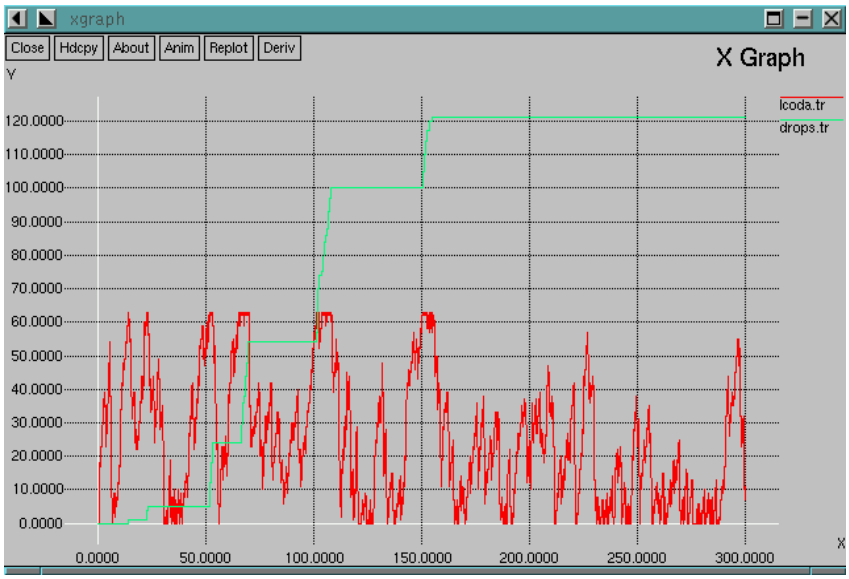

Figura 3.4: esempio di grafico bidimensionale tracciato da XGRAPH

Gli oggetti della rete comunicano fra loro tramite lo scambio di *pacchetti*. Lo scheduler può essere scelto fra quattro diversi tipi, come schematizzato in figura 3.5:

- *simple linked-list*, che implementa lo scheduler come una semplice lista e gli eventi verranno quindi eseguiti nell' ordine in cui compaiono nella lista;
- *heap*, simile al *simple linked-list*, ma l' ordine di esecuzione è quello di una struttura dati di tipo heap;
- *calendar queue* (default), in cui gli eventi vengono eseguiti secondo un preciso calendario;
- *real-time scheduler*, ancora in fase di sviluppo, che è un tentativo di realizzare uno schedulatore di tipo real-time.

Appena lo scheduler determina l'esecuzione di un evento, lo ricerca, lo esegue e si prepara all'esecuzione dell'evento successivo: l'esecuzione di eventi contemporanei avviene in ordine di inserimento (questo perchè il simulatore è un unico processo *single threaded* e non può eseguire più eventi contemporaneamente). È ovvio che il tempo di esecuzione non ha nulla a che vedere con il tempo di simulazione. L' unità di tempo usata nel simulatore è il *secondo*.

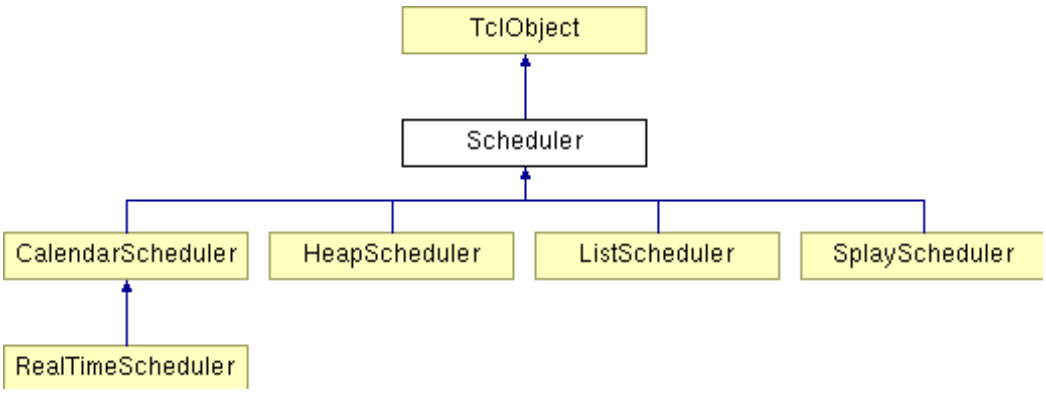

Figura 3.5: Tipi di Scheduler in NS-2

3.2 Modello delle WLAN in NS-2

Nel modello delle WLAN presente in NS-2 è fondamentale la funzione dell'oggetto *MobileNode* che aggiunge il supporto alla mobilità e all'uso del mezzo wireless all'oggetto *Node*.

