UNIVERSITÀ DEGLI STUDI DI BOLOGNA

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica
Reti di Calcolatori

Protocolli di Controllo e Monitoraggio Per Servizi Multimediali

Tesi di Laurea di:

Paolo Battarra

Relatore:

Chiar.mo Prof. Ing. Antonio Corradi

Anno Accademico 2002 – 2003

INDICE

Introduzione

Al giorno d'oggi stanno sempre più spesso abbiamo a che fare con applicazioni interattive, piene di immagini e suoni, che spesso richiedono l'utilizzo di contenuti multimediali. Questa situazione è sotto i nostri occhi ovunque e sta espandendosi a macchia d'olio, invadendo diversi settori di interesse. Può così succedere di riuscire a scaricare via Web una canzone in formato mp3 o di imbattersi, sempre in Internet, in siti che ci permettono di vedere foto, video e filmati. Ma non sono solo i settori tradizionali ad essere soggetti a questo fenomeno di sviluppo. Ad esempio si è ormai arrivati ad avere a che fare con questo tipo di applicazioni anche nell'ambito della telefonia cellulare, dal momento che si possono inviare e ricevere immagini e musiche. Fino a qualche tempo fa molte di queste cose erano addirittura impensate ed è perciò semplice immaginare quanti passi in avanti si sono dovuti fare, in questo senso, nello sviluppare tecnologie per supportare queste applicazioni multimediali.

Il problema maggiore che si deve affrontare in questa fase di studio e sviluppo è quello legato alla richiesta di risorse necessarie per la fruizione dei contenuti multimediali. Queste applicazioni infatti necessitano di un ingente quantitativo di risorse, siano esse computazionali o di altra natura e proprio questo aspetto ha spesso rappresentato un freno ad una loro ancor più rapida diffusione. Inoltre molto spesso si affianca a questo problema anche quello di una grave carenza di strumenti adeguati per la gestione di queste risorse. Fortunatamente negli ultimi anni si sono studiati diversi tipi di protocolli che hanno proprio lo scopo di fornire questo genere di strumenti.

Il nostro lavoro di tesi si prefigge proprio l'obiettivo di andare a studiare questa categoria di protocolli, con l'intento di andare poi ad utilizzare le loro funzionalità nella realizzazione di un modulo di controllo e monitoraggio delle risorse all'interno di una preesistente piattaforma per la gestione di flussi multimediali, chiamata MUM. Il nostro componente dovrà effettuare il monitoraggio delle risorse e segnalare eventuali situazioni di deterioramento delle prestazioni dovute ad un peggioramento

nella disponibilità delle risorse stesse. In particolare la nostra attenzione sarà focalizzata su una risorsa specifica che caratterizza la fruizione di flussi multimediali, ossia la banda trasmissiva.

La tesi sarà organizzata come segue. Nel capitolo 1 si introducono quelle che sono le problematiche legate al Network Management, si danno alcune definizioni utili per la successiva comprensione e si delineano i principi architetturali che dovranno essere seguiti nella realizzazione del componente. Nel capitolo 2 si presentano i protocolli studiati, ovvero RTP/RTCP e SNMP, descrivendone le caratteristiche e le funzionalità offerte. Nel capitolo 3 si passa a descrivere tutte quelle che sono le tecnologie impiegate nello sviluppo del nostro lavoro, dall'ambiente entro il quale operiamo, cioè SOMA e MUM, a quelle librerie che ci permettono di utilizzare i protocolli, ossia JMF ed SNMP Api di AdventNet. I capitoli 4 e 5 sono quelli direttamente collegati al nostro lavoro di progetto. Nel capitolo 4 si riassumono i risultati della fase di analisi del problema da affrontare. Nel capitolo 5 si passa invece a trattare la fase di progetto vera e propria e lo si conclude andando a trattare le problematiche legate all'implementazione ed al testing.

Capitolo 1

1 Network Management e architetture di supporto per applicazioni multimediali

In questo capitolo verranno introdotti alcuni concetti che saranno poi utili a comprendere il lavoro svolto in seguito. Per prima cosa ci si soffermerà su argomenti attinenti a quelle che sono le problematiche legate alla gestione di una rete. Verranno poi presentati anche alcuni parametri utili al controllo che si intende realizzare sui flussi multimediali, si accennerà alle risorse fisiche necessarie alle applicazioni multimediali ed infine si spiegherà quali dovrebbero essere i principi da seguire nella realizzazione di una architettura volta a supportare applicazioni multimediali.

