Capitolo 2

Background e presentazione del progetto

2.1 Contesto progettuale

Nel capitolo precedente è stata presentata l'integrazione delle architetture MPLS e DiffServ per la QoS e si è parlato del trial sperimentale MAID, allo scopo di introdurre a grandi linee l'ambito di lavoro per questa tesi. In questo capitolo si fornisce invece un quadro più dettagliato del contesto progettuale.

All'interno del progetto TANGO è stato implementato un sistema di misura del traffico di rete [2-1] per *edge router* Diffserv over MPLS chiamato Metercontroller, la cui descrizione è oggetto dei prossimi paragrafi (2.2 e 2.3). Esso ha trovato applicazione immediata nell'architettura MAID, nella quale il compito del Metercontroller è quello di rendere disponibili in tempo

reale le misure del traffico offerto ai MA-LER DS. La conoscenza di questi dati rappresenta una base di sviluppo per nuove funzionalità avanzate per il piano di controllo del dominio MAID.
Il sistema di misura è dislocato nei MA-LER DS, e compie misure suddivise per LSP e PHB del traffico offerto, costruendone un database.
Il lavoro presentato in questa tesi di laurea consiste nell'integrazione di un server all'interno del Metercontroller e nella creazione di un'applicazione per la gestione simultanea di più client connessi (tramite socket TCP) ciascuno ad un Metercontroller residente su un diverso MA-LER DS. Tramite l'utilizzo di un client si può richiedere ad un Metercontroller di inviare una porzione arbitraria delle misure di traffico (sia scaricare parte del database che connettersi per ricevere da un certo momento in poi le misure in tempo reale), e si possono anche variare durante l'esecuzione alcuni importanti parametri di funzionamento del Metercontroller stesso. Ovviamente si è resa necessaria la contestuale definizione di opportuni protocolli per la comunicazione bilaterale tra il lato client ed il lato server, per i quali si è scelto di utilizzare il linguaggio di markup XML. Nel paragrafo 2.4 si fornisce una visione d'insieme del progetto, mentre per i dettagli implementativi si rimanda ai capitoli 4 e 5.

Il sistema costituito dall'applicazione multi-client, chiamata *Client Interface Manager* (CIM), e dal Metercontroller costituisce uno strumento flessibile che consente di monitorare l'intera frontiera del dominio MAID (o di qualsiasi altro dominio Diffserv over MPLS), con potenziali utili riscontri per il traffic engineering e per lo sfruttamento più efficiente delle risorse di rete. I dati da esso forniti in tempo reale riguardo al traffico nella rete possono consentire di effettuare sui flussi in ingresso al dominio delle

procedure di admission control basate sull'effettiva situazione di carico, oppure su predizioni statistiche del carico stesso.

2.2 Il sistema di misura

Il Metercontroller è stato concepito come strumento di gestione di un dominio DiffServ over MPLS e di supporto al traffic engineering.
Nonostante la possibilità di avvalersi di svariati strumenti liberamente disponibili per la misura del traffico (come ad esempio *magnet*, *ticket*, oppure quelli basati sulle librerie di cattura di pacchetti *libpcap* come *tcpdump*), si è preferito progettare un sistema di misura *ad hoc*. Le ragioni di questa scelta sono da ricercarsi tra gli svantaggi presentati da questo o da quello tra gli strumenti disponibili, tra cui:

- La complessità e l'eccessiva ricchezza di funzionalità, pagate con un decadimento delle prestazioni ed un sovraccarico di lavoro del sistema. In altre parole, è necessario che lo strumento di misura non sia tale da oberare il MA-LER DS, falsandone il normale funzionamento.
- L'impossibilità di effettuare la misura in un punto della catena di forwarding tale che siano già disponibili le informazioni su LSP e PHB, oppure in un punto per il quale i pacchetti presenti saranno sicuramente inoltrati (dal momento che molti degli strumenti disponibili effettuano la misura a valle di tutta la catena, prima dei blocchi di classificazione e condizionamento del traffico).
- Le pesanti modifiche del kernel di Linux richieste, che possono provocare problemi di incompatibilità o di instabilità del sistema.
- Le prestazioni scadenti nella misura dovute alla necessità di copiare ogni pacchetto dal *kernel space* allo *user space*.