Come abbiamo visto, NS è organizzato secondo una struttura modulare, in cui ciascun oggetto esegue le proprie elaborazioni e poi ne comunica i risultati agli oggetti adiacenti. In particolare un nodo (W)LAN ha differenti oggetti che simulano indipendentemente i tre livelli più bassi del modello ISO-OSI: *LL, MAC e PHY*. Un apposito modello è schematizzato in figura 3.6.

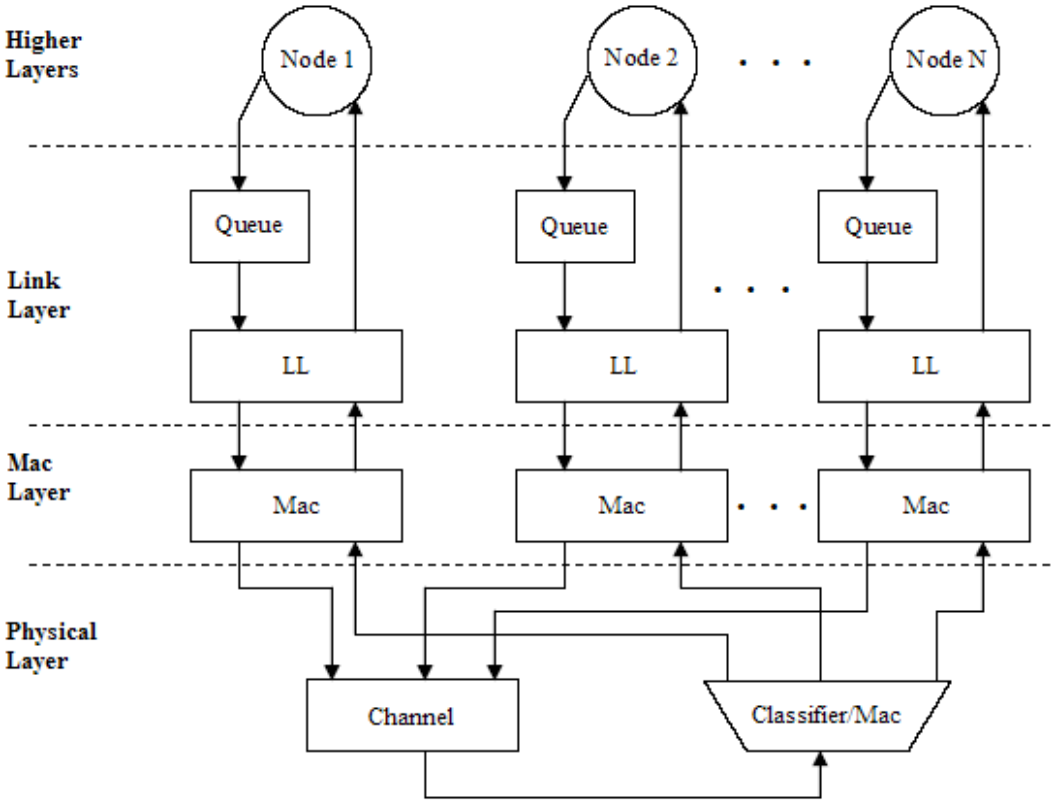

Figura 3.6: Modello della connettività fra i nodi in una LAN in NS-2

Un pacchetto generato dai livelli superiori viene accodato nel Link Layer, poi passato al MAC layer, che definisce le regole per accedere al canale fisico, implementato dall' oggetto *Channel*, che a sua volta provvede a simulare la propagazione e a far giungere in istanti opportuni i pacchetti ai diversi nodi in ascolto sullo stesso mezzo. Il pacchetto ricevuto si trova eventualmente ad effettuare il percorso inverso e a risalire i diversi layer, giungendo fino al livello applicativo del nodo destinatario. Come avviene realmente, tutti i nodi vedono sul canale tutti i pacchetti trasmessi (se sono in visibilità radio), ma sarà compito del livello MAC di ciascun nodo accettare e trasferire ai livelli superiori solo quelli destinati al nodo stesso. La maggiore complessità in ciascun nodo risiede proprio nell' oggetto MAC, che deve provvedere a simulare oltre al funzionamento dei protocolli di accesso al canale propri del particolare livello MAC simulato, anche gli eventuali meccanismi di carrier sense, collision detection, collision avoidance e i tempi di trasmissione fisica del segnale, mentre quelli di propagazione sono simulati dall' oggetto Channel. Il Link Layer invece è responsabile della simulazione dei protocolli data link. Ognuno di questi oggetti è implementato da una classe C++, che fornisce membri pubblici per interfacciarsi con gli altri oggetti. Quando viene creato un nuovo nodo, viene creata una nuova istanza per ciascuna delle classi opportune; l'unica eccezione è costituita dall' oggetto PHY che, per la sua stessa natura, è unico ed e messo in comunicazione con tutti i nodi presenti nello scenario di simulazione. In figura 3.7 è rappresentato il modello di un nodo wireless: l'oggetto ARP (*Address Resolution Protocol*) ha il compito di tradurre gli indirizzi IP in indirizzi MAC generando, nel caso di indirizzi incogniti, dei pacchetti broadcast; l'*IFq* implementa la coda dei pacchetti da trasmettere fornendo particolari precedenze ai pacchetti generati dai protocolli di routing (priority queue); l'oggetto *RTagent* implementa appunto il protocollo di routing che può essere scelto fra DSDV, TORA, AODV o DSR. L'oggetto Src/Sink è il generatore o il destinatario ultimo dei pacchetti di dati; esso è costituito solitamente da un protocollo di trasporto (TCP o UDP) e