1.1 Network Managent

Formalmente il network management può essere definito come il processo di controllo di reti complesse, al fine di massimizzarne efficienza e produttività e di minimizzarne i costi. Con questo obiettivo un amministratore di rete raccoglie informazioni e configura gli elementi di rete nel suo dominio. Un elemento di rete è un qualsiasi dispositivo che è collegato e può comunicare tramite rete. Un dominio, invece, può essere stabilito in modo arbitrario, sebbene sia generalmente definito da vincoli pratici; ad esempio, se i dispositivi su una rete locale sono parte di un dominio, quelli su un'altra rete che devono essere raggiunti tramite router, saranno parte di un altro dominio. Nella terminologia del network management un pezzo di codice in esecuzione su un elemento di rete, che fornisce informazioni relative all'hardware della macchina, viene chiamato *agente*. Un gestore di rete, in generale, può accedere alle informazioni sui dispositivi grazie a questi agenti.

Le funzioni di gestione di una rete possono essere ricondotte a due categorie: monitoraggio e controllo. Il network monitoring consiste nell'osservare e analizzare lo stato della rete ed è un aspetto essenziale del network management automatizzato. Le informazioni da reperire includono informazioni statiche, relative alla configurazione, come l'identificazione dei vari dispositivi di una rete (network

discovery); informazioni dinamiche associate agli eventi nella rete, quali le notifiche di interruzione di un servizio, e informazioni statistiche, derivate da quelle dinamiche, utili per stabilire trend di utilizzo. Il monitoraggio coinvolge anche l'identificazione degli errori e la determinazione della loro causa, per poi prendere decisioni correttive, ma richiede anche un intervento preventivo nei confronti di errori imminenti e la loro minimizzazione. Il controllo di rete è invece responsabile del cambiamento dei parametri associati ai componenti e deve far sì che i dispositivi eseguano azioni predefinite. L'area di configurazione comprende una varietà di funzioni relative alla rete e ai suoi elementi: questo include inizializzazione, mantenimento e shutdown dei componenti singoli e dei sottosistemi logici. Nell'area della sicurezza, ad esempio, il sistema di network management ha la responsabilità di coordinare e controllare i meccanismi finalizzati alla protezione delle risorse d'utente e di sistema, includendo il network management stesso.

1.2 Problematiche del Network Managent

Ci sono diverse attività che caratterizzano la gestione di una rete complessa ed impegnano il suo amministratore [Rete]. Uno degli aspetti più critici di questo Network Management è la configurazione della rete stessa. I problemi da superare sono legati alla necessità di confrontarsi con reti sempre più eterogenee in cui vengono impiegati tecnologie ed apparati sempre più complessi. Il problema si complica ulteriormente quando diversi gruppi di persone operano su porzioni distinte della rete. In tali condizioni è spesso difficile mantenere la consistenza nella configurazione degli apparati che andrebbero invece configurati in modo congruente. Tale problema è particolarmente sentito ogni volta che si rendono necessarie modifiche di configurazione, durante le quali diventa particolarmente difficile mantenere la consistenza tra le configurazioni dei diversi apparati. Ad oggi mancano, purtroppo, tecnologie per automatizzare la configurazione della rete e l'obiettivo di avere reti autoconfigurabili sembra ancora abbastanza distante.

Vi è poi una seconda grande criticità nella gestione della rete che è legata all'individuazione di un guasto sulla rete. Molti guasti infatti non sono facilmente individuabili anche se si passa ad un "monitoring" automatizzato della rete, dal momento che è molto comune la loro propagazione all'interno della rete che può

generare un elevatissimo numero di “sintomi”. In pratica ci si trova ad avere numerose disfunzioni e ad un sensibile calo delle prestazioni, senza poter risalire in maniera semplice alla loro fonte del problema. La propagazione va normalmente dal basso verso l'alto, cioè i problemi di livello fisico si propagano verso i sistemi terminali e le applicazioni.

Si deve quindi capire come si possono correlare tra loro i sintomi in modo da isolare il problema permettendo quindi di andare a porvi rimedio in tempi contenuti. A questo scopo si cerca sempre di più di automatizzare questo aspetto. Per quanto le tecnologie di rete diventino nel tempo sempre più affidabili dal punto di vista hardware, l’esplosivo aumento della complessità delle reti accresce il rischio di malfunzionamenti o inefficienze dovuti ad errori di configurazione e problemi software.

1.2.1 Ripercussioni economiche del Network Management

L’aumento del rischio di guasti ed inefficienze ha diverse cause tra le quali l’aumento della dimensione e della complessità delle reti e l’aumento della velocità di cambiamento. Inoltre nelle reti attuali l’effetto dei malfunzionamenti diventa potenzialmente più grave a causa del sempre maggiore impiego di applicazioni distribuite per assolvere compiti di importanza fondamentale in molte aziende. Al giorno d’oggi aeroporti, borse valori, banche ed aziende finanziarie, un numero crescente di aziende sanitarie e varie altre imprese dipendono pesantemente dal funzionamento della propria rete di telecomunicazione. Quanto sia forte questa dipendenza è dimostrato dagli effetti catastrofici del collasso della rete sulla operatività delle aziende più disparate. Fra gli esempi più famosi si possono citare la paralisi dell’aeroporto di NY nel 1992 e, sempre nello stesso anno, il blocco del sistema di dispatching per le ambulanze a Londra. Risulta evidente quindi quanto possano essere rilevanti, oltre ai disagi provocati, le perdite economiche che questi fenomeni possono provocare.