Piuttosto che modificare strumenti già esistenti si è quindi scelto di crearne appositamente uno, in modo da evitare di incorrere nei problemi appena menzionati. Le misure vengono effettuate direttamente in kernel space da un modulo inserito nel kernel e chiamato *meter*. Purtroppo in kernel space non sono disponibili gli identificativi degli LSP o dei PHB, ma soltanto i parametri *fwmark* e *tcindex*, come si è visto nel paragrafo 1.2.2. Un'altra componente in user space, chiamata *metercontroller* (con l'iniziale minuscola per distinguerla dal nome dell'intero sistema di misura), provvede alla conversione da *tcindex* a PHB (questa corrispondenza è statica, quindi sempre uguale) e da *fwmark* a LSP-ID, interrogando il *daemon* RSVP-TE. Compito della componente metercontroller è anche interfacciarsi con l'esterno del sistema di misura per rendere disponibili i dati raccolti.

Le esigenze alle quali vuole rispondere il Metercontroller sono le seguenti:

- **Misura** del traffico offerto al MA-LER DS, suddiviso per LSP e per PHB. La misura viene effettuata nell'arco di una finestra temporale (alla quale ci si riferirà con il termine *window*) e con una cadenza di campionamento (che diremo *period*) configurabili durante l'esecuzione (vedi figura 2.1 a pagina seguente).
- **Salvataggio delle misure** in un database, la cui profondità temporale, ossia la quantità di campioni mantenuti in memoria, è fissata in fase di compilazione tramite la costante `HISTORY_LENGTH`. Per come sono costruite ed utilizzate alcune strutture dati, il valore di questa costante deve essere una potenza intera di due.
- **Trasparenza** del sistema di misura nei confronti del MA-LER DS, del quale non deve influenzare le prestazioni. Ogni blocco costitutivo del sistema di misura deve rispettare quest'esigenza.
- **Accuratezza della temporizzazione** del sistema di campionamento.

- **Disponibilità del timestamp** relativo ad ogni campione.
- **Interfacciamento** con le componenti di routing e MPLS e con l'RSVP-TE daemon, al fine di ricavare le informazioni su LSP e PHB.
- **Presenza di un'architettura server TCP** per la gestione di più client connessi contemporaneamente. Il server deve poter rendere disponibili ai client le misure effettuate, secondo diverse modalità di accesso al database.

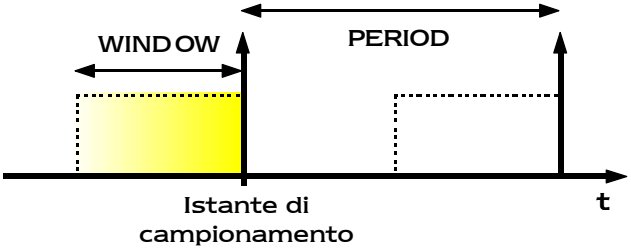

Figura 2.1 – Parametri di campionamento

2.3 Gli elementi del sistema di misura

In questo paragrafo vengono esaminate le due componenti del sistema di misura, il modulo del kernel *meter* ed il programma *metercontroller*. Si fornisce una sintetica spiegazione di come soddisfino le esigenze richieste ed una descrizione delle parti di cui a loro volta sono costituiti, nonché degli aspetti di funzionamento correlati maggiormente con il lavoro presentato in questa tesi. Per ulteriori dettagli si rimanda nuovamente il lettore interessato al riferimento bibliografico [2-1].

2.3.1 Il modulo *meter*

Come detto in precedenza, è necessario che le misure siano effettuate in kernel space per questioni prestazionali del sistema di misura. Un altro requisito è effettuare il campionamento in un punto della catena di forwarding che sia significativo, ossia a valle dei blocchi funzionali che decidono se il pacchetto in esame sarà preso in considerazione dal router oppure rifiutato, nonché a valle dei blocchi che assegnano al pacchetto i parametri *fwmark* e *tcindex*. La soluzione migliore è quella di aggiungere una funzione (dopo quelle preposte alla classificazione) alla catena di prerouting del netfilter del kernel di Linux. La funzione deve misurare la lunghezza del pacchetto in transito.
Per quanto riguarda la precisione nell'intervallo di campionamento, che è uno dei requisiti visti per il sistema di misura, è stata scelta una soluzione interamente in software ed in user space. Sarebbero stati possibili altri espedienti in kernel space o tramite apposito hardware, forse più accurati, ma avrebbero presentato problemi causando modifiche considerevoli al sistema oppure occupando una quantità cospicua di risorse.