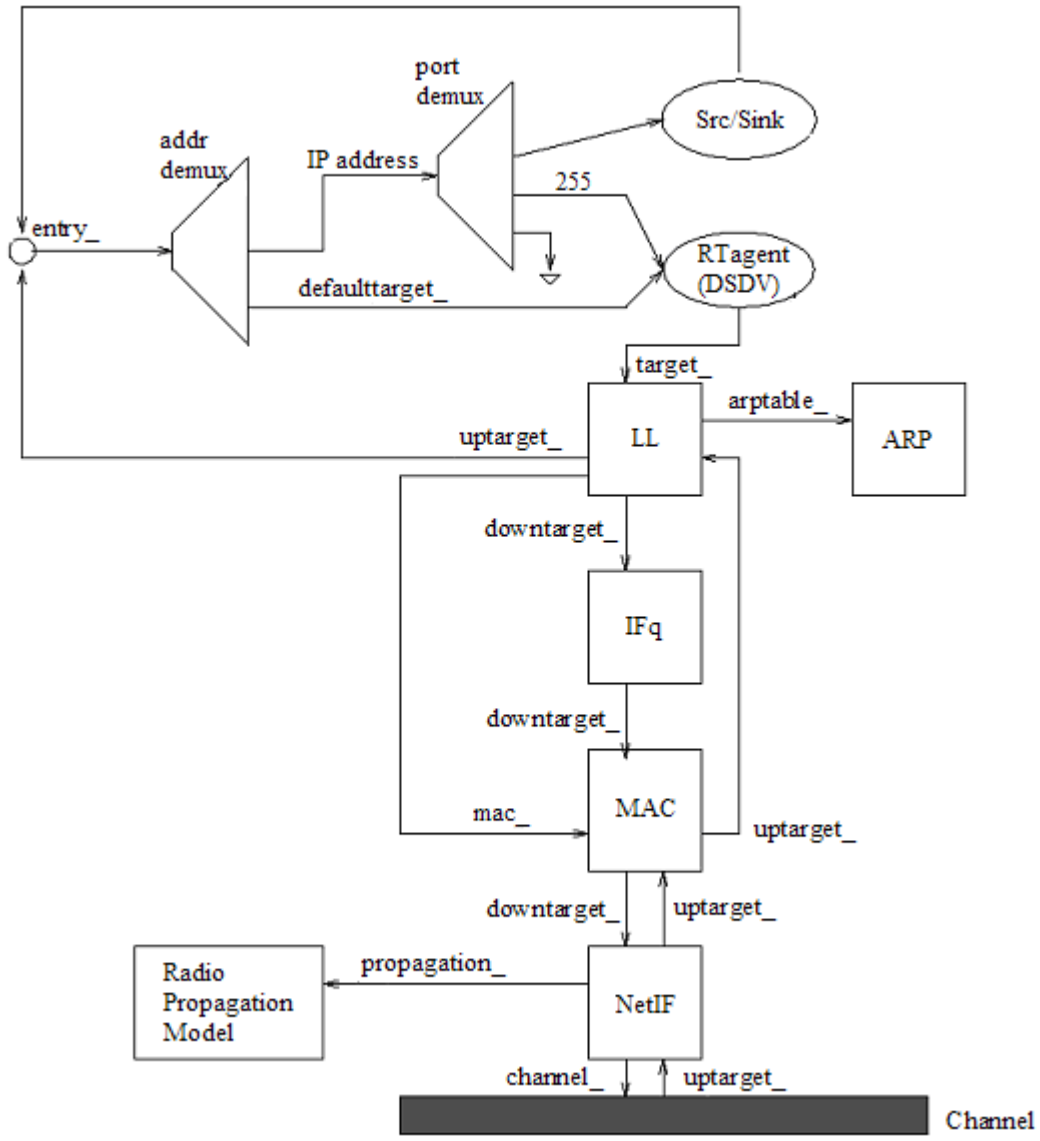

Figura 3.7: Modello di un nodo wireless in NS-2

dall'applicazione particolare; vi possono essere molte istanze diverse di oggetti Src/Sink in ciascun nodo, gestite opportunamente dai demux di indirizzo e di porta; l' oggetto *LL* ha il compito di rigirare all' IFq i pacchetti ricevuti da RTagent, dopo aver interrogato l' ARP

(pacchetti uscenti), oppure di rigirare ai demux i pacchetti ricevuti dal MAC (pacchetti entranti).

Per ciascun nodo wireless NS-2 prevede la possibilità di movimento all' interno di una certa area geografica rettangolare (espressa in metri), e quindi la simulazione della distanza, della velocità di movimento e della visibilità radio in funzione dei livelli di potenza di trasmissione delle diverse stazioni. In particolare l' oggetto *NetIF* simula l' interfaccia hardware di un nodo, e il *Radio Propagation Model* fa uso della formula di attenuazione di Friss (attenuazione proporzionale al quadrato della distanza) per i nodi vicini e del modello a due raggi di propagazione (considerando la terra come un piano perfettamente riflettente) per i nodi lontani, al fine di calcolare la potenza ricevuta da ciascun nodo.

Nell' oggetto *MAC* della versione 2.1b9 di NS-2, dell' 802.11 viene implementato esclusivamente il protocollo DCF, con il meccanismo RTS/CTS e senza l' uso della frammentazione; la velocità fisica di default considerata è 1 Mbps, con modulazione DSSS. Non essendo implementato il protocollo PCF, non vi è alcuna differenziazione fra le STA e l' AP, che quindi non fa uso nemmeno dei beacon frame di sincronizzazione.