Un secondo importante obiettivo della gestione della rete è il contenimento dei costi di esercizio. La rilevanza di questo aspetto si deduce immediatamente dall’analisi dei costi dei sistemi informativi aziendali. Infatti i costi di esercizio della rete e dei sistemi costituiscono almeno i due terzi del totale, e possono crescere i

alcuni casi fino ad una percentuale del 90%. Tale fattore è destinato ad accrescere ulteriormente la propria importanza in conseguenza dell'evoluzione tecnologica che produce un continuo calo del costo degli apparati, mentre il costo del personale, particolarmente se dotato di conoscenze tecniche specifiche, tende continuamente a crescere.

L'automatizzazione della gestione della rete tende quindi a ridurre il più importante fattore di costo di un sistema informativo, e non è un caso che, nello stesso scenario organizzativo ed economico, si stiano affermando soluzioni quali i Network Computer che mirano ad abbattere l'altro fattore che determina il costo complessivo di un sistema informativo, ossia il costo di gestione dei sistemi e delle applicazioni.

Considerando poi il caso specifico di un gestore della rete risulta evidente che questo abbattimento dei costi consente di formulare offerte economicamente convenienti per gli utenti, e quindi vincenti in uno scenario caratterizzato da una competizione sempre maggiore.

1.3 Principi alla base delle applicazioni multimediali

Alla base di una applicazione multimediale vi sono vari concetti che è utile capire per poi comprendere il proseguo del lavoro [MUM].

1.3.1 Flusso

Il flusso è una astrazione che viene introdotta per specificare la fruizione, cioè il recapito ed il rendering, di un singolo oggetto multimediale, ossia un generico flusso di dati caratterizzato dal fatto di variare con continuità e di essere soggetto a vincoli di tempo, come potrebbero essere ad esempio un video o un audio.

Questa definizione riveste una importanza fondamentale quando si comincia a trattare le problematiche della qualità del servizio (Quality of Service, indicata anche con l'acronimo QoS). Il singolo flusso infatti può essere considerato come l'unità elementare alla quale riferire la QoS, dal momento che sono anche l'entità minima identificabile per la trasmissione di oggetti multimediali. I flussi possono essere sia di tipo *unicast* (da una sorgente ad un destinatario) che di tipo *multicast* (da una

sorgente a più destinatari). Ogni flusso, in generale, transita nel suo percorso attraverso diverse entità che dovranno garantire ognuna, in un'ottica di QoS, delle risorse per processarlo.

I flussi sono per loro natura continui, ma per permettere ad un calcolatore di utilizzarli devono essere discretizzati e quantizzati. Ne consegue che possiamo considerare un flusso come una sequenza discreta di dati soggetti a vincoli di tempo. In questa ottica diventano rilevanti il ritardo con il quale questi dati arrivano al ricevente, dal momento che questo ne determina la validità. Una semantica così formulata viene detta isocrona, ed isocroni sono, per loro natura, i flussi multimediali.

1.3.2 Larghezza di banda

La larghezza di banda è il parametro che misura la portata di un canale trasmissivo digitale, poiché definisce il numero massimo da dati che possono transitare nella sezione nell'unità di tempo. Tale grandezza si misura usualmente in bit/secondo (bps) e risulta un parametro molto stringente per quello che riguarda i flussi multimediali. Questi infatti richiedono solitamente una banda molto ampia e questa richiesta si può diminuire solamente utilizzando opportuni algoritmi di compressione che spesso però richiedono, in particolare dal lato sorgente, una ulteriore computazione anche piuttosto consistente.

1.3.3 Parametri caratterizzanti i flussi

La **latenza** è il parametro che misura il ritardo introdotto dalla linea di trasmissione, cioè il tempo impiegato dai dati discretizzati ed impacchettati per giungere dalla sorgente al destinatario. Come accennato in precedenza tale parametro riveste un ruolo fondamentale quando si ha a che fare con flussi multimediali ed in particolare nelle applicazioni che necessitano di una interazione con un sistema remoto, come ad esempio per una conversazione, per la quale è stato valutato un ritardo massimo tollerabile pari a 150 ms. Come risulta ovvio il valore della latenza non è mai costante, ma è soggetto a continue variazioni e sarà quindi rappresentata da una variabile aleatoria.