Il modulo meter aggiunge una voce al filesystem `/proc` del sistema e misura il traffico in transito conservando per ogni pacchetto la lunghezza in byte, il *fwmark*, il *tcindex* e l'istante d'arrivo. Dallo user space è possibile consultare il filesystem per recuperare i dati della misura. Il meter rende disponibile il timestamp relativo all'istante di campionamento, seguito dalle misure cumulative riguardanti tutti i pacchetti che rientrano nella finestra temporale (*window*) precedente l'istante di campionamento, suddivise per *fwmark* e *tcindex*. Il timestamp dà un indicazione di tempo assoluto (il *wall*

clock time[1]), ed è quindi direttamente confrontabile con analoghi timestamp di altre macchine sincronizzate con quella che ha rilevato i dati. Effettuare la misura in kernel space è necessario anche dal punto di vista dell'accuratezza del timestamp: solo in questo modo è possibile associare correttamente un timestamp all'istante di campionamento.
Il meter è costruito in modo da poter cambiare i parametri *window* e *period* durante l'esecuzione; ciò avviene sempre tramite il filesystem `/proc`.

Delle esigenze richieste al sistema di misura viste nel paragrafo 2.2, il meter consente di esplicare l'effettuazione delle misure e la loro associazione ad un timestamp. Tutte le altre funzionalità sono delegate alla componente in user space metercontroller, poiché occorre non sovraccaricare eccessivamente una parte vitale del sistema quale il kernel.

2.3.2 Il software *metercontroller*

Il programma metercontroller ha una serie di compiti da svolgere in parallelo per ottemperare alle funzionalità richieste. A tale scopo è stato realizzato come un programma *multithread*. La programmazione multithread è una tecnica alternativa a quella, più tradizionale, costituita da IPC (Inter Process Communication), e serve sostanzialmente per avere in esecuzione più porzioni di codice allo stesso tempo. Ogni thread che viene creato può comunicare con il thread creatore, e, a differenza dei processi, condivide con esso lo stesso spazio di memoria, i descrittori e le altre risorse di sistema. In virtù della versatilità di questa tecnica di programmazione, si è deciso di farne ampio uso anche nel progetto presentato in questa tesi.

[1] Numero di secondi e microsecondi trascorsi dalla mezzanotte del 1/1/1970.

Il metercontroller è dunque suddiviso in blocchi operativi, ciascuno dei quali implementato da un differente thread, che saranno descritti singolarmente.

La info-unit

Come si è visto, il modulo meter, residente nel kernel, fornisce le misure suddivise secondo i parametri *tcindex* e *fwmark*. Un primo aspetto del lavoro del metercontroller è rappresentato dalla conversione di questi due parametri in LSP-ID e PHB. Il blocco che assolve questo compito è chiamato *info-unit*, e risponde all'esigenza dell'interfacciamento con i sistemi di routing del MA-LER DS.

Per quanto riguarda il parametro *tcindex*, la corrispondenza con i PHB è statica, perciò nel metercontroller è presente una semplice mappa per la conversione. Il recupero dell'identificativo di LSP comporta invece un lavoro più complesso. Le informazioni necessarie alla conversione da *fwmark* a LSP-ID sono disperse in diversi componenti del sistema, alcuni dei quali interni al kernel. La info-unit deve dapprima interfacciarsi con il sottosistema di routing del kernel e recuperare la corrispondenza tra *fwmark* ed interfaccia virtuale MPLS. Questa corrispondenza consente di risalire, consultando il sottosistema MPLS, al label che verrà apposto nello shim header dei pacchetti con un dato *fwmark*. Infine, mediante l'RSVP-TE *daemon*, si risale dal label allo LSP-ID. Si ricorda che l'identificativo di LSP nell'architettura MAID è unico per tutto il dominio, e viene comunicato dal piano di controllo ai vari nodi mediante l'utilizzo del protocollo RSVP-TE.

A partire dalle informazioni ricavate in questo modo, la info-unit genera una lista (*LSPList*) in cui memorizza la corrispondenza tra *fwmark* e LSP-ID,

che viene utilizzata dal metercontroller per associare i campioni provenienti dal meter (di cui si conosce il *fwmark*) al corretto LSP-ID.

