

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA

Facoltà di scienze matematiche, fisiche e naturali

Corso di Laurea in SCIENZE DELL'INFORMAZIONE

Tesi di Laurea in RETI DI CALCOLATORI

Analisi e sviluppo di un layer di switching ATM per dispositivi embedded

Candidata:
Marika Ercolani

Relatore:
Chiar.mo Prof. Vittorio Maniezzo

Co-Relatore:
Dott. Matteo Roffilli

Anno Accademico 2006/2007 - Sessione III

Indice

Introduzione	i
1 La tecnologia ATM	1
1.1 L'evoluzione di ISDN	1
1.2 La genesi di ATM	2
1.3 Architettura ATM: concetti generali e definizioni	4
1.4 La connessione	6
1.5 Virtual Channel e Virtual Path	7
1.6 La cella ATM	9
1.7 La commutazione ATM	11
1.8 Gli strati dell'architettura ATM	12
1.9 Lo strato fisico	13
1.10 Lo strato ATM	16
2 Il modello dei dispositivi	19
2.1 I dispositivi	20
2.2 I driver	22
3 I dispositivi ATM	25
3.1 Il disegno dell'interfaccia	27
3.2 Reference counting	31
3.3 Il dispositivo ATMDummy	33
4 Progettazione di uno Switch ATM	35
4.1 Aspetti realizzativi	36
4.2 Registrazione dei dispositivi	37
4.3 Meccanismi di sincronizzazione	39
4.4 Routing table	41
4.5 Politica di buffering	44
4.6 Commutazione delle celle	47
4.7 Ottimizzazione del processo di switching	48

4.8	La libreria ATMCell	52
4.9	L'interfaccia laterale di controllo	53
5	Prove realizzate e risultati	57
5.1	Ambiente sperimentale	57
5.2	Calcolo della performance	58
5.3	Impostazione dei test	59
6	Conclusioni e sviluppi futuri	65
	Bibliografia	66

Introduzione

L'ATM (Asynchronous Transfer Mode) [Tan03] è un protocollo di rete a commutazione di cella orientato alla connessione e sviluppato agli inizi degli anni 90 ad opera della International Telecommunications Union (ITU)[Uni] e dell'ATM Forum[Fora]. ATM è un protocollo di trasferimento dati che si situa nello strato 2 della pila dei protocolli ISO/OSI[Tan03] per l'implementazione di reti WAN broadband. L'obiettivo iniziale fu quello di ideare e progettare una tecnologia di rete che si caratterizzasse per una elevata velocità di trasferimento dati, tipicamente di 155 Mbps.

L'esigenza di bassi tempi di trasferimento influenza anche il design di una delle maggiori funzionalità di questo tipo di reti, la commutazione del flusso dei dati; la scelta di celle di dimensione costante con un header ben definito rispecchia questo bisogno, favorendo l'implementazione del processo di switching in hardware capace di commutare il traffico servendosi di matrici elettroniche di commutazione molto veloci.

Una rete ATM è costituita da molteplici circuiti virtuali sopra ai quali corre un flusso di celle opportunamente multiplexato dallo switch; la chiave del processo di commutazione è la determinazione del circuito di uscita della cella entrante nello switch basata sulla coppia di identificatori di cammino e circuito che definiscono il circuito virtuale dal quale la cella è arrivata.

L'ATM Forum definisce molteplici interfacce fisiche in grado di supportare traffico ATM. Le più comuni e veloci sono quelle basate su cavi coassiali e fibre ottiche capaci di raggiungere velocità oltre i 155 Mbps, ma nello standard si definiscono anche i criteri di interfacciamento verso interfacce TDM (Time Division Multiplexing) come E1/T1[Tan03], fino ad arrivare alle specifiche di trasporto di ATM sopra link di dati wireless IEEE802.16 (Wimax)[Forb].

Nel contesto di queste interfacce caratterizzate da una velocità di trasferimento relativamente minore, l'utilizzo di costosi equipaggiamenti hardware dedicati alle funzionalità di commutazione può essere accantonato a favore

di una implementazione software corrispondente dello switching ATM.

Uno dei settori più attivi dal punto di vista della ricerca e mosso costantemente dalla spinta di innovazione è certamente quello della telefonia.

Oggi una rete dedicata alla telefonia non si occupa più solamente di gestire canali di comunicazione atti al trasporto delle comunicazioni vocali ma bensì deve poter essere in grado di offrire un supporto alla natura multiservizio che ormai caratterizza i sistemi di comunicazione mobili di terza generazione, (UMTS, Universal Mobil Telecommunications System)[Tan03].

ATM è la tecnologia adottata in questo tipo di reti per via della sua flessibilità nell'assegnazione di banda, del suo limitato uso di risorse e per la possibilità che è in grado di offrire di gestire servizi caratterizzati da differenti Quality of Service.

All'interno di questo scenario questo progetto si propone di affrontare lo studio e l'implementazione di un layer software di switching ATM, chiamato ATM Soft Switch, per dispositivi di rete embedded ed è stato realizzato all'interno di un progetto di ricerca condotto presso la ditta spagnola Albentia Systems di Madrid¹. Albentia Systems, da poco entrata a far parte ufficialmente del Wimax Forum, si occupa della ricerca e dello sviluppo di prodotti innovativi nel campo delle telecomunicazioni, in particolare dello studio di soluzioni per il networking che si basano sulla tecnologia Wimax.

Telefonica è una delle maggiori compagnie telefoniche spagnole. Il dipartimento di ricerca e sviluppo di questa compagnia ha definito e concordato un progetto con Albentia Systems che prevede il dispiego di nuovi canali per la loro rete telefonica che facciano uso di Wimax come tecnologia del mezzo fisico di trasmissione. Le ragioni che spingono Telefonica a investire tempo e risorse nell'investigazione dell'uso di ponti radio Wimax risiedono principalmente nella maggiore facilità di dispiego di questi segmenti di rete. La stesura del cablaggio infatti, comporta un impegno molto elevato sia a livello di tempo che economico, soprattutto in zone che, per le condizioni sfavorevoli dell'ambiente naturale, rendono la stesura dell'infrastruttura di rete un'impresa ardua.

La Figura 1 rappresenta l'infrastruttura delle reti di accesso UMTS utilizzate nel settore della telefonia mobile ed illustra la topologia tipica di queste reti mettendo in evidenza l'architettura dei protocolli utilizzati negli strati fisico e data link. Gli elementi in blu rappresentano gli switch ATM e corri-

¹www.albentia.com

spondono ai nodi RNC (Radio Network Controller). La funzione svolta dagli RNC è quella di controllare e gestire le stazioni radio base, chiamate nodi B, che gli sono collegate. I nodi B, rappresentati mediante delle antenne, sono gli elementi finali di questa rete e si incaricano di comunicare direttamente con i terminali mobili utilizzando la tecnologia di trasporto WCDMA (Wideband Code Division Multiple Access)[Tan03].

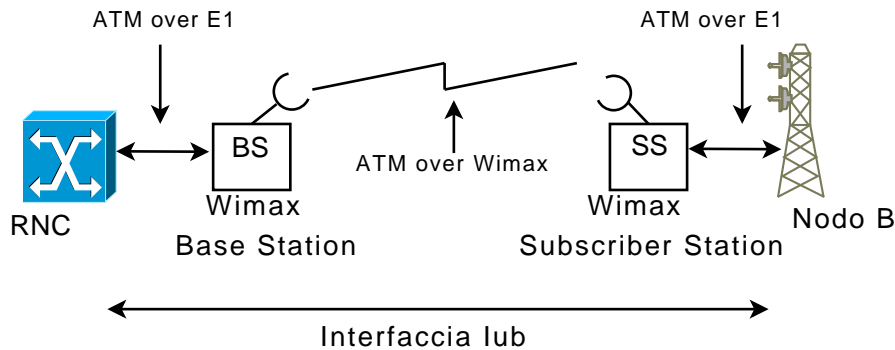


Figura 1: Topologia di una rete UMTS con dorsale ATM. I nodi blu sono switch ATM posizionati in corrispondenza degli RNC, mentre le antenne rappresentano le stazioni radio base (Nodi B). I nodi B comunicano con gli RNC attraverso delle interfacce Iub. Una interfaccia Iub è il collegamento dedicato alle connessioni fra RNC e nodi B.

Le specifiche definite per le reti UMTS prevedono l'impiego di ATM come protocollo data link per l'infrastruttura di trasmissione fissa, per le motivazioni anticipate anteriormente.

Lo strato fisico di questa rete è costituito da un insieme di cavi di tipo E1 che collegano fra loro gli switch ATM della dorsale, i nodi RNC (Radio Network Controller) alla dorsale ed infine le stazioni radio base ai rispettivi RNC. E1 è la sigla di *E-carrier level 1*, ed è uno standard europeo utilizzato come protocollo di trasmissione di conversazioni vocali simultanee su un singolo cavo. Gli standard E-carrier fanno parte di una classificazione gerarchica plesiocrona comunemente utilizzata nell'ambito delle reti telefoniche, come è possibile vedere in Tabella 1:

L'architettura della rete appena illustrata subirà un processo di evoluzione che prevede una trasformazione della configurazione attuale delle interfacce Iub. Queste interfacce collegano ogni stazione base al corrispondente controller RNC e sono le interfacce presenti in maggiore quantità all'interno

Classificazione	Frequenza
E1	2,048 Mbps
E2	8,448 Mbps
E3	34,368 Mbps
E4	139,264 Mbps
E5	564,992 Mbps

Tabella 1: Gerarchia plesiocrona europea.

della rete e, per tanto, sono quelle che incidono direttamente sui costi globali di trasmissione.

L'obiettivo che si sta perseguendo è quello di abbassare notevolmente i costi di creazione e gestione di questi tipi di collegamento ovviando alla ste-sura di cavi di tipo E1 in favore dell'installazione di nuovi link Iub basati su ponti radio che usano Wimax come protocollo di trasmissione.

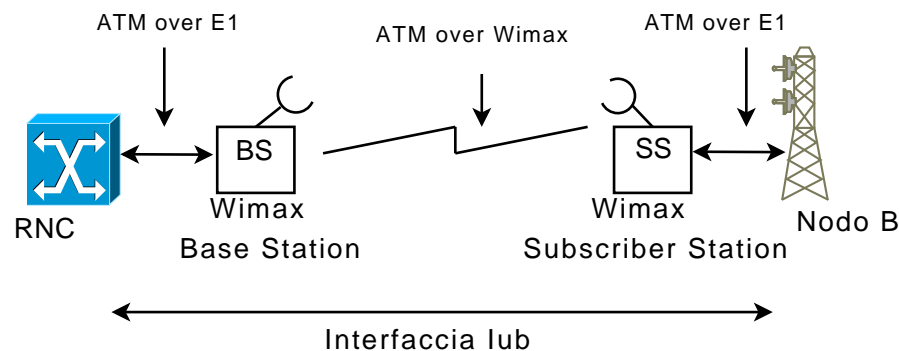


Figura 2: Ponte radio Wimax su un link Iub

I ponti radio sono costituiti da una coppia di ricevitori e trasmettitori posti in corrispondenza degli RNC e dei nodi B. In particolare, il dispositivo connesso al controllore viene chiamato Base Station (BS) mentre quello connesso al nodo B è chiamato Subscriber Station (SS). Ci riferiremo nel seguito a questi due tipi di dispositivi con l'appellativo di stazioni Wimax.

Fra queste due stazioni, una rete Wimax trasporterà il traffico ATM che verrà scambiato lungo l'interfaccia Iub.

Il risultato di questo progetto è l'implementazione realizzata completamente via software di uno strato di switching ATM destinato ad essere installato nelle stazioni Wimax precedentemente descritte. Il ruolo di questo

switch è quello di commutare il traffico ATM proveniente dalle interfacce E1 a quelle Wimax e viceversa.

Visto il numero ridotto di connessioni che si dovranno gestire attraverso una singola interfaccia Iub, lo sforzo di una implementazione completamente software del layer di switching viene ampiamente ripagato, dato che permette un risparmio considerevole sugli apparati utilizzati, essendo comunque in grado di mantenere inalterata l'efficienza del processo di switching ATM.

Lo switch verrà implementato in un driver che verrà direttamente inserito nel kernel del sistema operativo delle stazioni Wimax, e costituirà uno strato di gestione del sistema ATM che si posizionerà superiormente allo strato fisico. Il sistema di switching è stato progettato in maniera astratta e scalabile; lo scopo è quello di poter registrare all'interno di questo sistema interfacce caratterizzate da diversi strati fisici e, allo stesso tempo quello di garantire la capacità del sistema di scalare a seconda delle necessità e delle disponibilità della rete UMTS.

Come modello di riferimento per l'implementazione di un tale sistema, si prenderà il modello dei driver del kernel di Linux; si cercherà di capire le dinamiche del sistema di gestione dei dispositivi e si disegnerà un sistema di gestione ad hoc per dispositivi ATM. Il Capitolo 2 offre una panoramica dei tratti salienti del complesso modello dei driver del kernel di Linux.

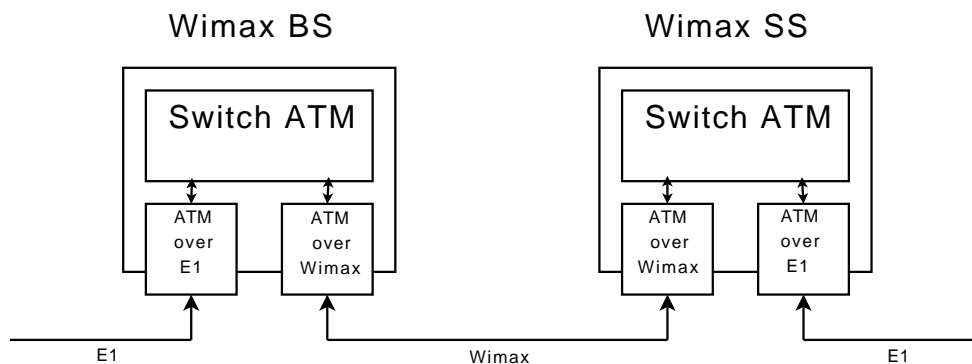


Figura 3: Interfacce ATM di un ponte radio Wimax

Nel Capitolo 3 viene spiegato come lo strato ATM è capace di gestire e controllare dispositivi ATM aventi differenti interfacce fisiche. Le interfacce d'interesse per questo progetto sono costituite dai collegamenti E1 e Wimax. Prima di procedere con lo sviluppo di driver per i dispositivi ATM over E1 e ATM over Wimax, si svilupperà un dispositivo ATM Dummy il cui scopo

sarà quello di offrire uno strumento di test del sistema in costruzione. La Figura 3 evidenzia le relazioni fra stazioni target, strato ATM e dispositivi ATM delle diverse interfacce fisiche coinvolte nel dispiegamento di un collegamento Iub.

Il Capitolo 4 illustra le funzionalità dello Soft Switch ATM e descrive in modo approfondito gli aspetti teorici e tecnici dell'implementazione di questo strato. Viene anche trattata la presentazione delle diverse strutture di dati e dei diversi algoritmi utilizzati per implementare la funzionalità più importante dello switch, la commutazione di celle ATM. Le differenti implementazioni sono volte a cercare di diminuire il tempo di switching. Nel Capitolo 5 verranno esposti i risultati dei test effettuati al fine di misurare la performance del sistema in relazione con i diversi algoritmi di switching utilizzati.

Capitolo 1

La tecnologia ATM

1.1 L'evoluzione di ISDN

Il primo esempio di integrazione nella fornitura di servizi è rappresentato dalla rete ISDN che è stata definita nei suoi aspetti base nel corso degli anni '70. La sua caratteristica fondamentale è quella di fornire una connettività numerica da estremo a estremo per il supporto di servizi con banda inferiore a 2 Mbps, tipicamente il servizio di telefonia e i servizi dati a bassa velocità. Il concetto di integrazione adottato nella ISDN è limitato alle modalità di accesso dell'utente alle funzionalità di trasporto; queste ultime rimangono però differenziate per tipologia di traffico e di servizio ed è compito della centrale locale dirigere il traffico verso la rete di trasporto più adatta (circuit, pacchetto, segnalazione).

Nel corso degli anni '80 lo scenario dei servizi si è rapidamente modificato rispetto a quello considerato nella definizione della ISDN. Sono emerse nuove esigenze di trasferimento in rete geografica principalmente dovute a:

- il supporto di flussi dati ad elevate capacità (oltre i 10 Mbps) quali quelli derivanti dalla interconnessione di reti locali o dal collegamento tra elaboratori remoti per il supporto di applicazioni distribuite;
- il supporto di flussi numerici derivati dalla codifica di immagini sia fisse che in movimento con diverso grado di definizione;
- il supporto di servizi richiedenti il trasferimento di diverse tipologie di flussi informativi, normalmente indicati come servizi multimediali.

1.2 La genesi di ATM

Queste nuove esigenze hanno dato luogo ad un processo di avanzamento del concetto di integrazione in cui fossero compresi anche il trasferimento di flussi numerici ad alta velocità e le relative funzionalità di accesso. L'approdo finale di questo processo è stata la definizione della rete numerica integrata nei servizi a larga banda B-ISDN (Broadband Integrated Service Data Network).

L'ITU (International Telecommunication Union), l'ente di standardizzazione internazionale nel settore delle telecomunicazioni, nella raccomandazione I.121, fornisce le seguenti definizioni:

“The main feature of the ISDN concept is the support of a wide range of audio, video and data applications in the same network. A key element of service integration for an ISDN is the provision of a wide range of services to a broad variety of users utilizing a limited set of connection types and multipurpose user/network interfaces. The term B-ISDN is used ”

L'obiettivo più ambizioso nella definizione di B-ISDN è quello di estendere il concetto di integrazione non solo alle funzionalità di accesso d'utente, ma anche a quelle di trasporto individuando una modalità unica di trattamento in rete dei flussi informativi, ovvero un unico modo di trasferimento.

La definizione del modo di trasferimento più adatto al supporto della molteplicità di flussi informativi previsti in un ambiente a larga banda ha richiesto vari anni di studio. La convergenza verso una soluzione si è avuta intorno alla fine degli anni '80 ed il risultato è stato la definizione dell'ATM (Asynchronous Transfer Mode), o Modo di Trasferimento Asincrono, come definito da ITU.

“ATM is the transfer mode solution for implementing a B-ISDN.”

I requisiti che devono essere soddisfatti da un modo di trasferimento sono i seguenti:

- flessibilità nella assegnazione della banda;
- efficienza di utilizzazione delle risorse;
- minimizzazione del ritardo di trasferimento e della sua variabilità.

I modi di trasferimento tradizionali, circuito e pacchetto, non potevano essere direttamente utilizzati nella B-ISDN. Il modo di trasferimento a circuito, se, da un lato, garantisce un ritardo di trasferimento in rete estremamente

basso e soprattutto costante, dall'altro, a causa dell'uso della moltiplicazione statica, pone notevoli problemi di flessibilità nell'assegnazione della banda e di efficienza delle risorse trasmissive; viceversa, il modo di trasferimento a pacchetto, basandosi sull'uso della moltiplicazione dinamica, è intrinsecamente orientato al raggiungimento di elevati valori di efficienza ed è naturalmente flessibile nell'assegnazione della banda; il suo punto debole riguarda il ritardo di trasferimento che è variabile in funzione dello stato di congestione in cui si trova la rete al momento del transito dei pacchetti appartenenti ad una comunicazione.

Alle precedenti considerazioni va aggiunto che i vicoli di integrità posti dai vari tipi di applicazioni possono essere estremamente diversi, imponendo o meno la necessità di prevedere meccanismi di controllo d'errore e quindi un trattamento differenziato dei vari tipi di servizi. Da quanto detto, risulta evidente che l'adozione di un unico modo di trasferimento poteva emergere solo dalla ricerca di un compromesso tra i fattori ora descritti. L'ATM è il risultato di questo compromesso.