Per questo motivo si prende in considerazione un altro parametro: il **jitter**. Questi misura la varianza della latenza, cioè lo scostamento di questa dal suo valore medio (fig 1.1). La situazione ideale si verificherebbe se tutti i pacchetti giungessero al ricevente nello stesso ordine nel quale sono stati generari, e venissero processati con la stessa velocità con la quale arrivano. In tal caso non ci sarebbe bisogno di una bufferizzazione dei dati arrivati al ricevente. In realtà questo non si verifica e può succedere sia che i pacchetti arrivino in un ordine diverso da quello con cui sono partiti.

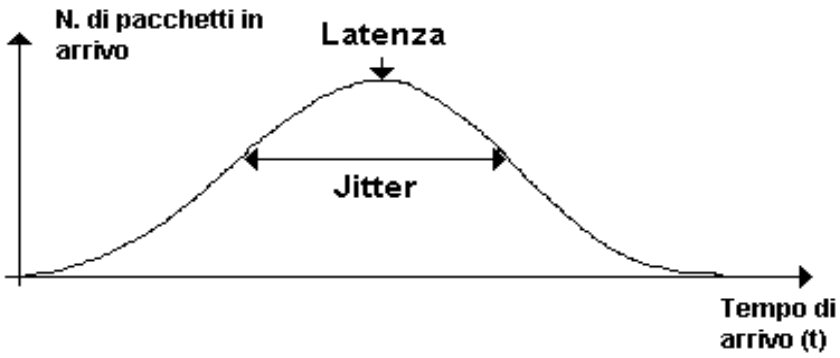

Figura 1.1: Latenza dei pacchetti

Inoltre, siccome la latenza è una variabile aleatoria, si dovranno bufferizzare i dati in modo da tollerare una certa varianza della latenza stessa. In particolare proprio il jitter assume particolare importanza in relazione a questo problema del dimensionamento del buffer di ricezione. Infine questo parametro fornisce un'informazione importante per valutare lo stato corrente del carico del canale trasmissivo. Come già detto in precedenza, i flussi multimediali sono isocroni e l'arrivo con troppo ritardo di un pacchetto, ne determina l'inutilità. Fortunatamente però molte applicazioni multimediali possono tollerare delle perdite di dati.

Un altro parametro molto importante per descrivere un flusso multimediale è il **loss rate** che quantifica la percentuale di dati "perdibile" in un flusso. Tale parametro è ortogonale agli altri dal momento che possono esistere flussi diversi con la stessa occupazione di banda e bitter, ma con loss rate molto diverse.

1.4 Carenza di risorse fisiche

Spesso le applicazioni multimediale richiedono, per il loro utilizzo, una grande quantità di risorse. Questo aspetto limita molto la diffusione di questo tipo di applicazioni. In particolare le risorse che più delle altre sono da considerare sono tre: la CPU, la banda trasmissiva e la memoria.

La **CPU** è la risorsa più contesa perché viene coinvolta in diverse fasi del recapito della presentazione multimediale:

- impacchettamento/spacchettamento dei dati;
- codificazione/decodificazione del flusso (cioè l'esecuzione di algoritmi di compressione/decompressione);
- rendering dell'oggetto multimediale.

Queste sono solamente alcune delle operazioni più costose operate durante la trasmissione di un flusso multimediale. Solitamente i requisiti, per un dato tipo di CPU, vengono espressi come frazioni di tempo di un periodo, oppure sotto forma di percentuale dei cicli del microprocessore necessari.

La **banda trasmissiva** è un'altra risorsa che solitamente viene fortemente richiesta. Per rendersene conto basta ricordare che un filmato non compresso con qualità video TV standard richiede di solito una banda pari circa a 120 Mbps (Mega bit per second), che eccede la capacità di molte delle Ethernet oggi utilizzate, pari a 100 Mbps. Come già accennato in precedenza, l'utilizzo degli algoritmi di compressione riduce notevolmente la richiesta di banda. Per esempio, utilizzando la prima delle codifiche studiate dal Moving Picture Experts Group (MPEG), cioè MPEG-1, possiamo abbassare questo valore a 1,5 Mbps. Naturalmente queste trasformazioni richiedono un costo, cioè pesano maggiormente sulla risorsa CPU.

L'ultima risorsa che consideriamo è la **memoria**. La trasmissione e visualizzazione dei flussi multimediali richiede un'elevata quantità di memoria. Questa viene usata ad esempio quando il ricevente deve bufferizzare i dati in arrivo per poter continuare la visualizzazione dell'oggetto multimediale, anche in presenza di momentanei problemi di trasmissione. Inoltre ci sarà bisogno di altri buffer per salvare i dati durante i processi di codifica/decodifica, conseguenti alle operazioni di comprensione/decompressione dei dati, e per operare il rendering degli oggetti multimediali in arrivo. Questa risorsa comunque produce meno problemi rispetto alle due precedenti dal momento che solitamente ce n’è una disponibilità maggiore e difficilmente si arriva ad averne carenza.