Il sampler

Il *sampler* è il blocco principale (il thread "padre") della componente in user space metercontroller. Una volta avviato, esso genera due thread figli, dei quali uno è il blocco info-unit di cui si è già parlato, l'altro è il *controller*, blocco che sarà esaminato immediatamente dopo il sampler. La funzione principale del sampler è quella di interrogare il modulo meter periodicamente per ottenere i dati sulle misure, e salvare ad ogni ciclo tali dati in un'apposita struttura detta *Traffic DB*. Il Traffic DB è un database organizzato come una lista di vettori di traffico indicizzati rispetto allo LSP-ID. Ogni vettore ha dimensione `HISTORY_LENGTH` (costante fissata in fase di compilazione, già vista nel paragrafo 2.2), è gestito con allocazione statica di memoria ed è percorso in modalità *round robin.* In pratica è presente un indice che punta all'elemento più vecchio nel vettore, in modo da poter scrivere ciclicamente ogni nuovo campione al posto del più vecchio. Si ricorda che, ad ogni campionamento, per ogni LSP-ID vengono forniti i dati relativi a tutti i PHB, che attualmente sono otto[2]. D'ora in poi ci si riferirà col termine "campione" di un LSP alle otto misure, una per PHB, riguardanti quel determinato LSP.

Per poter creare ed aggiornare il Traffic DB, il sampler, che riceve dal meter i campioni suddivisi per *fwmark* e *tcindex*, deve avvalersi della comunicazione con la info-unit per risalire agli LSP-ID ed ai PHB.

[2] Per la precisione, i PHB attualmente supportati nel trial sono BE, EF e sei della famiglia AF.

Parallelamente al Traffic DB, il sampler mantiene un vettore dei timestamps relativi ai campioni, che ha dimensione `HISTORY_LENGTH`, è statico ed è gestito in modo round robin come i vettori del Traffic DB.

Il funzionamento del sampler è ciclico, vista la natura delle operazioni che deve svolgere (accesso periodico ai dati del meter e trattamento di tali dati). Il ciclo principale consiste nelle seguenti azioni:

- **Campionamento**, ossia lettura delle informazioni fornite dal sampler, che consiste in una lettura da file. I dati rappresentano il numero di bytes offerti al MA-LER DS nell'intervallo di tempo pari a *window* secondi prima dell'istante di campionamento, per *fwmark* e *tcindex*.
- **Conversione** dei parametri *fwmark* e *tcindex* in LSP-ID e PHB, mediante consultazione della info-unit, in particolare della LSPList.
- **Aggiornamento del Traffic DB**. Si è già visto come il sampler memorizzi le informazioni nella struttura Traffic DB; nel caso però in cui i dati del DB siano richiesti da un client, il blocco controller dovrà effettuarvi una lettura, ed in tale frangente non è possibile la scrittura da parte del sampler. Per ovviare a questo inconveniente il sampler è dotato di una struttura analoga al Traffic DB, detta *Cache DB*. Essa è realizzata nella stessa maniera, con l'unica differenza della minore dimensione. Quando il Traffic DB è in uso da parte del controller, il sampler scrive provvisoriamente nel Cache DB.
- **Invio delle misure in tempo reale**, tramite scrittura sugli opportuni descrittori, ai client che ne abbiano eventualmente fatto richiesta. A questo scopo esiste una struttura, detta *DescriptorList*, dove si mantiene traccia di tutti i client che desiderano ricevere i dati in tempo reale per ogni LSP. I client sono rappresentati da descrittori di socket; la parte relativa alla connessione remota è gestita dal controller, dove risiede il server TCP implementato nel lavoro per la presente tesi. Nella

DescriptorList vengono immagazzinate anche le informazioni riguardo ai dati ai quali ogni client è interessato.

- **Messa in fase di sleeping del processo**; questa è la fase cruciale per la temporizzazione del campionamento, che deve essere eseguito a intervalli di *period* secondi. Il problema correlato con quest'operazione è il fatto che le azioni compiute dal sampler prima della messa in fase di sleeping impiegano un tempo variabile (in modo non facilmente predicibile) ad ogni ciclo. Come si è detto nel paragrafo 2.3.1, si è preferito adottare una soluzione interamente in software ed in user space. È stato messo a punto un algoritmo empirico per stimare l'istante esatto di campionamento e calcolare quindi il lasso di tempo in cui il sampler deve essere messo in fase di sleeping. L'algoritmo, che non implica attese attive nell'esecuzione del programma, si è rivelato valido e prestazionale nelle prove sperimentali eseguite.