L'ATM è un modo di trasferimento orientato al pacchetto. È in grado quindi di soddisfare naturalmente i requisiti di efficienza ed è flessibile nell'assegnazione della banda. I problemi relativi alla variabilità del ritardo di transito sono stati risolti cercando di diminuire il più possibile i fenomeni di accodamento all'interno dei nodi; ciò è stato possibile aumentando la velocità delle linee e quindi diminuendo il tempo di trasmissione delle unità informative e diminuendo il carico elaborativo dei nodi di commutazione legato al processamento di ogni singola cella in modo da aumentare il throughput del nodo stesso. Per ottenere una diminuzione del carico elaborativo si è assegnato ai nodi interni della rete (nodi di transito) solo l'esecuzione dell'insieme delle funzioni che sono comuni a tutti i tipi di applicazione e facendo migrare verso la periferia della rete (terminali o nodi di accesso) quelle funzioni caratterizzate da una maggiore complessità elaborativa che sono invece dipendenti dal tipo di applicazione (controllo di sequenzialità, controllo d'errore, equalizzazione dei ritardi).

L'ipotesi di partenza che ha guidato la definizione delle caratteristiche dell'ATM è quella di avere un modo di trasferimento per il supporto delle funzionalità di trasporto della rete integrata a larga banda. Tale obiettivo si è rivelato nel tempo troppo ambizioso e praticamente non raggiungibile e le applicazioni dell'ATM in realtà sono state limitate a settori specifici, quali ad esempio il supporto delle comunicazioni dati ad alta velocità. A conferma di questa tendenza applicativa dell'ATM diversa da quella originaria, nel 1991,

ad opera di quattro dei maggiori produttori di sistemi di telecomunicazioni (Stratacom, Newbridge, Cisco e NET), è stato creato l'ATM Forum. Lo scopo dell'ATM Forum è stato quello di accelerare e facilitare lo sviluppo della tecnologia ATM, affiancando l'ITU nell'opera di definizione degli standard. In particolare, l'orientamento dell'ATM Forum è stato quello di contribuire alle specificazioni nell'area dell'interconnessione per dati in area locale, che era riconosciuta come l'area in cui l'ATM avrebbe trovato la maggiore applicazione.

1.3 Architettura ATM: concetti generali e definizioni

L'ATM è un modo di trasferimento orientato al pacchetto e alla connessione; i pacchetti, indicati con il nome di celle, sono di lunghezza fissa pari a 53 byte. Il formato di una cella è costituito da 5 byte di intestazione (header) e da 48 byte di contenuto informativo (payload). Le celle sono assegnate ad una sorgente su domanda secondo le sue caratteristiche di attività e vengono multiplexate statisticamente sull'interfaccia trasmissiva. Si definiscono due tipi di interfacce:

- Interfaccia utente-rete UNI (User Network Interface) che definisce le procedure ed i protocolli che regolano l'interazione tra una postazione d'utente ATM e un nodo ATM;
- Interfaccia rete-rete NNI (Network Network Interface) che definisce le procedure ed i protocolli di colloquio tra due nodi ATM.

In Figura 1.1 viene rappresentata la struttura di una rete ATM evidenziando i due diversi tipi di interfacce che la compongono.

L'architettura protocollare dell'ATM è articolata in tre piani distinti:

- Piano d'utente (User Plane) comprende le funzioni relative al trattamento dei flussi informativi d'utente per il loro trasferimento in rete
- Piano di controllo comprende le funzioni di controllo e di segnalazione per l'instaurazione delle connessioni ATM;
- Piano di gestione gestisce le interazioni e coordina il lavoro fra il piano d'utente ed il piano di controllo

Il piano d'utente ed il piano di controllo sono articolati a loro volta in quattro strati:

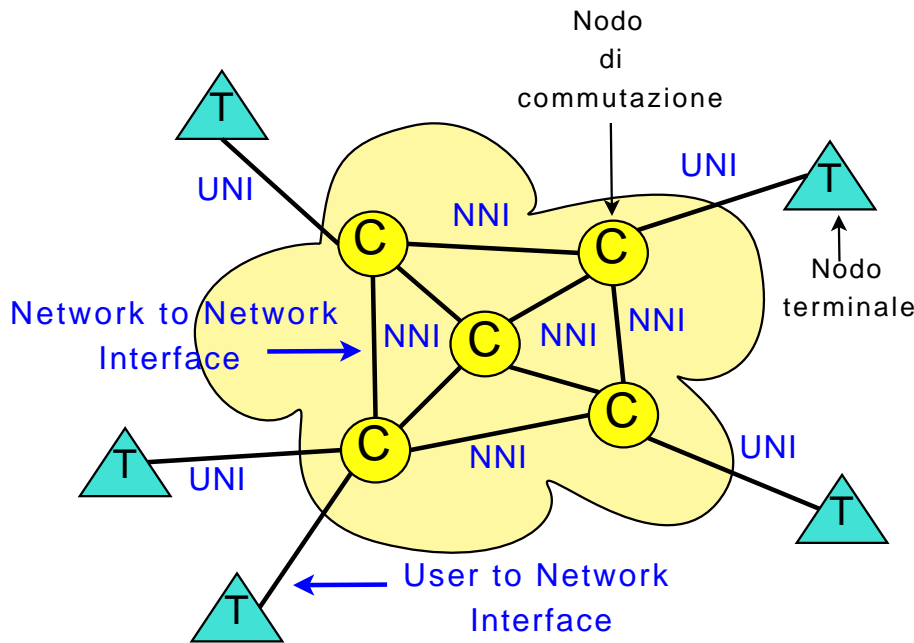


Figura 1.1: Struttura di una rete ATM.

- Strato fisico (Physical layer PH) questo strato è costituito dalle funzioni di gestione del mezzo trasmissivo e quelle riguardanti la trasmissione e la ricezione dei bit informativi; comprende inoltre le funzioni di adattamento delle celle alle caratteristiche specifiche dell'interfaccia trasmissiva; le funzionalità dello strato fisico sono eseguite in qualsiasi elemento di rete (terminale o nodo) e sono comuni al piano d'utente e di controllo;
- Strato ATM (ATM Layer) a questo livello risiedono le funzioni di trattamento delle celle, in particolare, appartengono a questo strato le funzioni di elaborazione delle informazioni di indirizzamento contenute nelle celle e le funzioni di commutazione di queste; queste funzioni costituiscono l'insieme delle funzioni comuni a tutte le applicazioni, qualsiasi altra funzione è specifica di una particolare applicazione e, come tale, non fa parte dello strato ATM; le funzionalità dello strato ATM sono eseguite in qualsiasi elemento di rete e sono comuni al piano d'utente e di controllo;
- Strato di Adattamento (ATM Adaption Layer AAL) il compito di questo strato è quello di provvedere all'interfacciamento dei servizi offerti dallo strato ATM ai requisiti posti dalla specifica applicazione; è evi-

dente come le funzioni di tale strato siano dipendenti dal tipo di applicazione e quindi queste sono eseguite solo dagli elementi esterni alla rete (terminali o nodi di accesso); l'insieme di queste funzioni viene differenziato secondo il piano d'utente ed il piano di controllo;

- Strati superiori (Higher layers) qui vengono implementate le funzioni esterne alla rete ATM, quali quelle dipendenti dalle caratteristiche dei protocolli posti al di sopra dello strato di adattamento; in particolare possono riguardare sia funzionalità applicative, nel caso in cui il terminale sia nativo ATM, oppure funzionalità di altri protocolli (IP, Frame Relay, LLC/MAC, ecc.) nel caso in cui la rete ATM sia connessa ad alte reti funzionanti secondo paradigmi diversi da quello ATM.

1.4 La connessione

L'ATM è un modo di trasferimento orientato alla connessione (connection oriented). Ciò significa che quando due sistemi terminali devono scambiarsi informazioni, tra questi due sistemi deve essere preliminarmente instaurata una connessione logica. Per tutta la durata della connessione, le celle attraverseranno la rete utilizzando lo stesso cammino individuato all'atto dell'instaurazione della connessione stessa. I vantaggi derivanti dal concetto di connessione sono i seguenti:

- semplicità di connessione, in quanto si evita che ogni nodo interno alla rete debba scegliere il cammino per ogni singola cella;
- possibilità di garantire predeterminati livelli di QoS, verificando che le risorse a disposizione siano sufficienti all'atto di instaurazione della connessione di rete

Le connessioni logiche in ATM possono essere di due tipi: permanenti o commutate. Una connessione viene detta permanente se è instaurata su base contrattuale dall'operatore di rete mediante funzionalità gestionali. Una connessione di questo tipo rimane normalmente attiva per un lungo periodo di tempo ed è legata alla durata del contratto tra utente e operatore di rete. Una connessione è di tipo commutato se viene instaurata automaticamente, in tempo reale, a seguito della richiesta specifica da parte di un utente. La durata della connessione dipende dal tempo necessario per lo scambio delle informazioni, al termine del quale la connessione viene abbattuta. La richiesta di instaurazione e di abbattimento sono presentate dagli utenti utilizzando le funzionalità ed il relativo protocollo di segnalazione. Un altro criterio

di classificazione si basa sulla molteplicità dei sistemi interessati dalla connessione. Una connessione punto-punto è definita come una relazione logica esclusivamente tra due utenti, mentre una connessione punto-multipunto è definita come una relazione logica tra un utente, che ha il ruolo di radice della connessione, e una molteplicità di utenti che assumono il ruolo di foglie della connessione. Una copia delle celle emesse dalla radice viene ricevuta da tutte le foglie appartenenti alla connessione.

1.5 Virtual Channel e Virtual Path

Una connessione ATM, indicata con il nome di Virtual Channel Connection (VCC), è definita come un canale logico tra due utenti finali utilizzata per il trasporto di celle. Una VCC è il risultato della concatenazione di uno o più canali virtuali (Virtual Channels VC); un VC descrive un trasferimento unidirezionale di celle ATM aventi lo stesso identificatore (Virtual Channel Identifier VCI). I punti terminali di un VCL (Virtual Circuit Link) possono essere costituiti da quei dispositivi che sono in grado di assegnare, modificare e terminare un VCI, ad esempio un terminale o un nodo ATM. Le relazioni fra gli elementi appena descritti è mostrata in Figura 1.2.

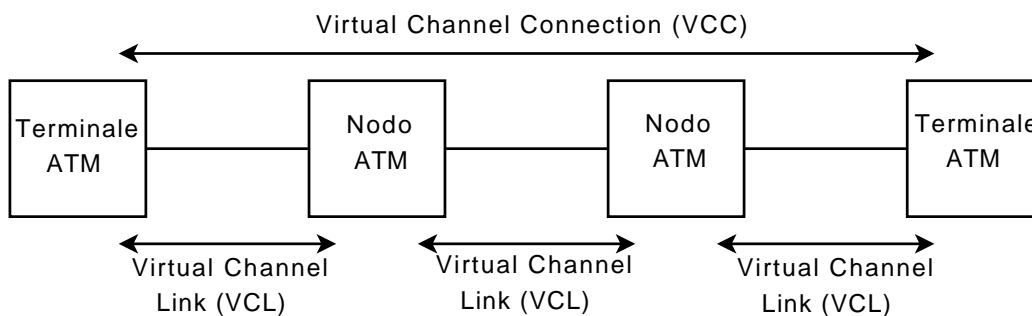


Figura 1.2: Definizione di Virtual Channel Connection (VCC) e Virtual Channel Link (VCL).

Accanto al concetto di VCC, è stato introdotto un secondo livello di moltiplicazione che ha lo scopo di introdurre la possibilità di trattare in modo aggregato un molteplicità di VC; questo secondo livello di moltiplicazione corrisponde al concetto di cammino virtuale (Virtual Path VP). Ogni VC è associato ad un VP e una molteplicità di VC può essere associata allo stesso VP. Le celle appartenenti a VC associati allo stesso VP trasporteranno lo stesso identificatore denominato Virtual Path Identifier (VPI).

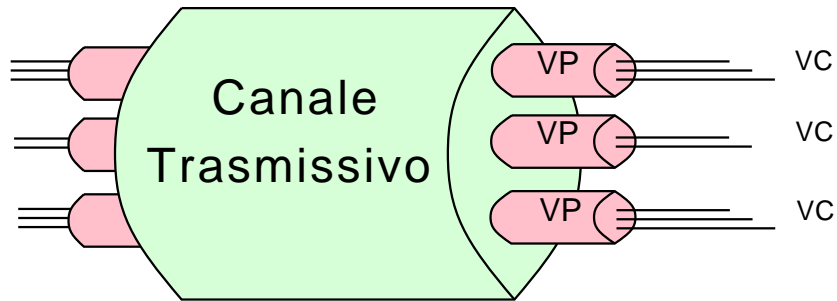


Figura 1.3: Gerarchia delle connessioni virtuali di ATM.

Analogamente a quanto definito per il concetto di VC, una Virtual Path Connection (VPC) è definita come un insieme di VCL caratterizzati dagli stessi punti terminali. In questo modo tutte le celle appartenenti ad uno qualsiasi dei VCL che fanno parte di un VPC seguiranno in rete lo stesso cammino tra i punti terminali delle connessioni. La tratta di rete in cui è valido un VPI viene indicata con il nome di Virtual Path Link (VPL). I punti terminali di un VPL possono essere costituiti da quei dispositivi che sono in grado di assegnare, modificare e terminare un VPI, ad esempio un terminale o un nodo ATM.

Il concetto di VP è stato definito per affrontare il problema dell'aumento del costo del controllo di rete che cresce proporzionalmente al crescere del numero delle connessioni presenti nella rete stessa. La definizione dei VP consente infatti di contenere il costo di gestione della rete poiché permette il controllo simultaneo di gruppi di connessioni. I vantaggi connessi alla definizione di VP possono essere così riassunti:

- semplificazione dell'architettura di rete; le funzioni di trasporto possono essere separate in quelle relative alle connessioni logiche individuali (VC) e quelle associate ad insiemi di connessioni logiche (VP);
- aumento delle prestazioni e dell'affidabilità; la rete è in grado di trattare un numero minore di entità aggregate;
- riduzione del carico elaborativo nei nodi e riduzione del tempo di set-up delle connessioni; la maggior parte delle funzioni di controllo viene svolta all'atto di instaurazione del cammino virtuale; nuovi VC possono essere stabiliti all'interno di un VP già instaurato in modo semplificato semplicemente operando ai punti terminali di una VPC, nessuna elaborazione è necessaria nei nodi intermedi della VPC.

I punti terminali di una connessione possono essere sia i terminali d'utente

sia delle entità di rete ma anche una combinazioni di questi. Una connessione può essere quindi utilizzata in una delle seguenti modalità:

- Connessione tra sistemi d'utente; è utilizzata per il trasferimento da estremo a estremo di dati utente; nel caso di VPC da estremo a estremo, questa offre una capacità complessiva dedicata al trasferimento di dati all'interno della quale i due utenti possono quindi instaurare delle singole VCC per il trasferimento di singoli flussi di informazioni;
- Connessione tra un sistema d'utente e una entità di rete; viene normalmente utilizzata per il trasferimento di informazioni di controllo tra utente e rete; una VPC di questo genere può essere utilizzata per aggregare il traffico di controllo da un utente verso uno specifico elemento di rete, ad esempio un nodo o un server.
- Connessione tra due entità di rete; è utilizzata per lo scambio tra elementi di rete di informazioni di controllo e di gestione.

Ad ogni connessione ATM (VCC o VPC) sono associate i seguenti parametri:

- Quality of Service (QoS), per ogni connessione deve essere specificato il valore di un insieme di parametri che definiscono la qualità del servizio che caratterizza il trasferimento delle celle appartenenti alla connessione stessa, ad esempio il valore del ritardo massimo e il valore della probabilità massima di perdita delle celle;
- Integrità della sequenzialità delle celle, una rete ATM preserva la sequenza delle celle trasmesse nell'ambito di una connessione;
- Negoziazione dei parametri di traffico e monitoraggio dell'uso delle risorse; per ogni connessione possono essere negoziati, tra utente e rete, i parametri del traffico relativi all'utilizzazione della connessione stessa, ad esempio tasso di picco e medio di generazione delle celle; durante la durata della connessione la rete provvederà a controllare che il traffico immesso dall'utente sia conforme ai parametri di traffico associati alla connessione.

1.6 La cella ATM

La lunghezza delle celle ATM è costante e uguale a 53 byte, di cui 5 byte sono riservati all'header e 48 byte al payload.

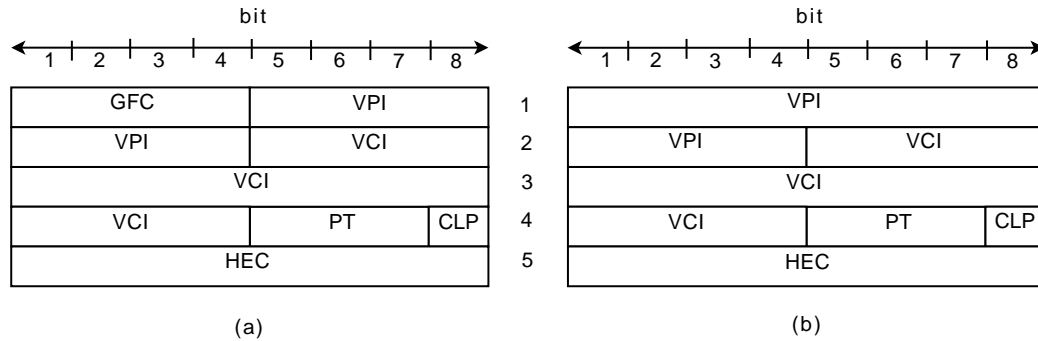


Figura 1.4: Formato di una cella ATM all'interfacci UNI (a) e all'interfaccia NNI (b).

Il vantaggio della scelta di una lunghezza costante per le celle risiede nel fatto che questa comporta una sensibile semplificazione delle operazioni di commutazione permettendo una loro completa realizzazione in hardware. La scelta di un formato di payload uguale a 48 byte è il frutto di un compromesso tra due esigenze contrastanti, quella di ridurre il più possibile il tempo della formazione delle celle da parte delle sorgenti a basso bit rate di emissione (ad esempio le sorgenti vocali) e quella di mantenere la percentuale di overhead entro limiti accettabili. Il formato delle celle ATM in corrispondenza delle interfacce UNI e NNI viene descritto brevemente in seguito:

- Generic Flow Control (GFC) è composto da 4 bit ed appare esclusivamente nella cella dell'interfaccia UNI; il suo scopo è quello di realizzare un meccanismo di controllo di accesso all'interfaccia utente rete; la configurazione di default di questo campo, nel caso in cui nessun protocollo d'accesso sia utilizzato, è 0000;
- Virtual Path Identifier (VPI) questo campo ha una lunghezza di 8 bit all'interfaccia UNI e di 12 bit all'interfaccia NNI; contiene l'identificatore del Virtual Path a cui appartiene la cella;
- Virtual Channel Identifier (VCI) la lunghezza di questo campo è di 16 bit indipendentemente dal tipo di interfaccia e trasporta l'identificatore del Virtual Circuit;
- Payload Type (PT) è lungo 3 bit e codifica il tipo di informazione trasportata, d'utente, informazione di gestione, controllo del traffico;
- Cell Loss Priority (CLP) è un bit che indica la priorità di scarto della cella in caso di congestione;

- Header Error Control (HEC) questo campo ha una lunghezza di 8 bit e viene utilizzato dallo strato fisico per rilevare e possibilmente correggere errori nell'header della cella.

1.7 La commutazione ATM

La commutazione delle celle in un nodo ATM è basata sul principio della conversione d'etichetta (label swapping). Per ogni cella entrante, ogni nodo ATM estrae dall'header il valore dei campi VPI e VCI, che identificano la connessione a cui appartiene la cella. Successivamente il nodo accede ad una tabella interna, denominata routing table, in cui, a partire dagli elementi VPI, VCI e identificatore della porta di ingresso, viene individuata la corrispondenza con l'identificatore della porta di uscita su cui deve essere rilanciata la cella e i valori dei campi VPI/VCI validi sulla porta di output. Questi ultimi saranno copiati nei rispettivi campi della cella prima di essere commutata sulla uscita che le corrisponde.