Il modello energetico implementato in NS-2 è una caratteristica disponibile solo per i *MobileNode*. All' inizio della simulazione si imposta il livello di energia disponibile nelle stazioni e si impostano anche i consumi, in termini di potenza, relativi alle fasi di ricezione e trasmissione; per ogni pacchetto ricevuto o trasmesso l' energia totale del nodo viene diminuita tenendo conto del tempo impiegato per ricevere/trasmettere il pacchetto in questione e dei valori precedentemente impostati. Se il valore di energia disponibile in un nodo dovesse raggiungere lo zero, allora il nodo non sarà più attivo, quindi scarterà tutti i pacchetti in arrivo e non sarà più in grado di trasmettere.

3.3 Sviluppo della simulazione dell' IEEE 802.11e

Per poter simulare il funzionamento degli algoritmi di allocazione dinamica della banda FBDS e PI-FBDS [cfr. par. 1.6.4.2], si è dovuto prima implementare i protocolli EDCA ed HCCA così come descritti nel draft 8.0 dell' IEEE 802.11e. È stato dunque necessario ridefinire completamente gli oggetti IFq e MAC del modello wireless di NS-2. Innanzitutto si è dovuto implementare le otto code (una per ogni TC) previste dal protocollo EDCA, e per fare ciò è stata introdotta una nuova classe *QoSPriQueue*: si tratta sostanzialmente di un array di otto istanze differenti della classe *PriQueue*, che in NS-2 implementava la coda con priorità maggiore per i pacchetti di routing. La parte di codice che smista i pacchetti che giungono dal layer superiore ciascuno nella coda appropriata, effettua una differenziazione dei pacchetti in base all' indirizzo ed alla porta di destinazione, il che permette di selezionare tutti i pacchetti destinati ad un determinato *traffic sink* ed isolarli in una coda specifica. Ovviamente i comandi che impostano tale caratteristica variano di nodo in nodo e sono comunicati dallo script OTcl che caratterizza lo scenario.