Le strutture sulle quali si basa la comunicazione tra sampler, info-unit e controller sono gestite in mutua esclusione dai diversi thread. Opportuni semafori regolano l'accesso a tali strutture permettendo ad un solo thread alla volta il via libera. Questo è necessario per evitare che due thread tentino contemporaneamente di accedere alla stessa struttura, causando un blocco irreversibile dell'esecuzione del programma che nella terminologia dei thread viene chiamato *race condition.*
Le funzioni per l'accesso al database di traffico sono state integrate in una libreria che è risultata di estrema utilità in fase di implementazione del lato client, come si vedrà nel capitolo 4.

Riassumendo, il sampler risponde alle esigenze di salvataggio delle misure in un database di traffico, al controllo dell'accuratezza degli istanti di

campionamento ed in parte all'invio dei dati ai client.

Il controller

Il controller è l'ultimo blocco funzionale del sistema di misura, e rappresenta il punto di contatto tra il lavoro già presente e quello sviluppato per questa tesi. Esso risponde all'esigenza di rendere disponibili i dati relativi al traffico all'esterno del sistema di misura (ad esempio al piano di controllo del dominio MAID, per le applicazioni di admission control di cui si è già parlato). In realtà è prevista la possibilità che ci siano più blocchi di questo tipo attivi contemporaneamente, uno per ogni client che richieda l'accesso alle misure di traffico.

La comunicazione con il sampler e l'arbitraggio per l'accesso concorrente di più controller alle misure sono funzionalità già implementate; nell'ambito del progetto presentato in questa tesi si è provveduto a realizzare l'unica mancante, ossia rendere il controller un server al quale si possa connettere un client remoto, attraverso dei socket TCP.

Mediante opportuna segnalazione (che sarà illustrata in dettaglio nei capitoli 4 e 5) il client richiede una parte delle misure, ed il controller comunica la richiesta al sampler. L'accesso ai dati può essere suddiviso in due categorie fondamentali:

- **Accesso *snapshot***: con questo tipo di richiesta si desidera, interamente o in parte, ricevere esclusivamente lo storico delle misure di traffico, ossia i dati relativi ai campioni passati, fino a quello attuale.
- **Accesso *real-time***: in questo caso si è invece interessati a ricevere, dal momento della richiesta in poi, le misure in tempo reale ad ogni istante di campionamento.

I due tipi di accesso possono essere combinati assieme in una richiesta; le

richieste di tipo snapshot sono gestite dal controller tramite la chiamata ad una funzione che legge dal Traffic DB ed invia direttamente le misure; quelle di tipo real-time sono invece esaudite dal sampler (che ne viene a conoscenza mediante lettura della DescriptorList) ad ogni suo ciclo di funzionamento.

È inoltre possibile scegliere di ricevere dati su alcuni o tutti gli LSP, secondo le due seguenti modalità:

- **Accesso *selective***: si richiedono le misure relative a singoli LSP, scelti di volta in volta; dal momento che in ogni singola richiesta se ne può selezionare al massimo uno, serviranno tante richieste quanti sono gli LSP a cui si è interessati.
- **Accesso *full***: si richiedono le misure relative a tutti gli LSP, in una singola richiesta.

I parametri *window* e *period* possono essere variati durante l'esecuzione, con un opportuno messaggio da parte del client. Le possibili richieste effettuabili al controller sono elencate e descritte dettagliatamente nel paragrafo 4.2, assieme alle regole per lo scambio di messaggi.

2.4 Integrazione del progetto nel sistema di misura

Il sistema di misura presentato nei precedenti paragrafi è stato pensato in primo luogo in relazione ad applicazioni di monitoring e predizione statistica del traffico, da eseguire generalmente su macchine diverse da quelle dove vengono effettuati i rilevamenti. Infatti è ragionevole supporre che gli algoritmi di predizione richiedano complesse elaborazioni, per cui può risultare opportuno demandare il lavoro del monitoring e delle predizioni ad un calcolatore dedicato.