1.5 Principi per una architettura di supporto per applicazioni multimediali

In questa parte si vogliono introdurre quelli che sono i principi che devono essere seguiti quando si va a realizzare una struttura per il supporto ad applicazioni di tipo multimediale. In particolare questo lavoro di tesi non si prefigge la realizzazione di un intero sistema con queste caratteristiche, ma comunque è bene tenere presente questi principi anche nella sintesi di un piccolo modulo che andrà comunque ad operare in questo contesto. In particolare si vuole andare a porre l'accento su quegli aspetti che influenzano il problema della gestione e del monitoraggio della rete. E' bene ricordare infine che quasi tutti questi principi ricalcano quelli che, più in generale, sono i principi caratterizzanti il processo di ingegnerizzazione del software.

1.5.1 Modularità

Il principio di modularità consiste nel suddividere il prodotto in sottoparti (o moduli). Questo principio si può applicare sia nella direzione verticale che in quella orizzontale. La modularità verticale consiste nel creare moduli indipendenti dalla piattaforma sottostante e quindi in maniera verticale. In questo modo un cambiamento di tale piattaforma richiede il minimo sforzo. La modularità orizzontale suggerisce invece di raggruppare funzionalità simili in uno stesso modulo. Si deve notare come lo strutturare un sistema in moduli assicuri anche incapsulamento, con vantaggi notevoli. Questo principio è oggi largamente adottato, come ad esempio in tutti i moderni sistemi operativi che sono costruiti su una architettura a microkernel.

1.5.2 Flessibilità

Il principio di flessibilità consiste nel predisporre il nostro sistema in modo che possa essere riconfigurato a seconda delle necessità e del contesto esterno. Naturalmente l'adozione di questo principio presuppone il precedente, senza il quale non può essere realizzato. Questo principio risulta fondamentale, ad esempio, quando ci si trova di fronte ad una sostanziale variazione in quella che è la disponibilità delle

risorse. A quel punto, infatti, la possibilità di poter riconfigurare dinamicamente il sistema in modo da avere minore necessità di risorse, come per esempio l'utilizzo della CPU, potrebbe risolvere una situazione altrimenti critica.

La flessibilità comporta un costo sia in termini di progetto di un'architettura che applichi questo principio che in termini di overhead dovuto ai tempi di inizializzazione e riconfigurazione

1.5.3 Scalabilità

La scalabilità è la capacità di un sistema di rimanere utilizzabile all'aumentare delle dimensioni dello stesso. E' un principio architetturale di grande importanza quando di vogliono progettare dei sistemi che devono servire una utenza molto ampia. Bisogna però notare che per il loro alto fabbisogno di risorse, i sistemi che servono le applicazioni multimediali sono solitamente poco o per nulla scalabili.

1.5.4 Prevedibilità

Il principio di prevedibilità consiste nella capacità del sistema di fare prevedere il proprio comportamento. Nel campo delle applicazioni multimediali si traduce nella capacità di fare in modo che i contratti stipulati fra il supporto e l'applicativo per richiedere l'uso delle risorse vengano rispettati. Questo principio è alla base di tutti i meccanismi di tipo pro-attivo. Questi meccanismi sono statici e prevedono la prenotazione e l'occupazione preventiva delle risorse prima che il sistema cominci ad erogare il servizio. Inoltre vi è inizialmente una fase di negoziazione nella quale si tiene conto anche delle possibili variazioni nella disponibilità di queste risorse che le applicazioni possono sopportare. A questa fase ne segue un'altra nella quale viene stipulato un contratto fra l'applicazione ed il sistema. Inoltre, secondo questo principio, il supporto si deve occupare della gestione della qualità del servizio su più livelli, anche se le soluzioni esistenti sono in maggioranza sviluppate al livello applicativo. Questo sistema deve poi essere in grado di gestire tutte le risorse attraversare da un flusso, anche se queste sono di tipi diversi. Purtroppo non sarà mai possibile costruire un prodotto che sia completamente prevedibile, nel quale cioè si possa avere la certezza che ogni risorsa richiesta e prenotata da una certa applicazione

sarà sempre disponibile per tutta la durata dell'erogazione del contenuto multimediale. Proprio per questo motivo nella maggioranza dei casi si rende necessaria una attività di monitoraggio sulle risorse, in modo da poter reagire in maniera adeguata ad un'eccessiva variazione della loro disponibilità. Meccanismi di prenotazione delle risorse che prevedono questa fase di negoziazione dinamica delle risorse, a seguito di un degrado di queste, rilevato tramite una attività di controllo e monitoraggio, sono detti di tipo reattivo. Questi approcci al problema della disponibilità ed alla prenotazione delle risorse non sono mutuamente escludenti, anzi è buona norma seguire un approccio ibrido che tenga conto di entrambi.