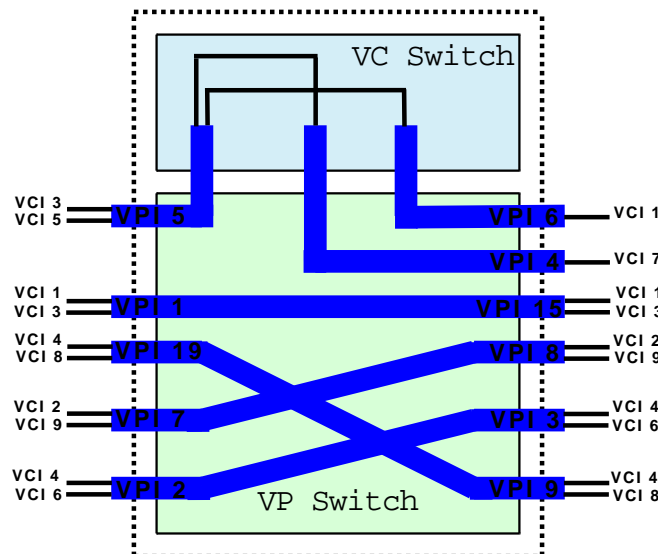


Figura 1.5: Tipi di commutazione di uno switch ATM; la parte superiore rappresenta uno switching a livello di circuito mentre la parte inferiore lo switching di cammino.

Questa associazione viene inserita nella routing table di ogni nodo all'atto dell'instaurazione della connessione. La tecnica della traslazione di etichetta è particolarmente efficiente. Infatti, poichè utilizza identificatori di lunghezza

molto limitata, è possibile una sua realizzazione completamente in hardware che permette di ridurre i tempi di commutazione e quindi favorisce il trattamento di comunicazioni con requisiti stringenti di ritardo.

Poichè l'identificatore di connessione in ATM è costituito dai due campi VPI e VCI che corrispondono a due livelli di multiplazione, si possono individuare due livelli di commutazione:

- **VP Switching**
Il nodo che esegue le funzioni di VP Switching termina un VPL ed esegue esclusivamente le funzioni di traslazione del VPI, il valore del campo VCI rimane inalterato.
- **VC Switching**
Il nodo che esegue le funzioni di VC Switching termina un VCL e quindi anche un VPL ed esegue quindi le funzioni di traslazione sia del VPI che del VCI.

1.8 Gli strati dell'architettura ATM

La Figura 1.5 indica la suddivisione in sottostrati dell'architettura ATM e riassume le funzioni eseguite da ogni sottostrato.

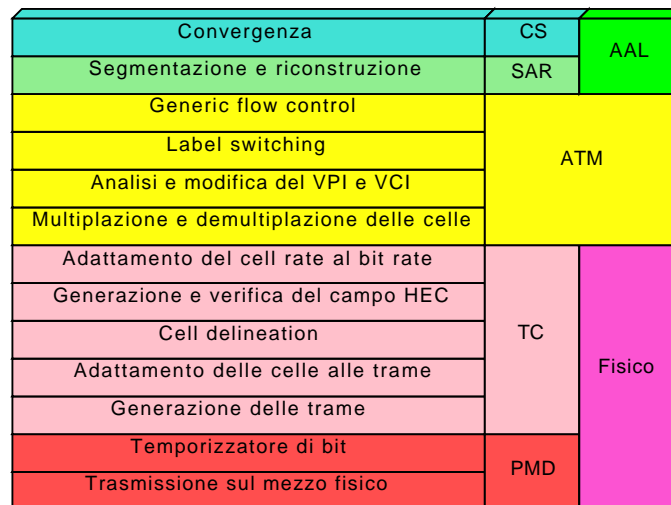


Figura 1.6: Architettura dei protocolli ATM.

1.9 Lo strato fisico

Lo strato fisico è lo strato responsabile della trasmissione delle celle attraverso le interfacce di rete. Lo strato Physical Layer (PH) è suddiviso in due sottostrati:

- Physical Medium Dependent (PMD)
Questo sottostrato è dipendente dallo specifico mezzo fisico utilizzato ed esegue le funzioni di inserimento ed estrazione delle informazioni di sincronizzazione e realizza la necessaria codifica di linea. Il sottostrato PMD non gestisce le celle, ma un flusso continuo di bit tra i due lati dell'interfaccia di rete.
- Transmission Convergence (TC)
Il sottostrato TC si colloca immediatamente al di sotto dello strato ATM ed ha il compito di effettuare, lato immissione, le funzioni necessarie alla trasmissione delle celle sul mezzo trasmissivo e, lato ricezione, le funzioni necessarie alla rivelazione e alla estrazione delle celle dal flusso binario entrante.

Durante la trasmissione lo strato PMD svolge operazioni di rilevanza riguardo alla corretta immissione di traffico ATM sul particolare tipo di mezzo trasmissivo. Siccome il flusso di dati nello strato fisico è strutturato in trame, questo strato si incarica di supervisionare la generazione delle trame ad uno specifico bit rate in trasmissione. In seguito ha luogo il processo di *transmission frame adaption*, ovvero l'inserimento e l'adattamento del flusso di celle provenienti dallo strato ATM nella struttura della trama in trasmissione. Prima di inserire le celle nelle trame questo strato si occupa della generazione del campo HEC, calcolato sull'intero header della cella. Un'altra funzione svolta in fase trasmissiva è l'adattamento del tasso di emissione delle celle, ovvero l'inserimento di celle vuote¹, nella trama allo scopo di adattare il tasso di emissione delle celle alla capacità della linea.

La ricezione a questo livello prevede l'attuazione dell'algoritmo di *cell delineation*, il quale permette di rilevare i confini delle celle basandosi sul riconoscimento del campo HEC e di estrarre le celle dal flusso delle trame; il campo HEC viene utilizzato quindi per l'identificazione e l'eventuale correzione degli errori. Le celle idle immesse nel flusso vengono estratte ed eliminate.

¹idle cells, ovvero celle che non trasportano nessuna informazione utile, la cui unica funzione è quella di adattamento al flusso di trame dello strato fisico.

Gli organismi di standardizzazione (ITU e ATM Forum) hanno negli anni definito un insieme di interfacce fisiche adatte alle diverse applicazioni possibili dell'ATM. Tali interfacce si differenziano essenzialmente per il bit rate (da 1.544 Mbit/s a 622 Mbit/s) e per il tipo di mezzo trasmissivo utilizzato, ad esempio doppino non schermato (UTP-3), doppino schermato (UTP-5), cavo coassiale e fibra ottica singolo modo (SMF) o multimodo (MMF). Una ulteriore differenza consiste nella diversa struttura utilizzata per la trasmissione delle celle. Alcune interfacce utilizzano una trasmissione delle celle organizzate in trame (interfaccia frame-based), in altre la trasmissione delle celle è di tipo continuo (interfaccia cell-based).

Nelle interfacce frame based l'asse dei tempi è suddiviso in periodi di durata costante uguale a $125 \mu\text{sec}$, denominati trame. La lunghezza della trama dipende dalla capacità dell'interfaccia trasmissiva, ad esempio nel caso dell'interfaccia E1 a 2.048 Mbps, la lunghezza della trama è uguale a 256 bit, ovvero 32 byte. Una trama è composta da una sezione di overhead, che contiene le informazioni necessarie all'allineamento e all'esecuzione delle specifiche funzioni di segnalazione e di esercizio e manutenzione, e un payload che contiene le informazioni d'utente. Nelle interfacce frame-based le celle ATM sono trasportate all'interno del payload delle trame. Nelle interfacce cell-based non esiste alcuna struttura di trama e la trasmissione delle celle è continua.

Ogni cella ATM include nell'intestazione il campo HEC dedicato alla protezione dagli errori dei 32 bit dell'header. Il codice polinomiale utilizzato per generare il campo HEC è $X^8 + X^2 + X + 1$. Poiché la lunghezza del codice HEC (8 bit) è percentualmente rilevante rispetto alla stringa di bit che deve proteggere (32 bit), è possibile realizzare sia la funzione di rivelazione di errore sia in alcuni casi la funzione di correzione d'errore. All'istante di

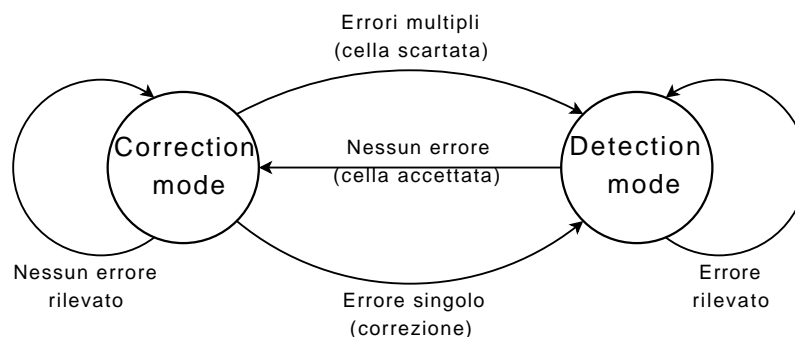


Figura 1.7: Rivelazione e correzione degli errori nell'header della cella ATM.

attivazione, l'algoritmo si trova nello stato *correction mode* in cui, sfruttando la ridondanza del codice HEC, si è in grado, oltre che di rilevare gli errori, anche di correggere gli errori singoli presenti nell'header. Il controllo del campo HEC avviene per ogni cella entrante e, fino a che non viene rilevata una cella che contiene un errore nell'header, l'algoritmo permane nello stato di correzione. Quando viene rilevato un errore, se l'errore è singolo l'algoritmo provvede a correggere l'errore e quindi ad accettare la cella, se invece l'errore interessa una molteplicità di bit la cella è scartata. In entrambi i casi l'algoritmo effettua la transizione verso lo stato *detection mode* in cui è attiva la sola funzione di rivelazione d'errore e non quella di correzione. Quindi se in questo stato viene ricevuta una cella contenente uno o più errori nell'header, questa è sempre scartata. La ragione del fatto che la funzione di correzione è disattivata nello stato detection mode dipende dal fatto che nel caso si presentasse un burst di errori che interessi una sequenza di celle, il codice HEC sarebbe insufficiente a realizzare una affidabile funzione di correzione. Il ricevitore rimane nello stato detection mode fino a che non viene ricevuta una cella corretta, momento in cui il ricevitore effettua una transizione all'indietro verso lo stato correction mode.

L'algoritmo di delimitazione di cella permette al ricevitore di individuare, all'interno del flusso di bit ricevuto, i raggruppamenti di 424 bit che costituiscono la cella ATM. Questa funzione è realizzata congiuntamente con la funzione di elaborazione del campo HEC.

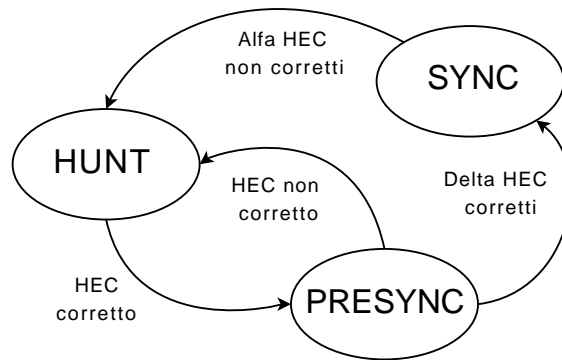


Figura 1.8: Stati dell'algoritmo di cell delineation.

1. Nello stato HUNT, l'algoritmo di cell delineation è eseguito bit a bit, per individuare in corrispondenza di quale bit è verificata la regola di codifica del campo HEC, ovvero c'è uguaglianza tra la configurazione dell'ultimo ottetto ricevuto e quella calcolata in base ai precedenti 32 bit; quando si

ottiene questa corrispondenza, l'algoritmo assume che sia stato individuato l'header di una cella e l'algoritmo entra nello stato di PRESYNC; tutti i bit ricevuti nello stato HUNT sono scartati a causa del fatto che le celle non possono essere riconosciute.

2. Nello stato PRESYNC l'algoritmo assume che la struttura di cella sia stata individuata e l'algoritmo di cell delineation viene eseguito cella per cella (a distanza di 424 bit); se la regola di codifica dell'HEC è verificata per delta volte consecutive, l'algoritmo entra nello stato SYNCH; se prima di raggiungere il valore delta la regola di codificazione dell'HEC non è verificata l'algoritmo ritorna nello stato HUNT; i raggruppamenti di cifre delineati nello stato PRESYNC sono comunque scartati poichè non è raggiunta la sicurezza che questi corrispondano effettivamente alle celle.

3. Nello stato SYNCH si assume che le celle siano effettivamente delineate correttamente, quindi il campo HEC è utilizzato per la rivelazione e la correzione degli errori sull'header; la sincronizzazione di cella si considera persa se per alfa volte consecutive il campo HEC rivela errori; questo evento è infatti attribuito ad un errore di delimitazione piuttosto che ad una sequenza di errori trasmissivi. I valori di alfa e delta devono essere fissati raggiungendo il migliore compromesso tra esigenze contrastanti. Tanto più è elevato il valore di delta tanto più grandi sono i tempi di recupero della condizione di sincronizzazione, ma tanto più robusto è l'algoritmo contro false sincronizzazioni. Allo stesso modo, tanto maggiori sono i valori di alfa, tanto maggiori divengono i tempi di rivelazione degli eventi di perdita di sincronizzazione, ma più grande sarebbe la robustezza contro gli eventi di falsa perdita di sincronizzazione. I risultati di studi e misure sulle prestazioni dell'algoritmo suggeriscono valori di alfa e beta a qualche unità.

1.10 Lo strato ATM

Il secondo strato dell'architettura protocollare è lo strato ATM ed è indipendente sia dallo strato fisico che dallo strato direttamente superiore, ovvero quello di adattamento.

Lo strato ATM ha il compito di effettuare il trasferimento delle celle tra le entità di rete e la consegna in sequenza delle celle ai sistemi ATM finali.

In corrispondenza del punto di origine della VCC, lo strato ATM:

1. riceve dallo strato di adattamento i 48 byte che compongono il payload di ogni cella;

2. determina l'appropriato valore del VPI e VCI e aggiunge 4 byte dell'header (escluso il byte HEC);
3. consegna la cella allo strato fisico che provvederà al calcolo del campo HEC e alla trasmissione sull'interfaccia fisica.

In corrispondenza del punto di terminazione della VCC, lo strato ATM:

1. riceve la sequenza di celle dallo strato fisico, che ha eseguito la funzione di controllo di correttezza dell'header mediante il campo HEC;
2. rimuove l'header delle celle, eseguendo eventualmente funzioni richieste;
3. consegna il payload allo strato di adattamento.

In corrispondenza di un nodo interno della rete, lo strato ATM è lo strato più alto dell'architettura e quindi non esiste lo strato di adattamento a cui deve offrire servizio. Le funzioni dello strato ATM in questo caso corrispondono alle funzioni di commutazione e di traslazione di etichetta.

Quindi un nodo ATM:

1. riceve la cella dell'entità di strato fisico relativa all'interfaccia di ingresso;
2. legge i valori di VPI/VCI contenuti nell'header;
3. effettua la lettura della routing table in cui è memorizzata l'associazione tra la coppia VPI/VCI uscenti e interfaccia di uscita;
4. effettua la traslazione di etichetta con i nuovi valori dei campi VPI/VCI, il trasferimento interno della cella verso l'interfaccia d'uscita e il suo inserimento nel relativo buffer di emissione;
5. consegna la cella allo strato fisico che provvederà al calcolo del campo HEC e alla trasmissione sull'interfaccia fisica.

Il servizio offerto da ATM non è perfettamente affidabile, non è garantito infatti che una cella sia consegnata a destinazione. Il servizio di trasferimento offerto dallo strato ATM è quindi caratterizzato da una probabilità non nulla di perdita di celle. I fattori che possono causare la perdita sono essenzialmente la presenza di errori non correggibili nell'header e la saturazione dei buffer di trasmissione dei nodi interni di una rete congestionata.

Nello strato ATM non viene definito nessun meccanismo di ritrasmissione delle celle perse, spetta agli strati superiori quindi rilevare questi eventi ed

eventualmente attivare le opportune procedure di recupero delle perdite. La reazione ad un evento di perdita dipende dal tipo di informazione che deve essere trasferita; nel caso di comunicazioni dati ad esempio, è evidente che i protocolli degli strati superiori devono provvedere a richiedere la ritrasmissione dei dati perduti mentre, nel caso di servizi vocali, la ritrasmissione non è necessaria.

Lo strato ATM tuttavia è in grado di garantire una fissata probabilità di perdita massima delle celle trasmesse nell'ambito di una connessione. Questo valore fa parte dei parametri di QoS che vengono negoziati tra utente e rete all'atto di instaurazione di una connessione e che la rete si impegna a soddisfare per tutta la sua durata.

Capitolo 2

Il modello dei dispositivi

Nel design e sviluppo di un sistema di gestione di dispositivi ATM occorre focalizzarsi sul modo di rappresentare logicamente un insieme di oggetti gerarchicamente distinti, tali come possono essere i dispositivi e lo stesso switch ATM. Sorge la necessità di sviluppare un modello coerente degli oggetti che giocano un ruolo fondamentale all'interno dell'ATM layer; affinché le relazioni semantiche degli elementi studiati possano essere correttamente definite e implementate si deve essere in grado di costruire un sistema di gestione e controllo di questi elementi. Questo progetto è stato sviluppato per sistemi embedded sui quali gira un sistema operativo Linux e pertanto l'approccio iniziale è stato quello di usare come riferimento il modello dei device drivers del kernel di Linux.

A partire dalla versione 2.5 del kernel, Linux semplifica e adotta una nuova infrastruttura di gestione dei dispositivi e dei relativi driver, allo scopo di offrire uno strumento generale ed efficiente per operare su ogni tipo di dispositivo. Questa nuova infrastruttura ha reso possibile l'eliminazione di codice e strutture dati ridondanti, soprattutto nell'area della gestione del conteggio dei riferimenti e del trattamento delle informazioni di linkage fra liste e gruppi di oggetti fra loro relazionati.

Le definizioni degli oggetti di questo modello si trovano nel file *include/linux/device.h*, mentre le operazioni che permettono di operare su ogni tipo di oggetto sono implementate in *driver/base*. Nel file di header vengono presentati gli elementi fondamentali del modello, si trovano infatti le dichiarazioni delle strutture *struct device*, *struct device_driver*, *struct class*, *struct class_device* e *struct bus_type*.

Nonostante le naturali differenze esistenti fra oggetti eterogenei e le loro relative funzionalità ci sono alcune linee comuni che possono essere evidenzi-

te nel core del driver model e forniscono una prospettiva del funzionamento generale di questo sistema.

Ogni oggetto ha associate le funzioni *register* e *unregister* specifiche per quel tipo di oggetto. La registrazione inizializza l'oggetto e lo inserisce in una lista di oggetti simili creando una directory per l'oggetto nel filesystem *sysfs*. Viene effettuata una registrazione ogni qualvolta viene rilevato un dispositivo, o quando un modulo contenente un dispositivo viene inserito.

La deregistrazione viene eseguita a seguito della rimozione di un modulo di un driver, o anche quando un dispositivo viene fisicamente rimosso e come conseguenza si ha la rimozione della directory nel *sysfs* e la cancellazione dell'oggetto dalla lista o liste in cui era stato precedentemente inserito.

Al fine di supportare la rimozione dinamica, ogni oggetto contiene un conteggio dei riferimenti che può essere manipolato tramite le specifiche funzioni *get()* e *put()*.

Il metodo *get* dovrebbe essere chiamato previamente ogni volta che si vuole usare un determinato oggetto all'interno di una funzione al fine di ottenere un riferimento valido all'oggetto su cui operare. Il metodo *put* serve per rilasciare il riferimento acquisito dopo aver terminato di utilizzare l'oggetto.

Se, e solo se, il conteggio dei riferimenti per un dato oggetto raggiunge il valore 0, viene permesso di liberare le risorse dell'oggetto, riuscendo in questo modo a prevenire l'insorgere di spiacevoli situazioni di incoerenza. Infatti, potrebbe succedere che un altro processo conservi un riferimento ad un oggetto dopo che su questo venga chiamata la funzione *unregister* e, se la memoria allocata per l'oggetto venisse liberata, l'accesso al suo riferimento non sarebbe valido o avverrebbe su memoria reallocata con evidenti conseguenze disastrose.

2.1 I dispositivi

Un device è un componente fisico che offre un determinato insieme di funzionalità sulle quali il kernel può esercitare il suo controllo. Un dispositivo risiede su un bus di un certo tipo mentre ogni bus definisce una architettura standard per i dispositivi che vi risiedono. Il kernel organizza i dispositivi che è in grado di riconoscere secondo il bus al quale sono collegati.