Il progetto presentato in questa tesi di laurea ha due obiettivi fondamentali:

- Dotare il Metercontroller di una struttura di server TCP, tramite la quale rendere disponibili all'esterno le misure di traffico effettuate, e definire i protocolli di accesso al server.
- Progettare ed implementare un sistema software che gestisca simultaneamente più client, ciascuno connesso al server di un sistema di misura presente su un diverso MA-LER DS, ed i protocolli di accesso e configurazione di tale sistema. Ogni client effettua sia il monitoring che le predizioni statistiche per il traffico presente sul MA-LER DS al quale è connesso. Questo sistema consente di avere sotto controllo la situazione di carico di tutti gli *edge router* del dominio MAID e del traffico per ogni LSP e PHB, e rappresenta quindi uno strumento su cui poter basare applicazioni di traffic engineering.

Per entrambi gli scopi è stato utilizzato il linguaggio di programmazione C. Ogni client per il pilotaggio remoto del Metercontroller è chiamato *Client Interface* (CI); il sistema per la gestione di client multipli è chiamato *Client Interface Manager* (CIM). Un CI comunica con il server TCP presente in un blocco controller del sistema di misura. Dapprima invia le richieste di accesso ai dati di traffico, quindi riceve le repliche a tali richieste ed i dati di traffico veri e propri. La comunicazione avviene su tre socket TCP distinti. Il primo, detto *control* socket, serve per lo scambio delle richieste e delle repliche, che indicano se le richieste sono andate a buon fine oppure se ci sono stati degli errori. Il protocollo di comunicazione sul control socket prevede l'utilizzo del linguaggio XML per la messaggistica. Gli altri due socket, chiamati *snapshot* e *real-time* socket, sono utilizzati rispettivamente dal controller e dal sampler per l'invio delle misure vere e proprie, sotto forma di dati binari. In figura 2.2 a pagina seguente si può vedere la

rappresentazione schematica di un CI connesso ad un Metercontroller.

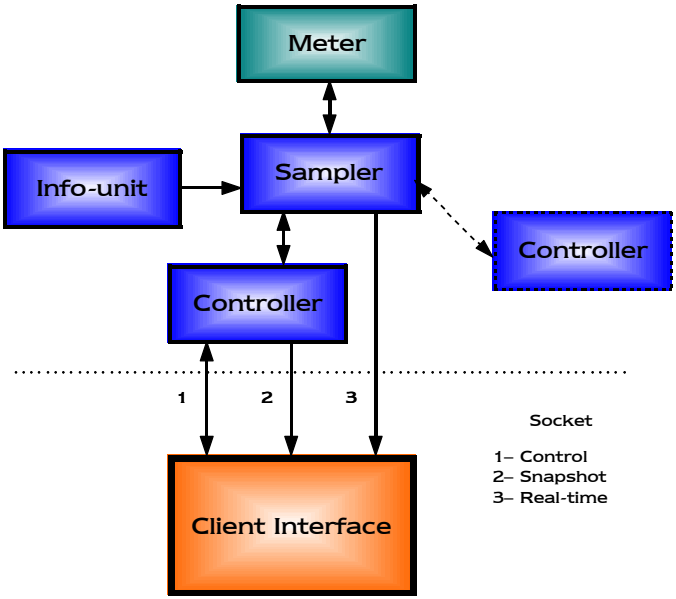

Figura 2.2 – Interfaccia tra CI e Metercontroller

Il Client Interface Manager gestisce contemporaneamente più moduli CI, connessi ciascuno ad un differente MA-LER DS, come si può vedere in figura 2.3 a pagina seguente. Le operazioni che può eseguire in tal senso sono innanzitutto la creazione o la cancellazione di un CI e lo smistamento delle richieste dirette ad un Metercontroller verso il corretto modulo CI (ossia quello connesso al MA-LER DS di interesse). È possibile fissare delle soglie di attenzione per le misure sul traffico di ogni LSP, ed il CIM segnalerà il superamento di tali soglie. Inoltre CI e CIM forniscono il supporto per eseguire le predizioni statistiche tramite un protocollo generalizzato che permette di rinnovare o cambiare gli algoritmi di stima

senza dover progettare nuovamente il sistema. Il CIM è stato realizzato come un programma multihread, come si vedrà con dettaglio nel capitolo 4.