1.5.5 Minima Intrusione

Il principio di minima intrusione si applica soprattutto alle attività di controllo e monitoraggio ed è quindi fondamentale per quelli che sono i nostri scopi. Esso stabilisce che il monitoraggio, ossia l'elemento fondamentale di ogni approccio reattivo, sia il meno intrusivo possibile. Per intrusivo si intende la capacità di andare ad influenzare ed a perturbare quella grandezza che si vuole invece monitorare. E' infatti noto che ogni volta che si opera una misurazione, si modifica lo stato del sistema che si è andati a misurare. In particolare, nel caso di un monitoraggio in un sistema distribuito, questo problema è aggravato dal fatto che il monitor utilizza, esso stesso, una parte di quelle che sono le risorse che sta monitorando.

A questo punto diventa chiara la necessità di studiare a fondo le problematiche legate al monitoraggio delle risorse, affinché esso non diventi troppo invasivo ed intrusivo e venga in definitiva seguito questo principio. Questa attività va quindi strutturata bene e si deve inoltre tenere ben presente che se il monitoraggio viene attuato in un momento di difficoltà del sistema, quando cioè le risorse cominciano a scarseggiare e la situazione peggiora, questa, se mal attuato, può far peggiorare ulteriormente le cose.

1.5.6 Adattabilità

Il concetto di adattabilità è di fondamentale importanza quando si affrontano le problematiche del monitoraggio. Esso è infatti alla base do ogni approccio di tipo

reattivo. Come già accennato in precedenza, utilizzando solamente un approccio di tipo pro-attivo, non si riesce sempre a garantire, a causa dell'impossibilità di costruire un sistema completamente prevedibile, che il sistema sia in grado di onorare un contratto di sfruttamento delle risorse una volta che questo venga stipulato. Un sistema adattabile è invece un sistema che non rimane statico, ma reagisce alla situazione nella quale si viene a trovare. Ricorre perciò ad approcci pro-attivi affiancati da quelli reattivi, dal momento che la disponibilità di risorse di un sistema distribuito può essere soggetta, in taluni casi, a grandi fluttuazioni. In questa situazione, il supporto effettua dei controlli per capire come si stia evolvendo la situazione della trasmissione monitorando il sistema, con azioni svolte dinamicamente. Se questi controlli registrano un sensibile degrado delle risorse disponibili, il supporto richiede una nuova fase di negoziazione dinamica della QoS. In pratica si dovrebbe strutturare il sistema in modo che, dopo aver rilevato un degrado nella situazione delle risorse, sia proprio il supporto a occuparsi della rinegoziazione delle risorse stesse con l'applicazione e di adattare il servizio alla nuova situazione che si è venuta a verificare. Ovviamente il risultato è quello di affrontare il problema della gestione delle risorse con un metodo sia pro-attivo che reattivo, come specificato anche in precedenza.

1.5.7 Visibilità

La visibilità stabilisce che, nell'ambito delle applicazioni multimediali, il supporto deve poter offrire mezzi per raccogliere tutte le informazioni utili per comprendere quale sia l'uso corrente delle risorse e per gestirle. Risulta ovvio come sia fondamentale questo principio, se si vogliono realizzare moduli che effettuino il monitoraggio e che quindi devono necessariamente avere accesso alle informazioni riguardanti le risorse che si vogliono monitorare, in modo da poter stabilire se vi siano o meno situazioni di criticità da gestire. Questo principio, nonostante la sua semplicità, apre però un gran numero di possibilità per lo sviluppatore del supporto. Nella pratica il problema diventa quello di fissare i livelli di trasparenza e di espressività che si vuole offrire a chi poi dovrà sviluppare l'applicativo.

1.6 Conclusione

In questo capitolo abbiamo presentato i concetti fondamentali associati al Network Management, in modo da fornire il lettore degli strumenti necessari per capire il lavoro che verrà poi svolto in seguito. Bisogna sottolineare come questi argomenti siano strettamente legati con le nostre finalità, ma anche che in fondo a noi interessa solo qualche aspetto di un più vasto problema legato alla gestione delle reti, ossia gestire volta per volta, effettuando dei monitoraggi, il traffico di rete presente su di una determinata macchina. In seguito si sono introdotti alcuni concetti legati alla fruizione dei servizi multimediali ed alle problematiche legate alla gestione delle risorse necessarie a tale fruizione. Infine si è redatto un elenco di quelli che dovrebbero essere i principi che stanno alla base della realizzazione di un sistema per la gestione dei flussi multimediali e che anche in questo lavoro di tesi devono seguirsi, ponendo in evidenza quei principi strettamente legati al monitoraggio ed al controllo. Nel prossimo capitolo si andranno a presentare i due protocolli (RTP ed SNMP) tramite i quali si sono svolti tutti i controlli che si intendevano realizzare, delineandone le principali caratteristiche.