I dispositivi vengono organizzati in un sotto sistema dei device, che il kernel provvede a pubblicare nella directory */sys/devices*. Il modello dei dispositivi definisce la *struct device* al fine di descrivere un device come un

oggetto astratto, indipendentemente dal bus che lo ospita o dalle funzionalità che offre. Questa descrizione astratta contiene solo pochi campi relazionati direttamente agli attributi fisici del dispositivo; il suo obiettivo è quello di delineare in un descrittore comune gli aspetti simili delle differenti rappresentazioni dei dispositivi.

Ogni dispositivo viene quindi rappresentato da un'istanza della struttura `struct device`. I campi di una `struct device` possono essere classificati in tre categorie di base.

Attributi fisici

I campi che descrivono le caratteristiche fisiche del device sono pochi, dato che pochi sono i parametri fisici comuni alla maggioranza dei dispositivi; quelli presenti, riguardano la gestione delle transizioni del *power state* del dispositivo.

Linkage

I dispositivi fanno parte di tre tipi di liste differenti: una lista di dispositivi appartenenti ad uno specifico bus, una lista secondo il driver usato ed infine una lista per associare i dispositivi ai descrittori di interfacce che essi possono implementare. I punti di entrata a queste liste sono costituiti da campi di tipo `struct klist_node`. Mediante il campo `knode_parent`, il dispositivo entra a far parte di una lista di dispositivi accomunati dallo stesso parent. I dispositivi, infatti, sono organizzati gerarchicamente secondo le dipendenze esistenti fra i vari elementi del sistema e, come riflesso, la struttura della directory `/sys/device/` coincide con l'organizzazione fisica dei dispositivi hardware. Il campo `knode_driver` viene utilizzato dal driver che gestisce il dispositivo per raccogliere tutti i tipi di dispositivi che controlla. Infine, ogni tipo di bus memorizza i dispositivi che vi risiedono fisicamente; il campo `knode_bus` dell'oggetto device memorizza i puntatori agli elementi adiacenti nella lista del bus. La struttura dati utilizzata, la `struct klist_node`, è una particolare implementazione wrapper della rappresentazione elementare che il kernel offre; alle caratteristiche di rapida ed efficiente gestione degli oggetti di una lista, viene affiancato il controllo di sincronizzazione fra accessi concorrenti inserendo uno spinlock di protezione.

Meta dati

I meta dati sono un insieme di informazioni che costituiscono un mezzo di gestione e controllo dell'oggetto che descrivono. Tutte queste informazioni

sono racchiuse in una struttura molto ricorrente in questo modello, la *struct kobject*. Ogni elemento del device driver model può essere visto come un container al cui interno un *kobject embedded* offre le seguenti funzionalità:

1. conteggio dei riferimenti attivi verso l'oggetto;
2. mantenimento di liste di oggetti;
3. protezione degli insiemi di oggetti;
4. rappresentazione a livello di userspace.

Inizialmente fu pensato solamente come un meccanismo di conteggio dei riferimenti, ma in seguito le sue responsabilità sono aumentate.

Esiste una stretta relazione fra i *kobject* ed il filesystem *sysfs*. Ogni oggetto mostrato nel *sysfs* filesystem nasconde dietro di sé un *kobject* registrato nel core dei *kobject*, che, interagendo con il kernel, rende possibile la sua rappresentazione e l'esportazione dei suoi attributi nello userspace.

Una altra funzionalità dei *kobject* è quella di agire da punto di unione nel complicato groviglio delle numerose strutture eterogenee esistenti nel sistema; infatti il sistema dei dispositivi altro non è che una complessa rete di oggetti appartenenti a gerarchie differenti e tra loro collegati proprio mediante dei *kobject*.

I *kobject* inoltre, sono il mezzo tramite il quale i dispositivi (e gli altri oggetti in generale) supportano la registrazione e deregistrazione dinamica all'interno del rispettivo subsistema. Quando si registra un oggetto, il suo *kobject* associato viene inizializzato e inserito nella lista degli oggetti del sotto sistema al quale appartiene.

2.2 I driver

Il kernel interagisce con i dispositivi attraverso i device driver. I driver sono delle parti del codice del kernel costituiti dalle definizioni di strutture dati e funzioni il cui intento è quello di guidare il controllo del kernel sui dispositivi. Il controllo si ottiene con l'implementazione delle interfacce programmabili dei dispositivi. Ogni driver comunica con il kernel mediante specifiche interfacce. L'approccio di accesso alle funzionalità di un dispositivo mediante interfaccia comporta alcuni vantaggi:

- il codice specifico di un dispositivo può essere incapsulato all'interno di un modulo specifico;

- esiste la possibilità di aggiungere nuovi dispositivi senza conoscere il codice sorgente del kernel, dato che risulta necessario accedere solamente alle interfacce.;
- il kernel tratta tutti i dispositivi in maniera uniforme accedendo ad essi tramite la stessa interfaccia;
- è possibile scrivere un driver come un modulo che può essere caricato dinamicamente nel kernel senza dover necessariamente effettuare un reboot del sistema, così come dinamicamente può venire rimosso al fine di minimizzare la dimensione dell'immagine del kernel nella RAM.

Ogni driver all'interno del modello dei device driver viene descritto da una *struct device_driver*. Queste strutture sono allocate staticamente e inizializzate al momento della loro registrazione.

Quando un driver viene registrato, viene associato al bus che dispone dei dispositivi sui quali il driver eserciterà il controllo. Con la registrazione del driver viene creata una directory che gli corrisponde nella directory *sysfs* del suo bus. In questa directory il driver può esportare allo userspace le funzioni di gestione e controllo che gli competono.

I driver che vengono compilati staticamente nel kernel sono registrati nella fase di inizializzazione del kernel, mentre quando vengono compilati come moduli la registrazione ha luogo all'atto di caricamento del modulo.

Capitolo 3

I dispositivi ATM

Ogni comunicazione che avviene all'interno di una rete implica l'esistenza di un flusso di informazioni trasportato su di un mezzo fisico. Il mezzo che veicola l'informazione è accessibile mediante uno specifico adattatore di rete che dipende dal tipo di collegamento fisico al quale è associato. Le funzionalità di una scheda di rete e del mezzo fisico che è in grado di controllare vengono mappate negli strati 1 e 2 del modello di protocolli ISO/OSI.

L'elemento che si incarica di gestire l'interfacciamento fra aspetti hardware di una comunicazione e le istanze dei protocolli di rete superiori è il dispositivo di rete. Il dispositivo di rete è un concetto usato nell'architettura dei sistemi operativi Unix al fine di astrarre dalle caratteristiche tecniche delle schede di rete fornendo un'interfaccia uniforme di accesso al mezzo fisico da parte dei protocolli superiori.

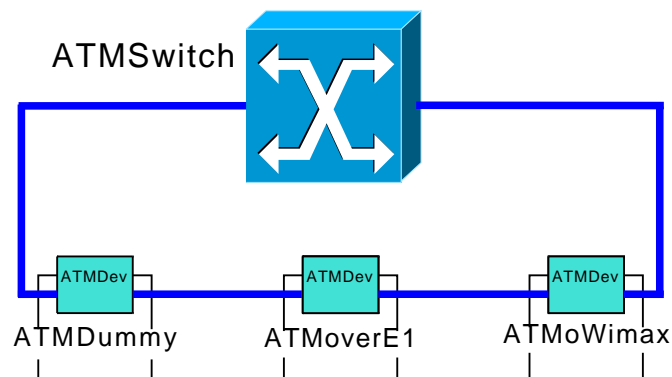


Figura 3.1: Schema delle interfacce utilizzate dal sistema ATM come mezzo di astrazione delle caratteristiche specifiche degli strati fisici inferiori.

Un dispositivo ATM è un dispositivo di rete capace di comprendere il protocollo ATM e di supportare la ricezione e la trasmissione di traffico ATM. L'ATM Forum ha studiato e definito differenti interfacce fisiche per i dispositivi ATM; ciò significa che gli strati fisici sottostanti ai dispositivi ATM possono essere caratterizzati da proprietà fisiche di comunicazione profondamente differenti.

Inoltre, non è da escludere il caso in cui i dispositivi possano ovviare alla presenza di un mezzo fisico sottostante, implementando le classiche funzionalità di rete ad un livello puramente logico, come è il caso ad esempio del dispositivo ATMDummy.

I dispositivi ATM permettono di astrarre dal mezzo fisico sottostante implementando una interfaccia di accesso uniforme ai protocolli di livello superiore.

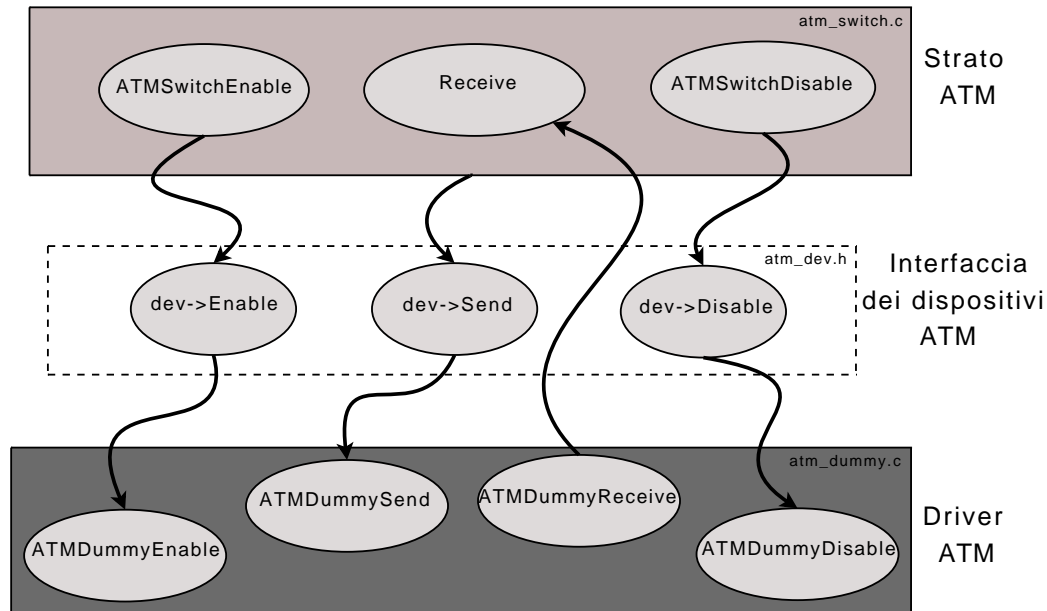


Figura 3.2: Interfacciamento di un dispositivo ATMDummy; il driver si occupa di inizializzare correttamente l'interfaccia associando le funzionalità richieste dallo strato superiore con quelle disponibili nello strato inferiore; lo switch ATM controlla il dispositivo ATMDummy tramite i metodi che gli vengono forniti dall'interfaccia.

La Figura 3.2 illustra l'architettura del sistema ATM evidenziando le relazioni esistenti fra lo switch e i dispositivi ATM. I dispositivi ATM vengono visti in maniera completamente uniforme dalla prospettiva dello switch, che

in questo caso risulta essere l'unico strato superiore della pila dei protocolli, essendo il sistema progettato per i nodi interni di una rete ATM.

I driver ATM sono costituiti da un insieme di metodi e porzioni di codice il cui ruolo è quello di far corrispondere specifici dispositivi hardware all'interfaccia programabile dei dispositivi ATM. Il compito del driver è quello di compaginare le funzionalità di un dispositivo ATM con le funzionalità offerte dallo strato fisico. Si occupa di effettuare le operazioni necessarie per la corretta inizializzazione e configurazione delle risorse hardware e software.

Un altro compito importante che assolve è quello di gestire il trattamento dell'informazione, occupandosi di gestire l'input e l'output asincrono di unità di dati appartenenti a protocolli differenti.

Nel caso di ATM over E1 il driver si incarica di bufferizzare le celle che gli vengono inviate dallo switch e mapparle correttamente sul flusso delle trame E1 in uscita, inserendo, se necessario, celle idle al fine di preservare il sincronismo del canale E1. In fase di ricezione invece, il suo compito è quello di sincronizzarsi con il flusso di trame E1 al fine di delineare correttamente le celle ATM in esso contenute; una volta stabilita la sincronizzazione mediante l'uso dello specifico algoritmo di cell delineation trasmette le celle estratte allo switch. Quando l'adattatore fisico non fornisce un supporto hardware per le operazioni relative al trattamento del campo HEC delle celle, il driver si deve occupare anche della generazione di questo campo prima di effettuare la trasmissione delle celle, e, in ricezione, di verificarne la correttezza per la correzione di errori singoli e la rilevazione di errori multipli.

3.1 Il disegno dell'interfaccia

L'interfaccia ATM è l'elemento fondamentale di ogni dispositivo ATM.

La sua definizione corrisponde alla definizione di una struttura dati che racchiude in sé le informazioni basilari relative ai protocolli superiori e allo strato fisico sottostante. Ogni dispositivo ATM è un particolare tipo di dispositivo che estende le sue funzionalità a quelle del protocollo ATM. Usando una definizione tipica dei linguaggi orientati agli oggetti si può affermare che tutti i dispositivi ATM ereditano un insieme comune di funzionalità dalla stessa classe base.

Tutto il codice del progetto è stato scritto in C.

Per definire le caratteristiche e i comportamenti degli oggetti in gioco nel sistema si usano le definizioni delle strutture. Ad ogni oggetto che si voglia

creare verrà associato un costruttore il cui compito è quello, se necessario, di riservare la memoria per tutto il tempo di vita dell'oggetto, e di eseguire le opportune inizializzazioni.

Esistono due modi per implementare l'ereditarietà in C.

Una maniera è quella di inserire un puntatore al tipo di oggetto dal quale si vuole ereditare, nella struttura dell'oggetto che rappresenta la specializzazione. Un'altra possibilità è quella di inserire, embedded, l'oggetto da cui si vuole ereditare direttamente dentro la struttura del nuovo oggetto. Quando possibile, si sceglierà di implementare l'ereditarietà per gli oggetti del sistema ATM seguendo la seconda modalità, al fine di poter effettuare la costruzione dell'oggetto completo con una sola chiamata alla funzione *kmalloc*, evitando una possibile frammentazione della memoria.

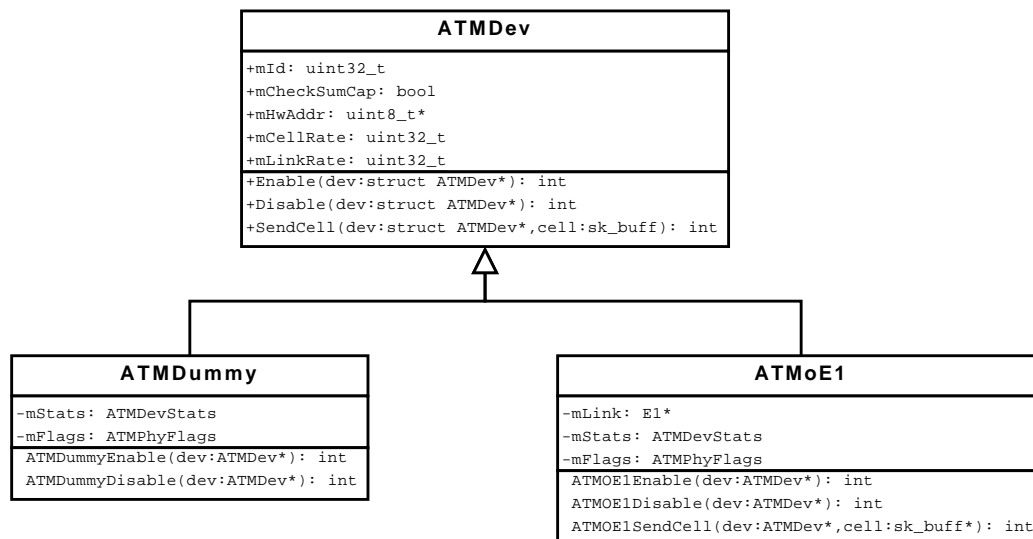


Figura 3.3: Diagramma delle classi relativo alla relazione di ereditarietà esistente fra i dispositivi **ATMDDev** ed i dispositivi **ATMDummy** e **ATMoE1**.

Nel file *atm_dev.h* si trova la definizione dell'intera classe base da cui tutti i dispositivi ATM ereditano. È contenuta la definizione della struttura che verrà inclusa in ogni dispositivo ATM e le definizioni dei metodi di costruzione e distruzione dell'oggetto **ATMDDev**. L'implementazione di questi metodi si trova nel file *atm_switch.c*, poichè è lo switch che si incarica di registrare i dispositivi ATM, effettuando le opportune inizializzazioni della struttura della classe base, la struttura *ATMDDev*. La Figura 3.3 utilizza un diagramma delle classi per evidenziare le componenti principali dei disposi-

tivi ATM trattati in questo progetto ed offre una rappresentazione grafica di come viene usata l'ereditarietà.

```
typedef struct ATMDev {
    ATMInfo mInfo;
    atomic_t mUsers;
    atomic_t mUsable;
    atomic_t mEnabled;
    wait_queue_head_t mQueue;
    uint32_t mNRuleRef;
    struct list_head mDevEntry;
    struct list_head mRuleList;
    struct class_device mClassDev;
    void *mFly;
    int (*Enable) (struct ATMDev *this);
    int (*Disable)(struct ATMDev *this);
    int (*SendCell)(struct ATMDev *this,alb_buff *cell);
    int (*SetExtraRoom)(struct ATMDev *this,int head,int tail);
    int (*GetExtraRoom)(struct ATMDev *this, int *head,int *tail);
    int (*GetStats)(struct ATMDev *this, ATMDevStats *stats);
    int (*ResetStats)(struct ATMDev *this);
    ATMPhySync (*AMTFlowSyncStatus)(struct ATMDev *this);
    int (*ATMFlowForceResync)(struct ATMDev *this);
    ATMPhyFlags (*GetFlags)(struct ATMDev *this);
    int (*SetFlags)(struct ATMDev *this,ATMPhyFlags flags);
} ATMDev;
```

La struttura *ATMInfo* racchiude dentro di sé tutte le informazioni che risultano di rilevante importanza per la gestione dei dispositivi da parte dello switch. Contiene i seguenti campi:

ID Identificatore unico del dispositivo all'interno dello strato ATM;

Checksum Capacità della scheda di rete di supportare la gestione degli errori;

HwAddr Indirizzo MAC del dispositivo fisico;

CellRate Tasso di ricezione e trasmissione delle celle ATM;

Link rate Link rate in Kbps.

L'ID viene assegnato ad ogni dispositivo nel momento in cui viene registrato come dispositivo ATM. La registrazione avviene invocando il costruttore *ATMDevConstructor*. Lo switch analizza la lista dei dispositivi registrati e provvede ad aggiungere il nuovo dispositivo assegnandoli un ID unico all'interno della lista. Per inserire il dispositivo nella lista, lo switch utilizza il campo *mDevEntry*. A partire da questo momento il dispositivo è pronto per essere abilitato e cominciare a trasmettere traffico ATM. Quando lo switch deciderà di attivare l'interfaccia fisica di un dispositivo registrato, chiamerà la funzione *Enable* su quel particolare dispositivo ed imposterà a true il campo *mEnabled*.

Ad ogni dispositivo ATM viene associata una lista di regole di switching. Si tratta di un accorgimento volto ad accelerare il processo di commutazione delle celle ATM. Il campo *mRuleList* rappresenta il punto di entrata alla lista di regole che si occupano di instradare il traffico proveniente da quel dispositivo. Un contatore tiene traccia del numero totale di regole associate al dispositivo.

I campi *mUsers*, *mUsable* ed *mQueue* sono utilizzati per fornire un meccanismo di sincronizzazione per gli accessi coerenti al dispositivo. Vengono utilizzati dallo switch per fornire un meccanismo di conteggio dei riferimenti simile a quello utilizzato nel device driver model con i *kobject*. Gli aspetti di gestione di questo sincronismo vengono presentati nella sezione 3.2.

I dispositivi ATM possono essere visti come una classe a sè stante di dispositivi e, al fine di offrire una visione esterna dello stato dell'oggetto, il campo *mClassDev* di tipo *struct class_device* viene usato per esportarne l'interfaccia nello user space aggiungendo una nuova directory nel sysfs, in */sys/class*.