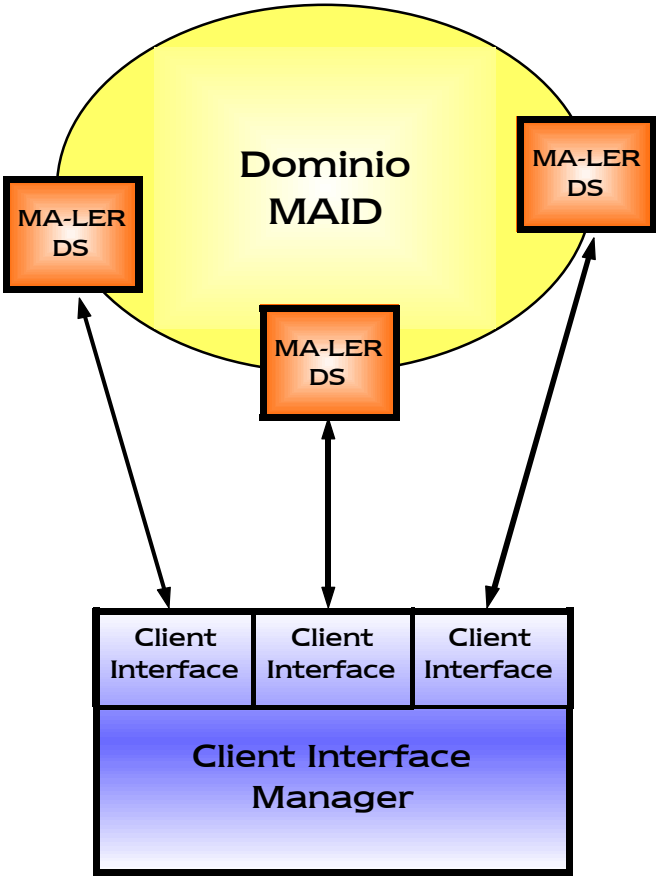

Figura 2.3 – Il Client Interface Manager

Il CIM ha anch'esso un'architettura server TCP, e può ricevere comandi in modo locale o remoto da un opportuno client, il quale rappresenta l'elemento finale della catena di applicazioni e deve interfacciarsi con

l'utente, amministratore di rete o chiunque egli sia. Per questo motivo è auspicabile la realizzazione di un'interfaccia grafica (UI, User Interface) per un agevole e rapido accesso, e per una presentazione efficiente dei dati. Lo sviluppo di una UI non è infatti compreso in questo lavoro di tesi, ma nei capitoli 4 e 5 si forniscono le istruzioni dettagliate riguardo alla struttura ed ai protocolli che esso deve utilizzare per comunicare con il CIM. La comunicazione viene effettuata su due socket TCP, uno per le operazioni di controllo detto *master* socket, l'altro per la gestione delle operazioni di predizione statistica e delle soglie per le misure e per le stime, chiamato *estimate* socket.

Le operazioni di controllo consistono nel pilotaggio diretto del CIM e indiretto dei Metercontroller mediante i CI. Le operazioni statistiche consistono nella configurazione di uno stimatore per il campione successivo per gli LSP desiderati. Le soglie possono essere attivate sia per le misure vere e proprie che per le stime del traffico futuro (uscita dei predittori).

L'interfaccia tra CIM e UI è mostrata in figura 2.4.

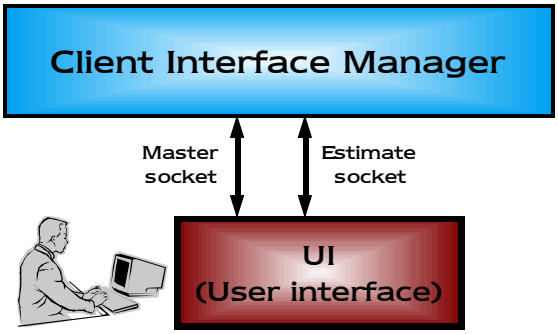

Figura 2.4 – Interfaccia di controllo del CIM

Come accade per il control socket nel controller, anche tutti i protocolli di

accesso al CIM sono basati sul linguaggio XML. Nel capitolo 3 si parlerà estensivamente di questo linguaggio, in modo da evidenziare le ragioni del suo impiego.