Capitolo 2

2 I protocolli: RTP ed SNMP

In questo capitolo verranno presentati quelli che sono i due principali protocolli di comunicazione e controllo utilizzati nello svolgimento di questo lavoro di tesi. Si vuole porre in particolare rilevanza le funzioni messe a disposizione da questi due protocolli che permettono di effettuare il monitoraggio della situazione della linea in termini di occupazione della banda.

2.1 Il protocollo RTP

Molto spesso nelle applicazioni distribuite è importante che i dati arrivino integri a destinazione e per assicurare tale integrità sovente si possono tollerare dei ritardi nella loro consegna. Nello sviluppo di applicazioni che devono fornire un servizio in tempo reale invece questi ritardi risultano per lo più deleteri e perciò si può anche accettare di perdere una parte dei dati trasmessi se questo può servire a contenere questi ritardi. Ne consegue che protocolli che sono progettati per garantire l'arrivo dei pacchetti mal si adattano al supporto di applicazioni multimediali. In questo contesto si introduce il protocollo *Real-time Transport Protocol* (RTP) [RFC 1889] che rappresenta lo standard proposto dall' *Internet Engeneering Task Force* (IETF) per la distribuzione di flussi multimediali su internet.

Il protocollo RTP si può definire, secondo lo stack OSI, come un protocollo di livello applicativo, il quale provvede alla distribuzione di servizi punto-punto per applicazioni che necessitano una trasmissione dei dati in tempo reale come ad esempio stream audio o video, ma non solo dal momento che sono state definite estensioni anche per altri tipi di dati. Questi servizi includono l'identificazione del formato dei dati trasportato (*payload type*), la numerazione dei pacchetti (*sequence numbering*), l'assegnazione di un *timestamp*, ed il monitoraggio della sessione. Tipicamente le applicazioni che utilizzano RTP lo pongono sopra al protocollo *User Datagram Protocol* (UDP), che fornisce le operazioni di *multiplexing* e *checksum*, anche se, per le sue caratteristiche, RTP può essere usato con altri protocolli di rete e di trasporto. RTP supporta il trasferimento dati verso molteplici destinazioni usando, una distribuzione di tipo multicast. Inoltre è da notare che questo protocollo non

prevede nessun meccanismo che assicuri una corretta trasmissione o che garantisca la qualità del servizio, in quanto questo compito è demandato proprio ai livelli sottostanti; esso però facilita la gestione del QoS dotando il livello applicativo di maggiore visibilità. RTP non si interessa, inoltre, del mancato arrivo a destinazione in ordine dei pacchetti né dell'affidabilità con cui i livelli di rete prevedono il riordino. La numerazione, permette al ricevente di ricostruire la corretta sequenza dei pacchetti inviati dal mittente; inoltre i *sequence number* possono essere usati per determinare la corretta posizione di un pacchetto all'interno di una sequenza senza necessariamente decodificarlo. Sebbene RTP sia stato implementato principalmente per le videoconferenze, esso viene comunemente impiegato anche nella memorizzazione di flussi continui, nelle simulazioni interattive distribuite, e nelle applicazioni di misurazione e controllo.

RTP consiste di due parti principali (fig. 2.1):

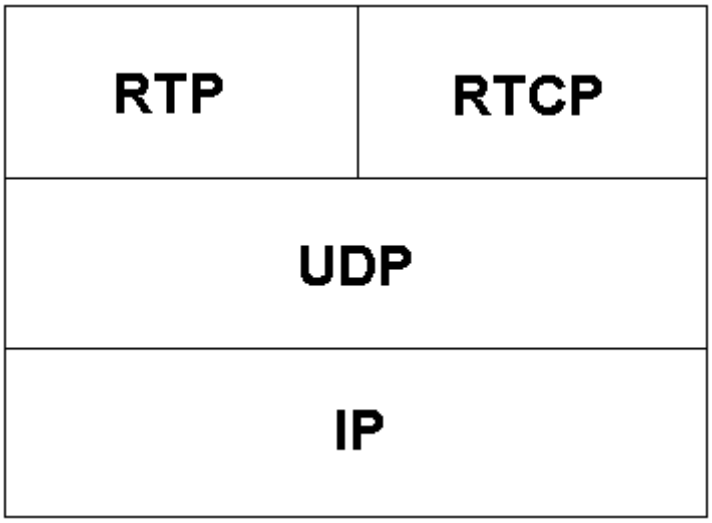

Figura 2.1: Lo stack protocollare di RTP

- il Real-Time Protocol (RTP), per trasportare dati che hanno vincoli di real-time
- l'Real-Time Control Protocol (RTCP), per monitorare la qualità del servizio e fornire informazioni sui partecipanti di una sessione in atto. Questo ultimo aspetto di RTCP può essere sufficiente per applicazioni dove non esiste un esplicito controllo dei partecipanti, ma non è sufficiente per implementare meccanismi di management dei gruppi.