Per una implementazione efficiente dell'astrazione che l'interfaccia *ATMDev* si propone di offrire, i dispositivi ATM adottano il meccanismo del polimorfismo, per unire i bisogni dello strato ATM sovrastante con le reali caratteristiche del particolare tipo di link fisico e delle caratteristiche dell'hardware della scheda di rete utilizzata. Per questa ragione, viene usato il concetto di puntatore a funzione, come strumento per l'implementazione del polimorfismo nel linguaggio C.

All'interno della struttura *ATMDev* troviamo una serie di puntatori a funzione che rappresentano le funzionalità tipiche di ogni dispositivo ATM. In fase di inizializzazione del dispositivo, è compito del driver inizializzare correttamente questa interfaccia polimorfica, implementando correttamente i metodi secondo le caratteristiche del particolare dispositivo fisico.

3.2 Reference counting

Il reference counting costituisce una tecnica di importanza cruciale il cui fine è quello di prevenire l'insorgere di memory leaks e accessi a memoria invalidi.

L'obiettivo è quello di garantire che la risorsa costituita dal dispositivo ATM non venga liberata fino a che ci siano dei processi attivi che potrebbero decidere di accedere ad essa. Si tiene traccia dei potenziali accessi al dispositivo attraverso l'uso della variabile *mUsers*. Come il nome stesso indica, in questa variabile vengono contati gli users del dispositivo.

Gli users sono costituiti dalle regole di switching memorizzate nello switch e da ogni cella che viene ricevuta dallo strato fisico. L'esistenza di una regola sul dispositivo o la presenza di celle in attesa di essere commutate nella coda dello switch, assicurano l'esistenza del dispositivo, impedendo che questo venga disabilitato o deregistrato mentre il sistema continua a fare uso delle sue funzionalità.

La funzione *ATMSwitchGetDev* permette di manipolare le variabili atomiche *mUsers* e *mUsable*. L'utilizzo di variabili atomiche si spiega con la necessità di proteggere le risorse condivise da più processi concorrenti.

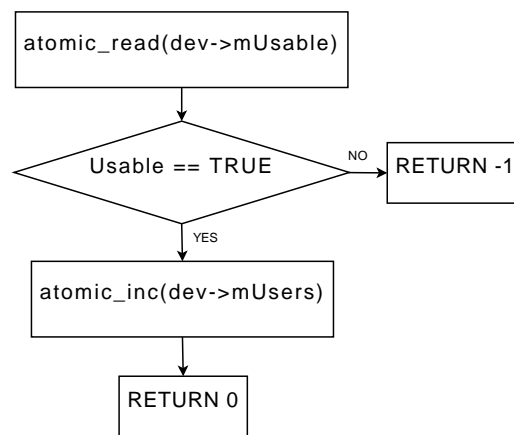


Figura 3.4: Get del dispositivo

Ogni entità che vuole utilizzare un dispositivo ATM deve incrementare il contatore degli users. La Figura 3.3 illustra il meccanismo di *getting* tramite il quale si ottiene un riferimento sicuro ad un dispositivo da poter utilizzare per tutto l'intervallo di tempo in cui il contatore di users per quel particolare dispositivo risulta essere maggiore di zero.

Anche il campo *mUsable* svolge un ruolo molto importante. In base al

valore che assume permette o nega l'accesso alla risorsa. Quando si vuole deregistrare un dispositivo, infatti, la prima operazione che deve essere eseguita è l'assegnazione del valore false a questa variabile. In questo modo, si evita che altri users ottengano dei riferimenti validi al dispositivo, rischiando di impedire per un periodo di tempo indeterminato la disabilitazione del dispositivo.

Ogni qualvolta che una regola di switching viene eliminata, o una cella commutata, si rilascia il riferimento al dispositivo. In termini operazionali ciò significa che il contatore degli users viene decrementato, e si effettua con la chiamata alla funzione *ATMSwitchPutDev*.

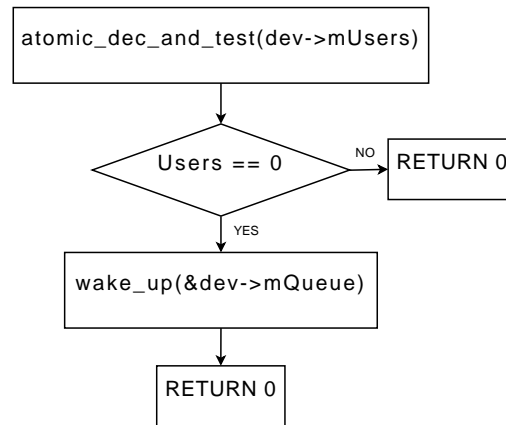


Figura 3.5: Put del dispositivo

Per garantire che un dispositivo venga disabilitato o deregistrato solamente nel momento opportuno, si utilizza una wait queue. Il campo *mQueue* rappresenta una coda di attesa dove i processi possono essere accumulati quando la loro esecuzione deve essere sospesa in attesa che si verifichi un determinato evento. Il processo che può trovarsi in uno stato di sospensione all'interno della coda di un dispositivo è quel processo che abbia cominciato ad eseguire la funzione dello switch *ATMSwitchUnregisterDev* su un dispositivo in uso da qualche user. Prima di effettuare la disabilitazione del dispositivo infatti, viene invocato sulla wait queue il metodo *wait_event_interruptible*.

Per questa ragione, in seguito al rilascio del riferimento ad un dispositivo, occorre risvegliare i processi nel caso in cui il numero di users sia diventato nullo.

3.3 Il dispositivo ATMDummy

Un dispositivo ATMDummy è una virtualizzazione di un dispositivo ATM implementato allo scopo di avere a disposizione un meccanismo di comprovazione delle principali funzionalità dello switch.

A differenza di un normale dispositivo, non vi è nessun tipo di supporto hardware dal quale estrarre le funzionalità richieste dall'interfaccia di un ATMDDev. Il driver che implementa e gestisce questo dispositivo è una pura astrazione software il cui intento è quello di simulare il comportamento di un dispositivo di rete ATM. Il modulo che implementa il driver per questo dispositivo è l' *atm_dummy.ko*.

Pur non essendoci un'interfaccia fisica di comunicazione con il mondo esterno al sistema, la struttura di questo driver rappresenta un buon punto di partenza per l'implementazione di qualunque altro driver per dispositivi ATM, in quanto fornisce un buon esempio di interfacciamento richiesto ad ogni dispositivo che si voglia registrare nello strato ATM.

La struttura del modulo di questo driver si compone di una prima parte di definizione del tipo specifico di dispositivo che si vuole implementare e di dichiarazione delle strutture necessarie per la gestione interna al modulo dei dispositivi creati.

Un dispositivo ATMDummy viene definito come segue:

```
typedef struct ATMDummy {  
    struct list_head mEntry;  
    ATMDDev mDev;  
    ATMDDevStats mStats;  
    ATMPhyFlags mFlags;  
    struct timer_list mRxTimer;  
} ATMDummy;
```

Come accade nel sistema dei dispositivi del kernel, il modo di costruire dispositivi complessi sulla base di una interfaccia comune è quello di ricorrere alla composizione con la struttura che definisce l'interfacciamento con lo strato superiore. Il campo mDev incorpora in un oggetto di tipo ATMDummy le funzionalità richieste dallo strato ATM, il quale sarà in grado di esercitare il controllo sul dispositivo proprio grazie a questo campo. Rappresenta il punto di incontro fra l'upper layer e il dispositivo fisico utilizzato dal driver; l'inizializzazione dei callback definiti nell'interfaccia permette di adattare il comportamento del dispositivo ATM allo strato fisico sottostante.

Simulazione del flusso di celle

Per testare l'efficienza dello switch, l'ATMDummy adotta un meccanismo di simulazione della ricezione di un flusso di celle. Come velocità di riferimento dell'arrivo di celle si è tenuto in considerazione il bandwidth di un link E1. In un collegamento E1 i dati vengono trasmessi alla velocità di 2048 kbps. Per cercare di produrre un traffico veritiero si calcola l'intervallo di tempo che intercorre fra l'arrivo di due celle consecutive.

La frequenza di ricezione delle celle si ottiene dividendo il bit rate caratteristico di E1, misurato in bps, per la dimensione in bit di una cella ATM, misurata in bit:

$$\frac{\text{Bit rate E1}}{\text{ATM cell size}} = \frac{2048000 \text{ bit/sec}}{53 * 8 \text{ bit}} = 4811 \text{ Hz}$$

Si ottiene un valore di 4811 celle al secondo per la velocità che rappresenta la frequenza con la quale le celle vengono trasmesse e ricevute su un canale fisico di tipo E1.

L'inverso di questa frequenza rappresenta il tempo necessario per la completa trasmissione e ricezione di una cella ATM e per tanto può essere utilizzato come tempo che intercorre fra la ricezione di due celle consecutive nella simulazione di un flusso di dati E1.

$$\frac{1}{4811 \text{ Hz}} = 0.000207 \text{ sec} = 207 \mu\text{sec}$$

La struttura ATMDummy contiene al suo interno un timer che viene utilizzato per simulare la ricezione degli interrupt che, nel caso di reali dispositivi di rete, l'hardware provvede a sollevare per avvisare dell'avvenuta ricezione di dati dal canale fisico.

Questo timer viene configurato per produrre un'interruzione software con una frequenza pari alla frequenza di ricezione delle celle precedentemente calcolata. Il timer misura il tempo contando i tick del clock di sistema. Nel sistema embedded utilizzato, il tick rate è impostato a 1000 Hz. Ciò significa che il kernel è capace di misurare il tempo con una granularità pari a un tick ogni millisecondo.

Ne segue che il timer non può considerare valori di tempo inferiori al millisecondo. Quindi, si imposta il tempo di espirazione del timer pari a 10 volte il tempo di ricezione di una cella, ovvero 2070 μsec . Ogni volta che il timer, espirando, simula l'arrivo di celle, si calcola una quantità aleatoria di celle compresa fra 0 e 10, per simulare l'estrazione di celle corrette da un flusso che può contenere celle non corrette o idle.

Capitolo 4

Progettazione di uno Switch ATM

Uno switch ATM è una infrastruttura in grado di associare ad una specifica data unit entrante una corrispondente porta di uscita, avvalendosi dei parametri VPI e VCI e del identificatore del port fisico di input. È proprio l' entità di dati indirizzata dallo switch, la cella ATM, a trasportare questi parametri permettendo al modulo logico di dirigere il suo instradamento.

Il compito di uno switch è quello di gestire le interfacce dei link capaci di trasferire le data unit e, congiuntamente, vigilare e dirigere la commutazione del traffico dati attraverso la maglia dei circuiti esistenti e previamente definiti fra queste interfacce.

È caratterizzato dai seguenti aspetti fondamentali:

- registrazione e gestione dei dispositivi ATM;
- creazione e gestione delle interconnessioni fra i dispositivi registrati;
- gestione di una tabella di mapping dei circuiti virtuali;
- implementazione di una logica di commutazione delle celle.

Gli switch sono generalmente apparecchiature hardware che contengono delle matrici di bus in cui i punti di intersezione rappresentano gli instradamenti dei flussi di data unit fra i link.

L'implementazione software deve poter competere con l'efficienza che caratterizza queste apparecchiature, la cui capacità di commutazione del traffico risulta essere elevata, poichè veicolata da circuiti elettronici in grado di dirigere contemporaneamente la commutazione di molteplici canali.

Per poter raggiungere questo obiettivo, occorre determinare una buona gestione della parte fondamentale di ogni switch, il nucleo del processo di switching delle celle, cioè il suo modo di operare su una struttura per la memorizzazione delle regole di switching fra i dispositivi.

4.1 Aspetti realizzativi

Il modulo che implementa le funzionalità dello strato ATM che competono a uno switch ATM è l' *atm_switch.ko*.

Una libreria per il trattamento delle celle, la *ATMCell*, viene linkata insieme al modulo *atm_switch.ko*. Questa libreria è stata implementata al fine di fornire l'insieme di utilità per la gestione delle celle ATM a tutti gli elementi che risiedono nello strato ATM.

Infine, l'ATM layer viene complementato con le funzionalità offerte dall'implementazione di una struttura di accelerazione usata dallo switch per accelerare il processo di commutazione delle celle.

L'architettura dello switch è definita da una struttura interna al modulo, la *struct ATMSwitch* e da una insieme di metodi pubblici definiti nel file di header *atm_switch.h*.

```
typedef struct ATMSwitch {
    struct list_head mDevList;
    struct list_head mRuleList;
    uint32_t mNDevs;
    uint32_t mNRules;
    rwlock_t mLockDev;
    struct workqueue_struct *mWorkQueue;
    struct work_struct mWork;
    sk_buff_head mCells;
    ATMSwitchStats mStats;
} ATMSwitch;
```

Una variabile globale di tipo *ATMSwitch* dichiarata all'inizio del codice del modulo viene correttamente inizializzata nel momento in cui il modulo viene inserito nel kernel.

A partire da questo momento le sue funzionalità vengono esportate e rese disponibili al resto dei driver che includano il file di header.

I metodi dichiarati da questo file sono:

- `int ATMSwitchRegisterDev(ATMDev *dev)`
- `int ATMSwitchUnregisterDev(ATMDev *dev)`
- `int ReceiveCell(struct ATMDev *this, sk_buff *cell)`

I dispositivi che saranno utilizzati per ricevere e trasmettere celle ATM dovranno registrarsi nel sistema come dispositivi ATM tramite la chiamata

alla funzione *ATMSwitchRegisterDev*. Una volta che il loro uso non sia più richiesto dovrebbero essere deregistrati per mezzo della funzione *ATMSwitchUnregisterDev*.

L'attività dei dispositivi attivi nello strato ATM di uno switch consiste nella ricezione di flussi di celle ATM, ognuno dei quali corrisponde ad un circuito virtuale entrante. Lo switch recupera le celle ricevute da ogni dispositivo. Affinchè un dispositivo possa notificare la ricezione di una unità di dati, chiama la funzione *ReceiveCell* dello switch delegando ad esso la gestione della cella ricevuta. Se esiste una regola di switching appropriata corrispondente al virtual channel della cella entrante, allora lo switch chiamerà la funzione di callback *SendCell* del dispositivo ATM di output indicato dalla particolare regola di commutazione.

Ogni dispositivo ATM registrato viene inserito in una lista globale il cui punto d'entrata viene identificato con il campo *mDevList*.

I dispositivi che appartengono a questa lista sono correttamente inizializzati e si trovano in uno stato adatto per poter essere attivati. A partire dalla loro registrazione in questa lista, in qualunque momento lo switch può attivarli. L'attivazione, o abilitazione, viene causata dall'inserimento nello switch di una regola di commutazione che coinvolge il dispositivo.

Quando non esisterà più nessun riferimento attivo verso un dispositivo, questo verrà disabilitato, pur continuando a mantenere la sua posizione all'interno della lista.

Le regole di switching sono organizzate a loro volta in due tipi di liste. Una lista globale, il cui punto d'entrata coincide con il campo *mRuleList* dell'*ATMSwitch*, contiene la totalità delle regole gestite dallo switch. Un secondo tipo di lista, introdotto per facilitare la gestione delle regole, le suddivide in gruppi relazionati con il dispositivo di input al quale si riferiscono. Lo switch si occupa del loro inserimento in ciascuna lista e della loro eliminazione.

4.2 Registrazione dei dispositivi

Il modo in cui lo switch ATM gestisce i dispositivi è stato pensato in maniera simile all'organizzazione dei dispositivi di rete adottata dal kernel di Linux. Esiste un parallelismo fra lo strato dedicato al networking del kernel e lo strato ATM che lo switch implementa e, allo stesso modo, fra i net device

e i dispositivi ATM.

Lo switch ATM possiede al suo interno le strutture necessarie per provvedere a gestire ed utilizzare correttamente questi dispositivi.

In particolare, concatena i dispositivi ATM registrati nel sistema in una lista lineare circolare doppiamente linkata. Un dispositivo appartenente a questa lista, può essere attivo oppure no, l'unico vincolo che deve rispettare è la corretta e uniforme implementazione dell'interfaccia comune ATM.

Il file `<linux/list.h>` del kernel contiene le definizioni della strutture dati e delle macro utilizzate per implementare le liste. Si tratta di una speciale implementazione che fornisce uno strumento generale per gestire le liste di ogni tipo di oggetto. La struttura che viene utilizzata è la `struct list_head`; gli unici campi che possiede sono `next` e `prev`. Il solo uso di questa struttura, congiuntamente alle primitive di gestione delle liste, consente di creare agilmente ed efficacemente la lista.

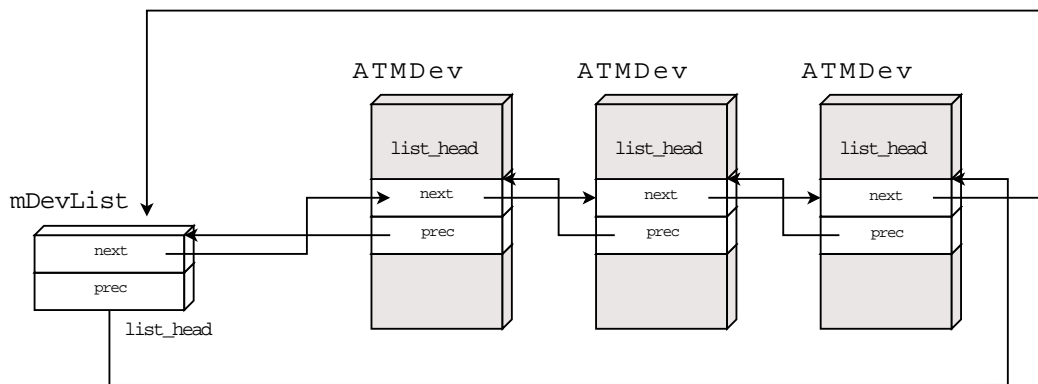


Figura 4.1: Lista di dispositivi ATM registrati

La funzione `ATMSwitchRegisterDev` viene usata per aggiungere un nuovo dispositivo alla lista. Questa funzione viene chiamata dal driver del dispositivo che vuole inserire la sua interfaccia `ATMDev` nella lista globale di dispositivi ATM. Il driver si preoccupa di effettuare previamente la corretta inizializzazione dell'interfaccia e di configurare correttamente l'hardware del dispositivo, controllando che tutte le risorse necessarie siano disponibili. Successivamente presenta il dispositivo al protocollo di strato direttamente superiore, in questo caso lo strato ATM, effettuando la registrazione del nuovo dispositivo.

La funzione `ATMSwitchRegisterDev` accetta come parametro il puntatore

all'interfaccia *ATMDev* che si vuole registrare. Ogni dispositivo ATM possiede un campo *mUsable* che determina se il dispositivo si trova in uno stato idoneo per essere attivato. Il dispositivo diventa "usable" non appena inserito nella lista. Un valore di identificazione unico all'interno dello switch viene assegnato al nuovo dispositivo registrato prima di essere inserito nella lista.

Per rimuovere una struttura *ATMDev* dallo switch viene chiamata la funzione *ATMSwitchUnregisterDev*. Questa funzione marca il dispositivo come non usabile e lo elimina dalla lista. Prima di eliminare la struttura dalla lista controlla lo stato del dispositivo; se il dispositivo è attivo, lo disabilita chiamando il metodo *Disable* dell'interfaccia *ATMDev*.

4.3 Meccanismi di sincronizzazione

I dispositivi alloggiati nella lista dello switch sono una risorsa alla quale diversi processi possono accedere contemporaneamente. Le routine di invio delle celle devono accedere ai dispositivi di output, così come le funzioni di gestione e controllo della lista dei dispositivi. Allo stesso modo, anche le regole di switching rappresentano una risorsa condivisa; le regole possono essere aggiunte o eliminate mentre allo stesso tempo alcuni processi potrebbero tentare di scorrere la lista delle regole al fine di poter trovare la regola di commutazione corretta per una determinata cella entrante.

Risulta per tanto indispensabile fornire dei meccanismi di sincronizzazione che siano in grado di proteggere queste risorse condivise, definendo delle particolari regioni del codice, chiamate sezioni critiche, sulle quali venga esercitato un controllo degli accessi concorrenti.