L' oggetto MAC è invece stato radicalmente modificato [18] per la simulazione delle regole di accesso al canale previste dai protocolli di QoS. La maggior parte del codice di seguito descritto si trova nei file *mac-802 11. {cc,h}*, implementato nella classe *Mac802_11*. Tutto il funzionamento della classe si basa sull' uso di alcuni timer, chiamati ogni qual volta si vuole schedulare un qualche evento futuro. Alla scadenza di un timer lo schedulatore di NS-2 invoca la funzione di gestione dell' evento specifico (chiamata *handler*), che provvederà a compiere le elaborazioni necessarie ed a schedulare eventuali altri eventi.

I timer *IFTimer, NavTimer, RxTimer, TxTimer, DeferTimer, BeaconTimer, SchedulerTimer* funzionano all' incirca tutti nello stesso modo (ognuno di essi può essere avviato e cancellato, e alla sua scadenza viene gestito l' evento corrispondente) e i rispettivi handler sono le funzioni *txHandler, navHandler, recvHandler, sendHandler, deferHandler,*

BeaconHandler, HCFscheduler. Il timer *BackoffTimer* è leggermente diverso dai precedenti perché è usato per l' attesa di un tempo casuale estratto secondo le regole del protocollo EDCA; inoltre può essere messo in pausa e riavviato, permettendo così il congelamento ed il ripristino del suo valore; il suo handler è la funzione *backoffHandler*.
La classe *QueueSizeTable* è utilizzata come una struttura dati in cui l' HC memorizza di volta in volta, alla ricezione di un frame, i valori dei livelli di coda dei vari flussi e i loro parametri di TSPEC, comunicati tramite lo script OTcl dello scenario di simulazione.
La parte di codice relativa all' interpretazione dei comandi impartiti via OTcl si trova nella funzione *command*, che permette tra l' altro di definire il nodo che funge da HC (in tal caso sarà aggiornata la variabile *HC*) ed i parametri della QBSS, come la durata del superframe (contenuta nella variabile T_{sf}).
I frame sono creati e distrutti dinamicamente in memoria, e a tale scopo sono usate le variabili puntatore *pktRTS_, pktCTRL_, pktQoSTx_, pktBeacon_, pktQoSPoll_, pktQoSNull_*.
Gli scheduler FBDS e PI-FBDS sono implementati nella funzione *HCFscheduler*, che costituisce l' handler dello SchedulerTimer e viene richiamata all' inizio di ogni CAP per simulare il protocollo HCCA. A sua volta *HCFscheduler* utilizza le funzioni *preassignTXOPs* ed *assignTXOPs* per allocare i TXOP rispettivamente nelle fasi di pre-polling e di polling secondo le modalità dettate dallo scheduler. In *assignTXOPs* vi è anche la parte di codice che gestisce la saturazione del canale e quindi la ridistribuzione dei TXOP.

Per implementare FBDS e PI-FBDS con power saving (PS FBDS e PS PI-FBDS), è stata introdotta la variabile *n_ps_sta*, che esprime il numero di stazioni all' interno della QBSS che hanno comunicato all' HC di voler usare il meccanismo di risparmio energetico: se il valore di n_ps_sta è diverso da zero, l' handler HCFscheduler non utilizzerà la funzione preassignTXOPs, in modo da disabilitare il pre-polling, e alle PS STA a coda nulla assegnerà TXOP adatti alla trasmissione di un MSDU di dimensione massima [21].

Per ulteriori dettagli si faccia riferimento agli appendici A, B e C. L'appendice A riporta un esempio degli script OTcl usati nelle simulazioni, l'appendice B riporta integralmente la funzione HCFscheduler, mentre l' appendice C contiene alcune utility create per elaborare i risultati delle simulazioni.