Il kernel di Linux fornisce differenti tecniche di sincronizzazione volte a impedire l'insorgere di "race conditions". Le tecniche maggiormente utilizzate in questo progetto sono costituite dall'impiego di operazioni atomiche e di spinlock, la cui combinazione riesce a garantire una efficace protezione dei dati condivisi.

Operazioni atomiche

Il modo più semplice di prevenire race conditions consiste nell'assicurare che le operazioni di scrittura e lettura di una variabile condivisa vengano eseguite atomicamente all'interno del processore. Ciò implica che le operazioni di questo tipo vengano eseguite in un'unica istruzione macchina, eliminando il rischio di una interruzione nel mezzo della sua esecuzione e prima del suo completamento, impedendo in questo modo l'accesso contemporaneo da par-

ti di altri processi alla stessa locazione di memoria.

Il kernel fornisce un particolare tipo di dato, l'*atomic_t*, e un insieme di funzioni e macro speciali per il trattamento delle variabili di questo tipo.

Un esempio dell'utilizzo di questo tipo di variabili è dato dall'implementazione del reference counting, descritto nel paragrafo 3.2. In questo contesto, le variabili atomiche consentono una implementazione corretta del meccanismo di conteggio dei riferimenti validi, consentendo di gestire valori coerenti dello stato dei dispositivi anche nei casi in cui molteplici users concorrono alla modifica del loro stato.

Spinlocks

Un'altra tecnica ampiamente utilizzata è il locking. Uno spinlock è un dispositivo di mutua esclusione che si può trovare solamente in uno dei due stati **locked** e **unlocked**.

Un control path del kernel deve necessariamente acquisire il possesso di un lock prima di entrare in una sezione critica; per fare ciò controlla lo stato del lock, e, se questo è disponibile, lo imposta a **locked**, impedendo che ogni altro processo che voglia entrare nella stessa sezione critica acquisisca il controllo di quel lock. Una volta acquisito il lock, nessun altro processo potrà accedere alle risorse protette. Quando un processo desidera acquisire un lock non disponibile, la sua esecuzione entra in un loop di attesa attiva, dove controlla ripetutamente lo stato del lock. Questo loop rappresenta la parte di *spin* di uno spinlock. In questo ciclo di attesa, non viene portato a termine nessun lavoro utile; per questo motivo gli sforzi nella progettazione delle sezioni critiche protette dagli spinlock si concentrano nel cercare di rendere l'attesa della disponibilità dei lock la più breve possibile.

Per la loro natura, gli spinlock vengono comunemente utilizzati in sistemi multiprocessore; tuttavia, il loro uso in sistemi dotati di un unico processore non è da considerarsi inutile per il fatto che uno spinlock acquisito causa la disabilitazione della preemption per il processo che si trova nella regione critica. La preemption continua ad applicarsi invece sui processi in attesa di acquisire il lock e, per tanto, la loro esecuzione può venire ritardata in favore dell'esecuzione di processi più prioritari.

L'acquisizione di un lock è legato al vincolo che il codice che viene eseguito all'interno della sezione critica sia atomico; il processo che detiene il possesso del lock non può sospendersi, in quanto favorirebbe l'insorgere di possibili situazioni di deadlock nel caso in cui un altro processo richieda di entrare nella stessa sezione critica, occupando le risorse di calcolo del processore nel

corso di un infinito loop di attesa attiva.

La struttura dello switch contiene al suo interno due variabili di tipo *rwlock_t*. Il compito delle variabili *mLockRule* e *mLockDev* è quello di proteggere gli elementi appartenenti alla lista dei dispositivi e alla liste delle regole di switching. Per cercare di ridurre la latenza causata dalla presenza di un lock, si è deciso di adottare degli spinlock di lettura e scrittura. Questi lock sono dei tipi particolari di spinlock che permettono l'accesso a una sezione critica a molteplici processi lettori simultaneamente, mentre l'accesso di un processo scrittore avviene in maniera esclusiva.

Questa scelta è motivata dalle numerose letture che si effettuano degli elementi che compongono queste liste, rapportate con le poche operazioni di scrittura e modifica. Infatti, si ricorre la lista delle regole ogni volta che una cella deve essere commutata, mentre si effettua una modifica solo quando una regola viene aggiunta o eliminata. Le liste sono interessate da processi di lettura anche quando si richiede allo switch di stampare gli elementi che le compongono, siano questi dispositivi o regole. La lista dei dispositivi registrati viene modificata solo al fine di inserire o eliminare i dispositivi, operazioni che in un sistema reale, vengono effettuate solamente poche volte per ogni dispositivo.

4.4 Routing table

Una routing table è una tabella di traduzione della coppia di identificatori VPI/VCI contenuti negli header delle celle che provengono da un determinato circuito virtuale.

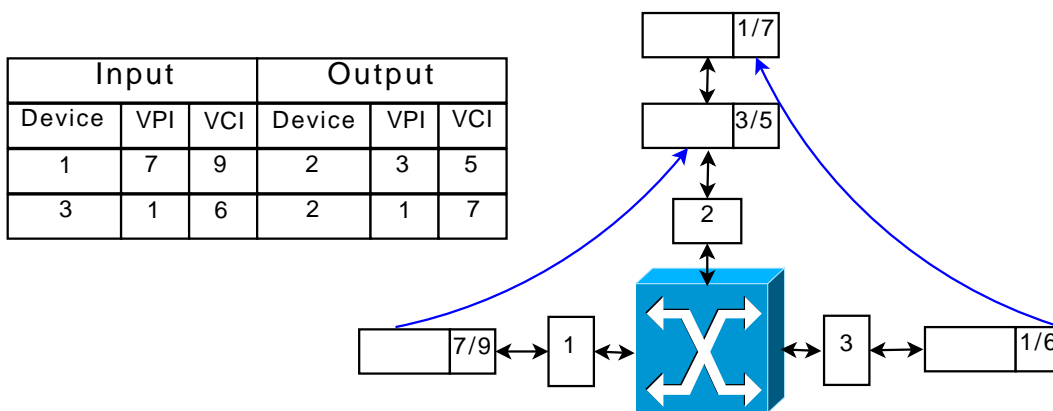


Figura 4.2: Traduzione dell'ATM header cell

Mentre negli switch ATM correnti la routing table è costituita da una matrice di circuiti elettronici che permette la traduzione delle etichette delle celle a livello hardware, in questo progetto il sistema di switching viene implementato completamente via software. Lo switch è un driver che viene inserito nel kernel e al suo interno implementa le strutture dati e gli algoritmi necessari per la gestione della tabella di switching.

Ogni entrata della routing table rappresenta una regola di switching. Gli header delle celle ricevute dallo switch vengono tradotti usando una determinata regola contenuta nella routing table. La regola definisce il dispositivo e la coppia di identificatori del circuito di output sul quale reinviare la cella.

La struttura dati utilizzata per l'organizzazione e la gestione delle regole di switching è una lista circolare doppiamente linkata.

Gli elementi che compongono questa lista sono definiti dalla struttura *ATMSwitchRule* definita come segue:

```
typedef struct ATMSwitchRule {
    ATMRuleDesc mDesc;
    ATMDev *mInDev;
    ATMDev *mOutDev;
    struct list_head mRuleEntry;
    struct list_head mDevEntry;
} ATMSwitchRule;
```

La parte che definisce univocamente una regola è costituita dal suo descrittore. Al suo interno, un descrittore di regola *ATMRuleDesc* contiene i seguenti campi:

Identificatori del circuito di input ID del dispositivo di entrata accompagnato dal coppia VPI/VCI del circuito entrante.

Identificatori del circuito di output ID del dispositivo di output insieme alla etichette VPI/VCI del circuito uscente.

Maschere filtro di input valori booleani associati a VPI e VCI che determinano se gli identificatori di circuito della cella sono determinanti oppure no al fine di produrre un matching con la regola; un valore a true di queste maschere implica che qualunque valore dell'ID corrispondente nell'header della cella concorda con la regola.

Maschere di commutazione valori booleani che determinano se i valori degli ID di virtual path e virtual circuit devono essere tradotti negli identificatori specificati dalla regola

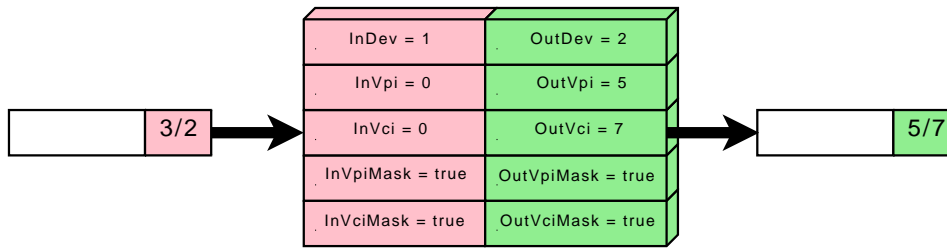


Figura 4.3: ATMRuleDescriptor; la parte in rosa contiene le informazioni utilizzate per il matching con gli identificatori del circuito entrante, mentre quella in verde definisce la commutazione sul circuito di uscita.

Nella Figura 4.3 viene presentato un rule descriptor che rappresenta un tipo particolare di regola di switching di default. Le maschere associate ai valori di identificazione del circuito di input sono poste a true.

Questa regola viene utilizzata per commutare il traffico di celle provenienti da qualsiasi circuito. Possono esistere altre tipologie di regole di default, in cui solo una delle due maschere di input è a true.

Le maschere di commutazione di questo descrittore attuano in modo da cambiare il VPI ed il VCI. Se le maschere di output fossero poste a false, i valori di VPI e VCI di input verrebbero conservati nell'header della cella commutata.

Oltre al descrittore, una regola contiene anche altre informazioni di controllo che assumono un ruolo importante nella gestione delle regole.

I campi *mInDev* e *mOutDev* sono, rispettivamente, i puntatori al dispositivo di input e al dispositivo di output ai quali la regola si riferisce.

Il dispositivo di input rappresenta l'interfaccia ATM dalla quale proviene il flusso di celle che deve essere commutato; quello di output invece rappresenta l'interfaccia verso la quale il traffico deve essere reinviato.

I campi *mRuleEntry* e *mDevEntry* rappresentano i punti di entrata a due diversi tipi di liste. Il primo permette di aggiungere la regola alla lista globale di tutte le regole che lo switch amministra, mentre tramite il secondo la regola può essere aggiunta anche alla lista dello specifico dispositivo di input.

L'introduzione di quest'ultimo tipo di lista è un accorgimento il cui scopo è quello di fornire una più rapida consultazione delle regole di switching, riducendo notevolmente il numero degli elementi della lista da percorrere alla ricerca di un matching per i valori di circuito della cella entrante.

Una regola di default è una regola che possiede una o entrambe le maschere associate al circuito di input a true. Si tratta di regole speciali che per-

mettono di commutare molteplici circuiti utilizzando una stessa regola. Per il particolare significato che hanno, queste regole devono, indipendentemente dal tipo di lista considerato, occupare le posizioni finali della lista. La regola mostrata in Figura 4.3 deve necessariamente occupare l'ultima posizione, per fare in modo che solamente le celle che non trovino una corrispondenza con le regole precedenti vengano commutate usando questa regola di default; se così non fosse, la sua presenza impedirebbe che le regole successive venissero considerate.

4.5 Politica di buffering

Un switch ATM gestisce contemporaneamente un insieme di interfacce attive che provvedono alla ricezione e all'inoltro di traffico ATM. Ogni interfaccia corrisponde ad un dispositivo ATM che, se abilitato, riceve continuamente un flusso di celle. Lo switch deve essere capace di processare contemporaneamente tutti i flussi di celle provenienti dai diversi circuiti del sistema; per risolvere il problema di contesa delle risorse di processamento dello switch, si implementa una politica di buffering il cui scopo è quello di memorizzare tutte le celle in una coda di attesa comune, delegando la gestione di tali celle ad un processo che verrà eseguito quando le risorse dello switch saranno disponibili.

Quando i dispositivi fisici ricevono delle celle ATM dal Physical Layer, avvertono il sistema sollevando una interruzione hardware.

A seguito di un interrupt, il processore interrompe l'esecuzione dell'attività corrente per eseguire la corrispondente routine di gestione dell'interrupt. Nel caso di una interruzione sollevata al fine di avvisare il sistema dell'avvenuta ricezione di un pacchetto di celle ATM, la routine di trattamento dell'interrupt si dovrebbe incaricare di commutare ogni cella sul suo corrispondente port di output. Questa operazione potrebbe richiedere un tempo pari a diverse migliaia di tick dell'orologio di sistema prima che le celle ricevute siano correttamente processate dallo switch.

Il problema nel caso delle interruzioni hardware è che queste vengono eseguite in un interrupt context. Ciò significa che non possiedono in background un processo al quale poter essere ricondotte a seguito di una eventuale sospensione. Perciò, la routine di gestione di una interruzione non può essere interrotta una volta che inizia la sua esecuzione, acquisendo il possesso del processore per tutto il tempo necessario al suo completamento.

Altri processi ugualmente prioritari sono costretti ad attendere fino al termine dell'esecuzione della gestione dell'interrupt.

Per questa ragione, si cerca di fare in modo che la gestione dell'interrupt sia la più rapida possibile e si suddivide la risposta all'evento di ricezione in due parti:

Top Half Coincide con la gestione dell'interrupt; le responsabilità di questa parte vengono ridotte per riuscire a mantenere accettabilmente basso il tempo della sua esecuzione. Le celle contenute nella PDU ricevuta vengono memorizzate nel kernel, e bufferizzate nella coda di attesa contenuta nello switch; la variabile *mCells* di tipo *sk_buff_head* rappresenta la coda di attesa.

Bottom Half Questa parte si occupa di tutte quelle operazioni che possono ammettere un ritardo ragionevole nella loro esecuzione e che per ragioni di vincoli temporali non possono essere portate a termine all'interno del gestore di interruzione. Le celle vengono prelevate dalla coda di attesa, processate, ed infine commutate.

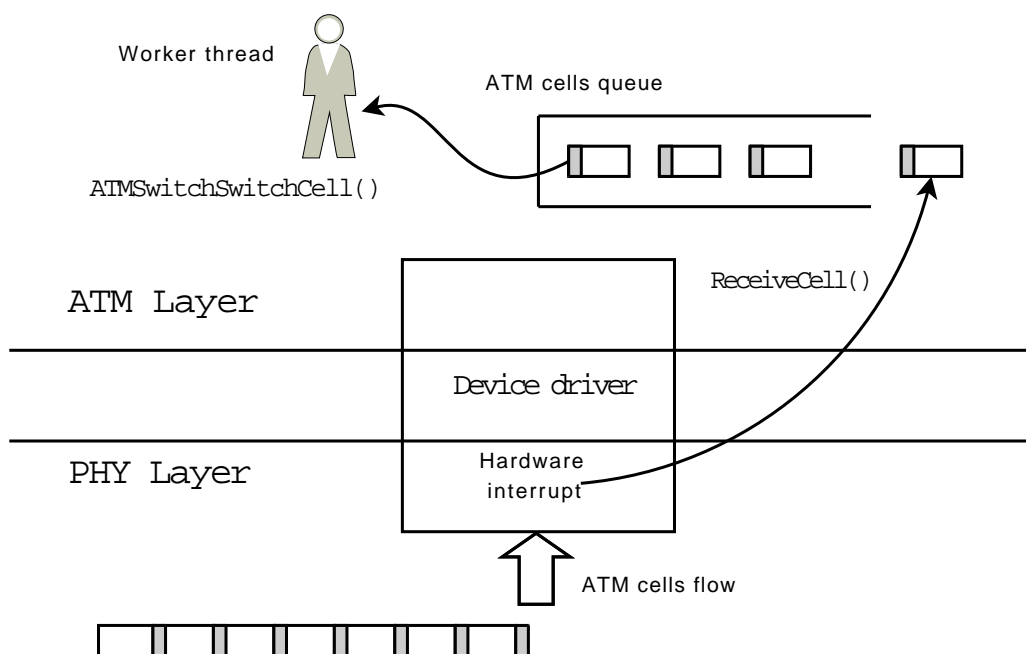


Figura 4.4: Top Half e Bottom Half

La divisione in Top Half e Bottom Half si ottiene impiegando una work queue. Le work queue sono uno strumento offerto dal kernel di Linux per posticipare l'esecuzione di un lavoro.

La funzione `create_single_thread_workqueue` permette di creare una coda di lavoro disponibile solamente per lo switch, la *SwitchWorkQueue*, con

un thread attivo per ogni processore del sistema. Il lavoro viene delegato alla *SwitchWorkQueue* ogni volta che un dispositivo ATM chiama alla funzione di callback *ReceiveCell*.

A seguito di una interruzione viene invocata la funzione *ReceiveCell*; si tratta di un metodo pubblico esportato come simbolo dal modulo che implementa il driver dello Switch ATM. Ogni volta che ad un dispositivo arriva una PDU contenente celle ATM, il device chiama questa funzione dello switch per poter consegnare una alla volta le celle che sono pronte per essere elaborate dallo strato ATM. La cella viene depositata nella coda di attesa solo se si verificano le seguenti condizioni:

- la dimensione della coda non supera la lunghezza massima consentita;
- è possibile ottenere un riferimento valido al dispositivo di input.

Si impone una dimensione massima alla coda per non degradare significativamente le prestazioni dello switch con un carico elevato di lavoro. È importante assicurarsi che il dispositivo si trovi in uno stato consono a poter essere utilizzato; la funzione *ATMGetDev* non andrà a buon fine se il dispositivo è stato disabilitato o si trova in fase di deregistrazione. Inoltre, occorre garantire che il dispositivo continui ad esistere nel momento in cui la cella verrà estratta dalla coda per poter estrarre le informazioni necessarie al processo di commutazione.

Se una o entrambe delle condizioni sopra citate non sono verificate, la cella verrà rifiutata e si incrementerà il contatore *RxCellDrop* contenuto nelle statistiche dello switch.

L'ultima operazione importante da eseguire all'interno di questo interrupt handler è lo schedulamento della parte Bottom Half; la chiamata *queue_work* permette di mettere il worker thread associato alla coda di lavoro in attesa di essere eseguito.

Una volta eseguito l'interrupt, il gestore della work queue verrà eseguito dal worker thread associato alla *SwitchWorkQueue* quando lo scheduler lo riterrà più conveniente. In fase di inizializzazione dello switch si inizializza il gestore della coda di lavoro con la funzione *ATMSwitchSwitchCell*. All'interno di questa funzione si svolge l'intero processo di commutazione della cella. Il tempo impiegato è variabile a seconda del circuito virtuale dal quale la cella proviene e dalla posizione della regola associata nella lista.

La durata dell'esecuzione di questo metodo tuttavia non risulta più essere un problema in quanto la funzione viene eseguita da un thread del kernel, ed

è quindi consentita la sua sospensione a seguito, per esempio, dell'arrivo di un nuovo interrupt.

4.6 Commutazione delle celle

La totalità delle celle ricevute dalle interfacce attive dello switch vengono depositate nella coda di attesa contenuta all'interno dello switch.

Le celle vengono trasportate nella parte riservata ai dati di un *sk_buff* e la coda altro non è che l'implementazione che il kernel fornisce delle code di *sk_buff*. Non sono necessari meccanismi di protezione contro l'insorgere di race condition, in quanto all'interno di una struttura *sk_buff_head* vi è già uno spinlock che il kernel provvede a gestire opportunamente.

Nel momento in cui lo scheduler risveglia il worker thread dal suo stato di sospensione viene iniziato il processo di commutazione. Lo switching di una cella si articola nei seguenti passi:

1. estrazione di una cella dalla coda;
2. risoluzione del puntatore al dispositivo di input;
3. estrazione di VPI e VCI dall'header della cella;
4. ricerca di una regola di commutazione associata al circuito entrante;
5. identificazione del dispositivo di output;
6. ottenimento di un riferimento valido per il dispositivo di output;
7. rilascio del riferimento al dispositivo di input;
8. label switching;
9. invio della cella al dispositivo di output;
10. rilascio del riferimento al dispositivo di output;
11. rischedulamento del worker thread.

Il puntatore al dispositivo di input è disponibile all'interno del buffer che trasporta la cella. Una funzione definita nella libreria di gestione delle celle ATM permette di estrarlo dalla parte privata dell'*sk_buff*.

L'utilizzo di questo puntatore è sicuro in quanto la cella viene depositata all'interno della coda solo dopo aver ottenuto un riferimento valido ad esso ed incrementato il contatore degli users di quel dispositivo.

Il puntatore così ottenuto svolge due funzioni: permette di stabilire il tipo di interfaccia (UNI/NNI) per una corretta interpretazione del campo VPI dell'header della cella e consente di accedere alla lista delle regole associate al dispositivo di input.

La lista delle regole associata all'*InDev* viene percorsa interamente fino ad incontrare una opportuna regola di switching che corrisponda con i valori di identificazione del circuito entrante. In caso non venga prodotto nessun matching durante la scansione della lista la funzione termina.

Una volta incontrata la regola, al suo interno si trovano tutte le informazioni necessarie per procedere allo switching della cella. Il descrittore della regola indica il tipo di commutazione da applicare nella traduzione dell'etichetta, mentre il puntatore *OutDev* identifica il dispositivo da utilizzare per l'invio della cella.

A questo punto il dispositivo di input non è più necessario ed il suo riferimento viene rilasciato invocando la funzione *ATMSwitchPutDev*. Se il dispositivo di output è disponibile, la cella viene commutata e inviata all'*OutDev* chiamando il callback *SendCell* sul dispositivo.

Affinchè tutte le celle presenti nella coda siano processate, occorre rischiudere il thread che si occuperà della commutazione delle celle rimanenti.

4.7 Ottimizzazione del processo di switching

La velocità con la quale viene portato a termine il processo di commutazione è molto importante in quante da essa dipende direttamente il valore della latenza sui circuiti virtuali che compongono la rete ATM.

In sostanza, l'operazione fondamentale che deve essere portata a termine durante il processo di switching è la ricerca della regola di commutazione adeguata; per questo motivo lo studio e l'analisi di una buona struttura dati e di algoritmi per l'esecuzione di questo processo si baseranno prevalentemente sulla performance delle implementazioni dell'operazione di ricerca.

L'implementazione di base della routing table dello switch è costituita da una double linked list. La ricerca all'interno di una lista lineare composta da n regole di switching ha un costo di $O(n)$. Se le regole sono molte, il costo diventa elevato.

La situazione desiderabile sarebbe quella in cui il costo di una ricerca fosse pari a $O(1)$. Le strutture dati che hanno un comportamento di questo tipo sono gli array associativi. Gli array associativi mappano un insieme di chiavi

in una collezione di valori. L'operazione di ricerca viene chiamata *indexing* e pretende essere eseguita in un tempo asintoticamente costante.

Un semplice vettore in cui il valore dell'indice corrisponda alla chiave in ingresso utilizzata per la ricerca, ed il valore registrato nell'elemento corrispondente sia l'elemento ricercato, fornisce indubbiamente l'implementazione più efficiente dal punto di vista di costo computazionale.

Lo svantaggio che l'uso di un vettore comporta è costituito dal fatto che il dominio degli indici deve coincidere con l'universo di tutte le possibili chiavi. Nel caso della commutazione di celle ATM, la chiave è costituita dall'insieme dei valori di VPI e VCI. Il dominio di questi valori è estremamente ampio, dato che si tratta di valori rappresentabili congiuntamente con 24 bit nel caso di interfacce UNI e con 28 nel caso di interfacce NNI.

L'ammontare di memoria utilizzata per le strutture dati è un parametro che assume una notevole importanza nel caso dei sistemi embedded, dove si dispone di risorse di calcolo e memoria molto limitate.

Una possibile soluzione che mantiene costante il costo dell'operazione di ricerca evitando il consumo eccessivo di memoria è costituita dalle tabelle hash. Una tabella hash usa tipicamente un array di dimensione proporzionale al numero di chiavi effettivamente memorizzate. Invece di usare la chiave come indice per indirizzare direttamente l'array, l'indice viene calcolato applicando una funzione hash alla chiave di ricerca.

Sebbene la ricerca di un elemento in una tabella hash possa richiedere lo stesso tempo di ricerca in una lista concatenata, un tempo pari a $O(n)$ nel caso peggiore, in pratica le prestazioni del metodo hash sono estremamente buone. Sotto ragionevoli ipotesi di uniformità e di semplicità della funzione hash, il tempo medio di ricerca di un elemento in una tabella hash è $O(1)$.

La definizione della tabella hash utilizzata e le dichiarazioni delle funzioni per la sua gestione si trovano nell'header *atm_acc.h*. Le tabelle hash utilizzate in questo progetto vengono così definite:

```
typedef struct ATMAcc {
    unsigned int mLength;
    ATMAccSlot *mTable;
    void *mDef;
    void *mBigDef;
    unsigned int mNumDefRules;
    unsigned int mNumRules;
} ATMAcc;
```

La tabella hash vera e propria è un array di elementi di tipo *ATMAccSlot* il cui riferimento viene memorizzato nel puntatore *mTable*. La dimensione della tabella è variabile; a seguito di ogni inserzione di chiave si controlla che il valore del load factor non superi il valore 0.7. Mantenere un load factor ragionevolmente basso evita il degrado delle prestazioni della tabella hash.

Le regole di switching si possono suddividere in due categorie: le regole fisse che definiscono valori ben precisi di VPI e VCI del circuito entrante e le regole di default, che possiedono una o entrambe le maschere associate agli identificatori del circuito di input a true, e che si applicano a molteplici circuiti entranti.

All'interno della tabella di hash vengono memorizzate le regole fisse. La chiave è costituita dalla combinazione di VPI e VCI, come definito dalla macro:

```
#define ATM_ACC_KEY(vpi, vci) ( vpi*(1<16) + vci )
```

Alla chiave così definita viene applicata la funzione hash implementata nella funzione *ATMAccFunc*. Si tratta di una funzione di hash specificamente pensata per la natura del dominio di valori di chiavi considerato. Utilizza la tecnica del folding per ovviare a possibili concentrazioni dei valori che identificano il circuito in determinati intervalli.

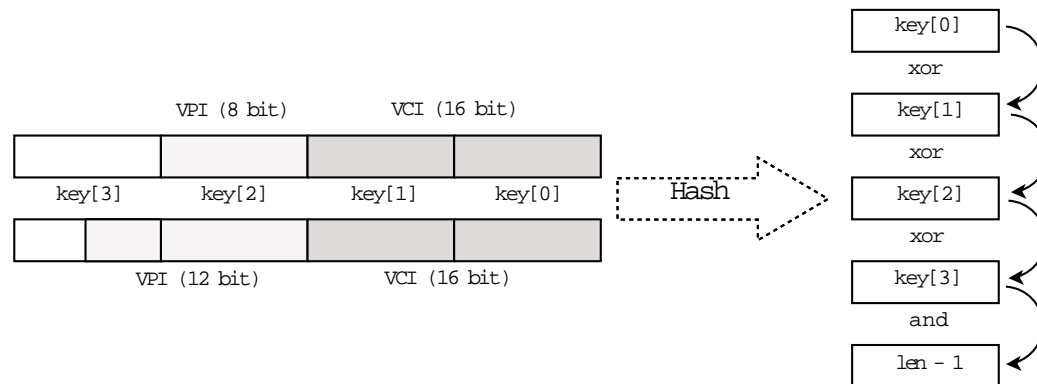


Figura 4.5: Funzione hash

I byte che compongono la chiave vengono ripiegati su se stessi applicando bit a bit l'operazione xor. Come ultimo passo, al risultato degli xor consecutivi, viene applicato un and bit a bit con il valore della lunghezza della tabella hash - 1. L'effetto è lo stesso che si otterrebbe tramite una operazione modulo, ma risulta computazionalmente più veloce: il valore hash viene

mappato sui valori degli indici della tabella hash.

Viene utilizzato il metodo dell'open addressing. Tutte le chiavi sono memorizzate all'interno della tabella. Ogni slot contiene il valore della chiave memorizzata ed il puntatore alla regola che verrà usata per commutare le celle provenienti dal circuito definito dal VPI e VCI contenuti nella chiave. Le collisioni vengono risolte con la tecnica del linear probing.

Le regole di default ricevono un trattamento particolare.

Non possono essere inserite nella tabella hash come fossero normali regole. La ragione risiede nel fatto che queste regole si applicano a molteplici circuiti. Risulta evidentemente complicato fare in modo che una stessa funzione hash riesca a indicizzare differenti identificatori di circuito nello stesso slot della regola di default. Per questo motivo le regole di default vengono gestite tramite un puntatore che punta alla prima regola delle regole di default della lista del dispositivo.

In caso la cella entrante non trovi una corrispondenza con le regole fisse memorizzate nella tavola hash, si provvederà ad accedere direttamente alla prima occorrenza delle regole di default nella lista, ed a proseguire la ricerca sui suoi componenti fino ad arrivare all'ultimo.

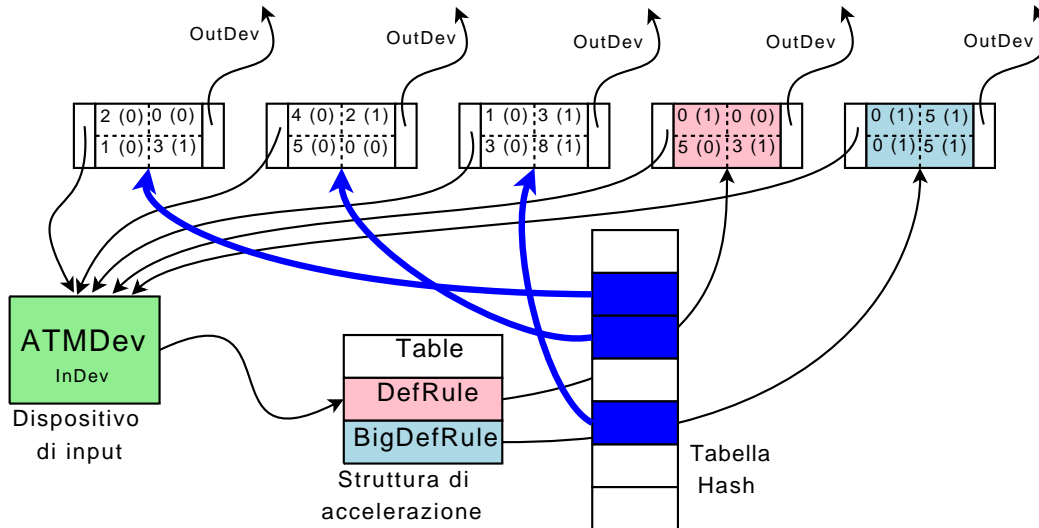


Figura 4.6: Tabella hash

La commutazione di una cella avviene estrendo i campi VPI e VCI dell'header trasformandoli in una chiave di ricerca per la tabella hash. La chiave

così determinata viene indicizzata per mezzo della funzione hash. Le collisioni vengono risolte con linear probing. Il tempo medio della ricerca è $O(1)$. Nel caso di un insuccesso, la ricerca prosegue nella parte di lista dedicata alle regole di default. Nei sistemi reali non ci si aspetta un elevato numero di regole di default, pertanto il tempo di scansione delle regole di default si può considerare costante.

4.8 La libreria *ATMCell*

La libreria ATM ha lo scopo di fornire le funzioni di trattamento dell'unità fondamentale di trasferimento dei dati all'interno dell'ATM layer, la cella ATM.

La libreria *ATMCell* viene linkata nello stesso modulo che implementa lo switch, al fine di rendere accessibili a tutti e soli dispositivi ATM registrati nel sistema le utilità in essa implementate.

Ogni cella ATM è trasportata dentro un *sk_buff*.

Gli *sk_buff* sono dei buffer che vengono usati all'interno del kernel per organizzare e gestire i pacchetti di rete. Sono stati pensati e ideati per adattare le loro funzionalità al loro uso fra i vari strati che compongono gli stack dei protocolli di rete. Sono costituiti da un header di dimensione fissa e da una parte di dimensione variabile atta a contenere il contenuto di un pacchetto di dati. L'header contiene tutte le informazioni necessarie a manipolare questi buffer; si trovano puntatori alle differenti zone di memoria di un *sk_buff*, le dimensioni di queste parti, un contare dei riferimenti e infine le strutture dati necessarie per poter raccogliere e gestire i pacchetti in liste di *sk_buff*.

La *ATMCell* astrae dalle funzioni specifiche del kernel un'interfaccia appositamente studiata per il caso in cui il dato trasportato sia una cella ATM; vengono definite le funzioni per la costruzione e la distruzione delle celle, le gestione dei campi dell'header, e infine per l'accesso diretto al payload della cella.

Le operazioni implementate con l'intento di apportare strumenti di ausilio per il compimento delle funzionalità dello strato ATM sono:

- Costruzione e distruzione della cella ATM.
- Setting e getting dei campi che compongono l'header della cella ATM.
- Verifica di cella idle.
- Calcolo e verifica del campo HEC.

In ogni *sk_buff* c'è uno spazio di 48 byte riservato per il trasporto di dati privati di ogni strato. In questo caso, il campo *cb* (control buffer) è stato utilizzato per memorizzare il puntatore al dispositivo di input da cui la cella è stata ricevuta. Tramite questo puntatore lo switch riesce a ricavare rapidamente le informazioni sul dispositivo di input che gli sono necessarie per la commutazione della cella. Inoltre, per lo switch è indispensabile accedere all'interfaccia *ATMDev* del dispositivo che ha ricevuto la cella per sapere a che tipo di interfaccia appartiene, UNI o NNI, e dedurre di conseguenza come interpretare i bit che compongono l'ATM cell header.

Un altro campo usato, e di particolare importanza in questo progetto, è il campo *stamp*. Rappresenta il timestamp della cella, ossia il momento in cui la cella viene ricevuta. Questa informazione in genere risulta essere interessante per gli strati superiori, mentre, in questo caso, viene utilizzata per effettuare uno studio sul tempo medio di permanenza della cella nella coda di attesa. Il campo *stamp* è di tipo *ktime_t*, e viene settato al momento della ricezione con il tempo di wall time. Questo valore verrà usato come riferimento per determinare il tempo di attesa della cella prima di essere commutata dallo switch.

4.9 L'interfaccia laterale di controllo

Il controllo sul driver che implementa lo switch, viene ottenuto mediante l'implementazione della system call *ioctl*. Questa system call rappresenta un'interfaccia verso l'utente per il controllo dei dispositivi; è uno dei meccanismi per la comunicazione fra kernel space e user space.

Una applicazione d'utente può essere in grado di comunicare con un driver del kernel attraverso la chiamata alla funzione *ioctl()*; questa funzione accetta differenti parametri, uno dei quali rappresenta il comando che si vuole passare al driver mediante la omonima system call.

Il corpo della system call nel driver è costituito essenzialmente da un'istruzione *switch* che seleziona il comportamento corretto del driver in funzione del comando che gli viene passato dall'applicazione. Un header incluso sia nel codice dello switch che nell'applicazione d'utente, elenca una serie di macro ognuna delle quali definisce le operazioni che si possono effettuare sullo switch, oltre a definire le strutture dati che serviranno a trasportare l'informazione che si vuole passare fra kernel space e user space.

Molte delle operazioni definite hanno avuto uno scopo puramente di controllo sul comportamento del driver durante l'implementazione e sono state di ausilio al debugging del codice, altre invece, rivestono un ruolo importante

nella configurazione e gestione dello switch.

Il programma consente di monitorare lo stato del sistema e dei suoi componenti; in ogni momento è possibile richiedere al driver di elencare i dispositivi ATM registrati così come di stilare una lista delle regole di switching totali o suddivise secondo il dispositivo di input a cui si riferiscono.

Questo tipo di operazione avviene consultando le liste contenute nello switch senza alcun meccanismo di protezione contro la possibilità di accessi concorrenti.

Non ci si avvale pertanto degli spinlock di protezione dei dati al fine di non interferire con altri processi del driver prioritari che necessitano accedere alle strutture condivise dello switch. Per questo motivo si adotta una semplice tecnica che permette di ottenere mediante chiamate consecutive la lista degli oggetti in lista: ogni volta si invia un riferimento all'oggetto corrente del quale si è appena effettuata la stampa e una apposita funzione dello switch si incarica di percorrere la lista appropriata ricercando un eventuale oggetto successivo risiedente nella lista. Se la ricerca va a buon fine si invia la descrizione dell'oggetto alla applicazione, in caso contrario la si informa che la scansione della lista è terminata.

Lo switch colleziona una serie di statistiche di invio e ricezione delle celle. Questa informazione viene memorizzata in una variabile di tipo *ATMSwitchStats* contenuta all'interno dello switch. Questo campo viene aggiornato continuamente ad ogni invio e ricezione di dati e può essere facilmente estratto e fatto pervenire all'utente al quale viene anche data la possibilità di effettuare un reset di queste statistiche.

Oltre a poter accedere alle statistiche proprie dello switch è anche possibile effettuare la lettura delle statistiche relative ai dispositivi registrati. Ogni dispositivo pubblica la sua interfaccia *ATMDev* nello switch, che pertanto è in grado di accedere ad ogni campo contenuto in essa. Le statistiche di un dispositivo risiedono in un campo di tipo *ATMDevStats* e vengono rese disponibili all'utente sempre tramite la chiamata di sistema *ioctl*.

Il compito dello switch è quello di creare, a partire da una infrastruttura fisica di dispositivi ATM, una rete di tipo logico. Questa rete è costituita da una serie di canali virtuali che generano un complesso di circuiti trasmissivi arbitrariamente magliati.

Al momento, l'*ATMSwitch* non implementa il protocollo di signalling per la gestione dinamica dei virtual channel, pertanto i canali virtuali sono permanenti e devono essere configurati manualmente dall'utente.

Il modo di definire un circuito virtuale all'interno dello switch consiste

nell'introdurre una regola di switching che mappa i parametri caratterizzanti il circuito locale entrante nei valori di VPI e VCI che identificano un circuito locale di uscita. I parametri che devono essere indicati alla applicazione all'atto di inserimento di una regola sono l'ID del dispositivo di input, ID del dispositivo di output, VPI e VCI di input e output; inoltre, vengono richiesti alcuni valori aggiuntivi che definiscono le maschere associate alla regole, al fine di poter implementare i meccanismi di commutazione di cammino e di circuito.

Capitolo 5

Prove realizzate e risultati

5.1 Ambiente sperimentale

Il progetto è stato implementato e parallelamente testato su un sistema embedded dedicato al networking utilizzato da Albentia Systems nella produzione di alcuni dei suoi prodotti. I sistemi embedded sono computer specializzati, in questo caso caratterizzate da specifiche funzionalità di supporto alle tecnologia di rete. Tipicamente sono dotati di risorse hardware minime che sono indispensabili per assolvere alle funzioni per le quali sono progettati. Per questa ragione sono caratterizzati da frequenze di lavoro abbastanza limitate, da una bassa disponibilità di memoria e dall'assenza di dispositivi di I/O. La Figura 5.1 mostra l'ambiente di lavoro; nella foto sono presenti il developement host utilizzato per la produzione e la compilazione dei sorgenti ed il sistema target sul quale venivano installati i moduli compilati dei driver che costituiscono lo strato ATM.

Il processore integrato nel sistema è un IXP425¹. Si tratta di un processore Intel indicato per i sistemi che richiedono connettività di rete ed un considerevole supporto alle applicazioni di rete. La frequenza di clock del processore è pari a 533 MHz mentre la sua architettura è di tipo ARM (Advanced RISC² Machine). La diversa architettura dei processori delle macchine host e target è stata la ragione per la quale si è dovuto utilizzare un cross compiler per la compilazione dei sorgenti sulla macchina host al fine di poter installare i moduli sul sistema target.

¹www.intel.com/design/networks/products/npfamily/ixp425.htm

²Reduced Instruction Set Computer, indica una filosofia di progettazione di architetture per microprocessori formata da un set di istruzioni semplici che possono essere eseguite in tempi simili



Figura 5.1: Sistema di prova utilizzato per lo sviluppo di questo progetto.

Sulla scheda risiede una memoria flash di 64 MB, mentre si dispone di una RAM di 32 MB. Nella flash vengono memorizzati permanentemente il ramdisk e l'immagine binaria del kernel specificamente compilata per il sistema che verranno caricati in RAM dal redboot, il boot loader utilizzato. La versione del kernel Linux attualmente installata nel sistema è la 2.6.22.1-rt7, con patch di Ingo Molnar per avvicinare la latenza del sistema a quella dei sistemi real time.

Le ragioni che sono alla base dell'uso di Linux sono svariate; l'uso di software open source non è una scelta conveniente solamente per la gratuità del codice, ma è anche una garanzia della presenza di un grande supporto informativo proveniente dalla comunità degli sviluppatori. Inoltre, la possibilità di analizzare il codice del kernel e la natura modulare tipica dei sistemi operativi che seguono una filosofia UNIX favoriscono una buona e relativamente facile implementazione dei driver.

5.2 Calcolo della performance

Uno dei parametri più significativi la cui stima riesce a dare una idea della capacità di commutazione dello switch è dato dal tempo di permanenza della cella all'interno dello switch. Questo tempo è dato dalla somma del tempo di permanenza della cella nella coda di attesa più il tempo necessario al

thread dello switch per ultimare il processo di switching della cella. Inoltre occorre tener presente che il sistema è di tipo real time, pertanto i tempi misurati saranno anche influenzati dalle modalità di context switch dei task e dai ritardi di esecuzione dovuti alla preemption sui processi.

La mancanza di tempo utile per effettuare una completa serie di test metodologici ha impedito lo studio approfondito dei vari fattori che influenzano il tempo totale di switching di una cella. I test attualmente implementati riguardano il calcolo del tempo di permanenza della cella nella coda di attesa. Questo tempo, pur non essendo il tempo totale del processo di switching, offre comunque una buona idea di quelle che possono essere le reali performance del sistema reale.

L'obiettivo delle prove eseguite è stato quello di effettuare una stima del tempo medio di permanenza delle celle in coda, osservando come questo valore cambia secondo il carico di lavoro dello switch e, ancora più importante, a seconda della particolare gestione dello switch con rispetto all'individuazione delle regole di switching da associare alle celle ricevute. Con carico di lavoro si intende il numero di circuiti dai quali i dispositivi ATM ricevono le celle.

Il modo con cui si individuano le regole di switching appropriate alle celle da commutare dipende da come viene gestita la routing table. La routing table viene implementata usando tre diverse strutture dati con i relativi algoritmi di switching:

1. lista generale di tutte le regole memorizzate nel sistema;
2. liste contenenti solo le regole che si riferiscono al dispositivo di input al quale la lista è associata;
3. tavola Hash per ogni dispositivo registrato nel sistema contenente le regole del dispositivo.

Per ognuna di queste implementazioni si sono effettuati dei cicli di test volti a misurare il tempo medio di permanenza delle celle in coda.

5.3 Impostazione dei test

La capacità di commutazione dello switch ATM è stata testata in laboratorio. L'architettura complessa del sistema finale in cui si inserirà il layer di switching progettato non si trova ancora in uno stato idoneo all'utilizzo finale. I principali ritardi sono dovuti a problemi riscontrati nello sviluppo del controllore della scheda E1.

Non potendo perciò disporre di dispositivi fisici E1, non si è potuto terminare di sviluppare il driver ATMoE1. Comunque, grazie al supporto fornito dai dispositivi ATMDummy, non solo si è riusciti ad effettuare un debugging contemporaneo alla sviluppo del software, ma si è anche avuto uno strumento per poter testare l'efficienza del sistema.

La simulazione di uno scenario reale è avvenuta creando dei dispositivi ATMDummy. La rete logica di circuiti virtuali è stata costruita sulle interfacce ATM registrate nello switch ed offerte da questi dispositivi. Il parametro in base al quale si relaziona la performance del sistema è stato il numero di circuiti entranti, poichè da questo dipende direttamente la capacità di dirigere il traffico delle celle ATM.

Sono stati costruiti degli schemi aleatori di circuiti virtuali e di regole di commutazione corrispondenti. Si è tenuto in considerazione le caratteristiche dell'infrastruttura di rete all'interno della quale il layer ATM verrà installato. Il numero di interfacce ATM registrate nel sistema non può essere elevato, a causa della limitatezza di risorse di cui dispone il sistema embedded. Questa limitazione non costituisce un problema rilevante, dato che le stazioni di rete finali su cui si installerà lo switch non saranno tenute a gestire molte di queste interfacce.

Per poter valutare il tempo medio di permanenza della cella nel sistema sono stati creati un insieme di problemi giocattolo (toy test), progettati al fine di evidenziare e distinguere chiaramente il diverso tipo di risposta del sistema di switching secondo le diverse implementazioni utilizzate della routing table.

In Figura 5.1 viene illustrata la metodologia di testing utilizzata per misurare la performance dell' ATM Soft Switch.

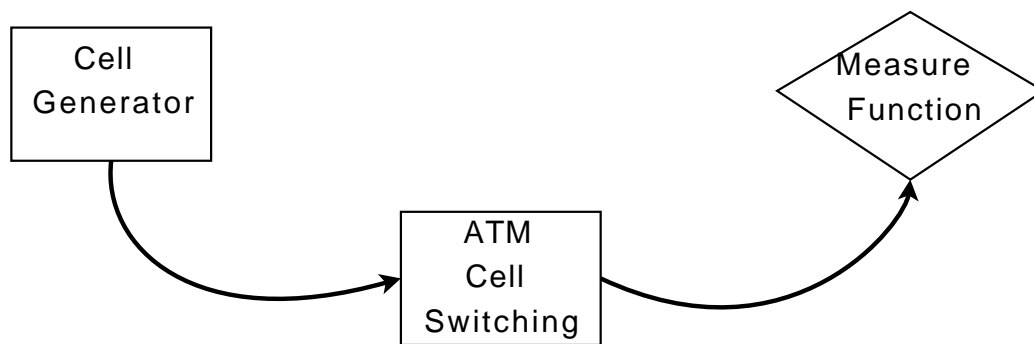


Figura 5.2: Framework utilizzato negli esperimenti.

Generazione delle celle

I dispositivi ATMDummy simulano il comportamento di un dispositivo fisico prendendo come riferimento il flusso reale di celle che si ottiene da un link fisico di tipo E1. Come calcolato nel Paragrafo 3.3, la velocità di ricezione delle celle in questo caso è pari a 4811 celle al secondo. Un dispositivo ATMDummy riceve un flusso virtuale di celle ogni volta che il timer scatta. Il timer è impostato per scattare ad ogni intervallo di tempo pari a 10 volte il tempo di ricezione di una cella da un canale fisico E1 ($207 * 10 = 2070 \mu sec$). In questo momento viene lanciata un'interruzione software che provvede a scegliere, utilizzando una distribuzione di probabilità uniforme, il numero corretto di celle ricevute, ossia un numero compreso fra 1 e 10 (numero di celle ricevuto fra due consecutivi tick del timer), per simulare la presenza di celle idle e celle non corrette sul canale fisico. Queste impostazioni determinano il flusso di dati ricevuti.

Il passo seguente prevede di determinare il circuito di provenienza della cella. Nel file che implementa il driver di ATMDummy sono state definite staticamente, una per ogni dispositivo registrato nel sistema sperimentale, delle tabelle contenenti le informazioni da inserire nell'header della cella ATM generata, inclusi gli identificatori VPI e VCI.

Ogni entrata di queste tabelle rappresenta pertanto una determinata tipologia di cella proveniente da un determinato circuito virtuale. L'indice utilizzato per costruire la cella corrente viene generato con una distribuzione di probabilità uniforme nell'insieme di indici di questa tabella. Il 20% dei circuiti così definiti non avranno un riscontro nelle regole inserite nello switch durante l'esecuzione dei test, per includere nel traffico simulato un grado il più possibile reale di assenza di regole di commutazione appropriate all'interno del sistema.

Questo secondo passaggio determina la tipologia di cella ricevuta, ovvero il tipo di servizio veicolato da ATM (es. VoIP, dati, streaming video). Una volta terminato di inizializzare la cella secondo la tipologia determinata, il dispositivo simula la ricezione chiamando il callback dello switch *ReceiveCell*.

Switching della cella

La cella viene inserita nella coda di attesa dello switch all'interno della routine di gestione dell'interrupt software. Quando il worker thread dello switch termina di processare tutte le celle che sono state introdotte precedentemente, estrae la cella e ricerca nella routing table una regola appropriata. Il ritardo introdotto nell'esecuzione completa di questo processo dipende dalla gestione

dello scheduling dei processi del kernel e dal tempo di ricerca della regola di commutazione.

Misura del tempo di permanenza della cella in coda

Al fine di calcolare il tempo di permanenza nella coda di attesa, ad ogni cella ricevuta viene assegnata una marcatura temporale relativa al momento in cui viene depositata nella coda di attesa. All'interno dei buffer utilizzati per trasportare le celle, gli *sk_buff*, c'è un campo chiamato *tstamp* il cui scopo è proprio quello di memorizzare il tempo attuale del succedersi di un evento di particolare importanza. Seguendo l'esempio offerto dal kernel di Linux, questo campo è stato utilizzato per mantenere traccia, tramite la funzione del kernel *ktime_get_real*, del momento in cui la cella viene ricevuta dallo switch.

Ogni volta che lo switch preleva una cella dalla coda per commutarla, estrae il valore di timestamp precedentemente memorizzato, calcola la differenza fra il tempo corrente del sistema ed il tempo del timestamp ottenendo il tempo di permanenza in coda per quella cella. Gli intervalli di tempo così ottenuti vengono sommati e raccolti insieme al numero di campioni in una struttura interna dello switch. La struttura che memorizza questi valori è:

```
typedef struct ATMTime {
    spinlock_t mLock;
    ktime_t mTime;
    uint32_t mNumSamples;
} ATMTime;
```

Il tipo di dato utilizzato per rappresentare il tempo è *ktime_t*, definito nel file *include/linux/ktime.h*. Si tratta di un tipo di dato che fornisce una duplice risoluzione del formato del tempo in secondi ed in microsecondi. Le informazioni contenute all'interno della struttura *ATMTime* sono protette da uno spinlock il cui fine è quello di garantire che i dati vengano raccolti correttamente.

È stata implementata una applicazione che provvede ad effettuare rilievi di questi dati ciclicamente, per evitare che si possa verificare un overflow delle variabili utilizzate. A determinati intervalli di tempo impostati dall'utente, l'applicazione effettua un reset del tempo e del numero di campioni memorizzati nello switch e rende disponibili questi dati allo user space. Per poter trasferire la misura del tempo raccolta all'interno dello switch è necessaria una trasformazione ad un tipo di dato maneggiabile nello user space.

A questo scopo è stato utilizzato il tipo *struct timeval* definito in *include/linux/time.h*. Dopo 10 intervalli di tempo trascorsi, viene calcolato il tempo medio di permanenza delle celle che sono state commutate durante la durata dell'applicazione.

Per ogni implementazione della routing table, l'applicazione è stata utilizzata per testare il comportamento del sistema all'aumentare del numero di circuiti entranti. L'applicazione è stata lanciata 10 volte per ogni caso considerato ed è stata calcolata una media dei valori medi dei tempi di attesa ricavati.

La Figura 5.2 mostra il grafico dei risultati ottenuti per i test eseguiti.

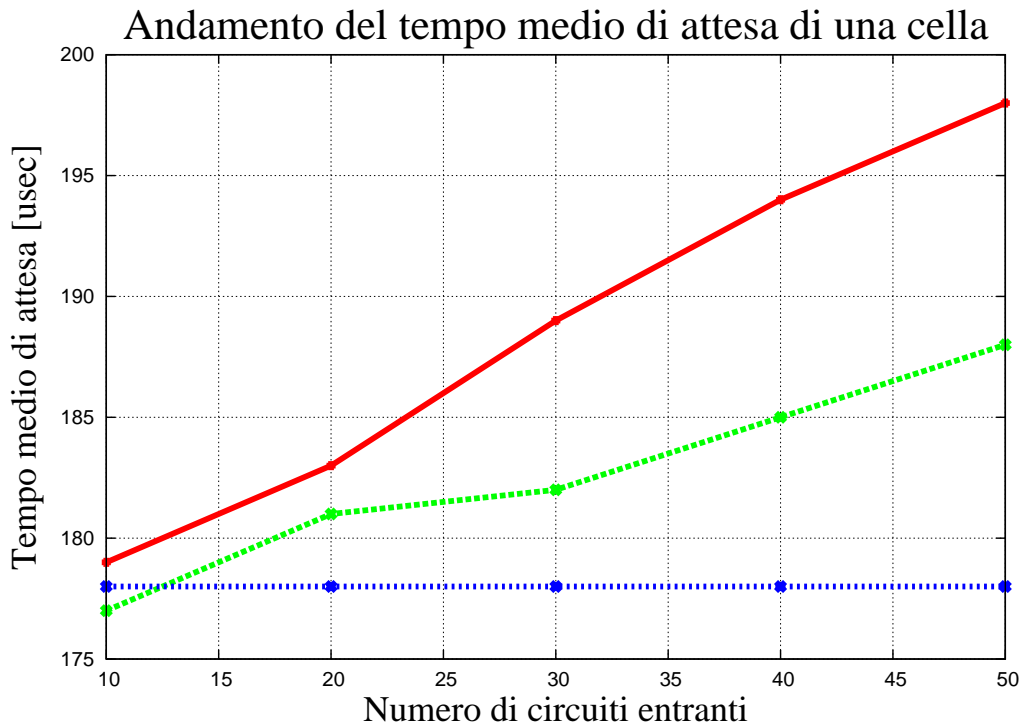


Figura 5.3: Grafico dell'andamento del tempo medio di attesa in coda delle celle in funzione del numero di circuiti entranti da cui lo switch riceve le celle. La curva rossa rappresenta il tempo misurato utilizzando la lista generale di regole, quella verde si riferisce alle liste associate ai dispositivi di input, mentre quella azzurra riguarda l'implementazione con le tabelle hash.

I test hanno confermato le ipotesi iniziali. L'implementazione della routing table mediante tabelle di hash non solo risulta essere più veloce, ma,

come sperato, raggiunge l'obiettivo con un tempo asintotico praticamente costante.

Per quanto riguarda le liste di regole, la suddivisione delle regole per dispositivo di input diminuisce la latenza del sistema rispetto al caso della lista unica che raccoglie la totalità delle regole. In entrambe le implementazioni che usano liste si nota una dipendenza lineare del tempo medio di attesa rispetto al numero di circuiti entranti nello switch.

Capitolo 6

Conclusioni e sviluppi futuri

L'obiettivo di questo progetto è stato quello di studiare ed implementare uno strato software di switching ATM finalizzato all'installazione su sistemi embedded dedicati al networking. Questi sistemi saranno i nodi centrali di una rete ATM. Per questa ragione, la tecnologia ATM è stata presentata con particolare riguardo alle sue funzionalità all'interno degli switch e la parte di stack dei protocolli ATM implementata comprende solamente il livello fisico ed il livello data-link.

Come modello di riferimento per la progettazione di un sistema di switching e di gestione di dispositivi ATM è stato considerato il driver model del kernel di Linux. La comprensione di questo modello ha fornito gli strumenti necessari per una valida analisi e progettazione dei componenti e delle meccaniche rilevanti all'interno del sotto sistema di switching ATM. In particolare di notevole importanza è risultato essere il meccanismo di gestione dei riferimenti validi agli oggetti del sistema, realizzati nel modello dei driver tramite l'uso dei *kobject* e parallelamente ridisegnati nel layer ATM per soddisfare i requisiti e la filosofia adottata del sistema costruito.

Un altro concetto chiave che ha guidato lo sviluppo di gestione dei dispositivi è quello di interfaccia. Questo termine assume differenti significati all'interno di questo progetto. Può riferirsi al punto di interconnessione fra un terminale utente e una rete, come ad esempio è il caso delle interfacce UNI dei dispositivi ATM, oppure fra due switch di una rete ATM, come è il caso delle interfacce NNI. Nel contesto degli strati fisici dei protocolli di rete, ci si riferisce alle schede di rete come interfacce fisiche di trasmissione. Per questa ragione un link di tipo E1 viene spesso menzionato con il termine di interfaccia E1. Dal punto di vista della progettazione software, un'interfaccia è un'astrazione di un componente software. Questo concetto è stato ampia-

mente studiato, prendendo come esempio le interfacce che il kernel di Linux usa per rendere il codice modularizzabile, ed è stato applicato con successo ai dispositivi ATM.

L'ATM Soft Switch implementato rappresenta una versione software dei dispositivi elettronici di switching ATM convenzionalmente utilizzati. Il layer software è stato disegnato tenendo presente la prospettiva reale di utilizzo dei dispositivi embedded sui quali verrà utilizzato, caratterizzato da un numero di circuiti virtuali relativamente basso e da specifiche interfacce fisiche di comunicazione. In ultima analisi, i test effettuati hanno mostrato, in modo semplificato, il diverso rendimento raggiunto dal sistema utilizzando tre diverse implementazioni per la routing table contenente le regole di commutazione.

Il progetto svolto fino ad ora corrisponde a una realizzazione a livello prototipo di uno switch ATM. Mancano infatti gli strumenti fisici di comprovazione di reali trasmissioni dati fra interfacce fisiche come quelle per le quali lo switch è stato progettato (ATM over Wimax e ATM over E1). Inoltre, a causa della mancanza di tempo, è stato impossibile realizzare una completa suite di test volti a misurare la performance del sistema variando i valori dei parametri di interesse e le risposte del sistema di fronte a situazioni di stress dovute ad un altro carico di lavoro.

Rientra pertanto negli sviluppi futuri la progettazione di accurati test del sistema volti a studiare in maniera più sistematica il comportamento e l'efficienza dell'ATM Soft Switch.

A parte questi dettagli tecnici, gli esperimenti hanno permesso di testare in dettaglio il funzionamento del software dei driver che costituiscono il sistema ATM, e ne hanno dimostrato la stabilità, la robustezza e la scalabilità.

Bibliografia

- [BC05] Daniel P. Bovet and Marco Cesati. *Understanding the Linux Kernel*. O'Really, third edition, 2005.
- [CCI91a] CCITT. B-ISDN protocol reference model and its application. Recommendation I.321 E 2054, ITU, 04 1991.
- [CCI91b] CCITT. Broadband aspects of ISDN. Recommendation I.121 E 2009, ITU, 04 1991.
- [CCI99] CCITT. B-ISDN asynchronous transfer mode functional characteristics. Recommendation I.150 E 16116, ITU, 02 1999.
- [Com90] Technical Committee. ATM on fractional e1/t1. Technical report, ATM Forum, 10 1990.
- [Com96] Technical Committee. E1 physical interface specification. Technical report, ATM Forum, 9 1996.
- [CsdkdL] versione 2.6.22 Codice sorgente del kernel di Linux. <http://www.it.kernel.org/pub/linux/kernel/v2.6/>.
- [Fora] ATM Forum. http://www.ipmplsforum.org/tech/atm_specs.shtml, marzo 2008.
- [Forb] Wimax Forum. <http://www.wimaxforum.org>, marzo 2008.
- [JCKH05] Alessandro Rubini Jonathan Corbet and Greg Kroah-Hartman. *Linux Device Drivers*. O'Reilly, third edition, 2005.
- [Jou] LINUX Journal. <http://www.linuxjournal.com/>, marzo 2008.
- [KH05] Greg Kroah-Hartman. *Linux Kernel in a Nutshell*. O'Reilly, 2005.
- [KWR05] Frank Pahlke Klaus Wehrle and Hartmut Ritter. *The LINUX Networking Architecture*. Prentice Hall, first edition, 2005.

-
- [Lov05] Robert Love. *Linux Kernel Development*. Sams Publishing, second edition, 2005.
- [PH05] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, third edition, 2005.
- [PJSP05] Michael Burian Peter Jay Salzman and Ori Pomerantz. *The Linux Kernel Module Programming Guide*. <http://opensource.org/licenses/osl.php>, 2005.
- [Sta99] William Stallings. *ISDN and Broadband ISDN with Frame Relay and ATM*. Prentice Hall, fourth edition, 1999.
- [Tan03] Andrew S. Tannenbaum. *Reti di calcolatori*. Addison Wesley, fourth edition, 2003.
- [TCR01] C. Leier T. Cormen and R. Rivest. *Introduction to algorithms*. MIT Press, second edition, 2001.
- [Tra] Kernel Trap. <http://kerneltrap.org>, marzo 2008.
- [Uni] International Telecommunication Union. <http://www.itu.int>, marzo 2008.
- [Wik] The Free Encyclopedia Wikipedia. <http://www.wikipedia.org>, marzo 2008.