RTP rappresenta un nuovo protocollo di livello applicativo e vuole essere malleabile per fornire all'applicazione le particolari informazioni di cui essa ha

bisogno. Esso sarà quindi integrato nell'applicazione, invece di essere implementato come uno strato separato. (vedi fig. 2.2)

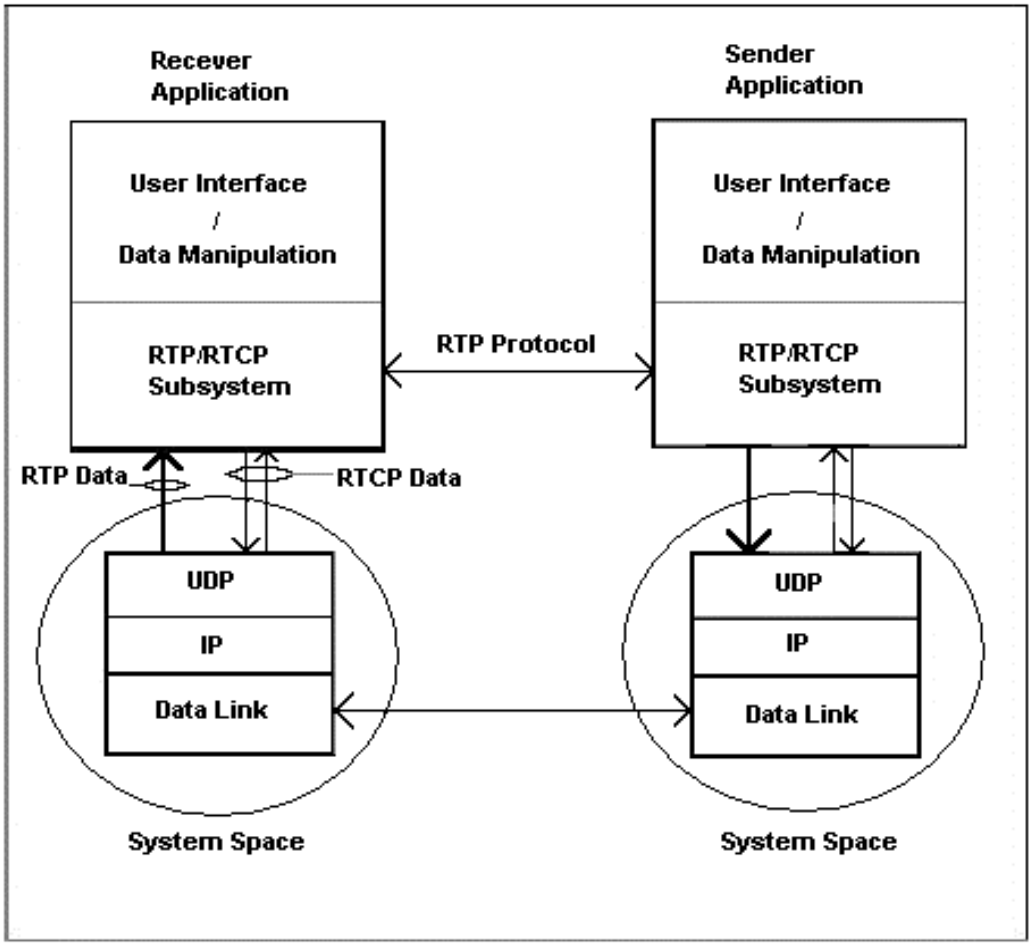

Figura 2.2: Struttura tipica di una applicazione RTP

Da notare inoltre che l'RFC1889 non specifica completamente il protocollo RTP, infatti esso è stato progettato per essere "aggiustato su misura" ogni volta che lo si implementa. Per questi motivi una completa specifica di RTP richiede uno o più documenti d'accompagnamento:

1. documento di specifica del **profile**: definisce i payload type possibili, estensioni e campi particolari dell'header fisso. Io in particolare farò riferimento all'RFC 1890 che definisce il profile per conferenze audio/video e che è il profile usato in H323.
2. documenti di specifica del **payload format**: specifica come un particolare payload deve essere trasportato nel pacchetto RTP (ricordo che il payload la parte di dati audio o video da inserire nei pacchetti). Da notare che questo documento non è necessario per tutti i tipi di payload, infatti, quelli più semplici (G.711, G722, ...) sono già contemplati nel profile.

2.1.1 RTP: transport protocol

Come accennato in precedenza, il protocollo RTP è nato inizialmente per fornire un supporto valido alle applicazioni in real-time ed in particolare per le conferenze audio/video. In questa situazione entrambi i media sono trasmessi come sessioni RTP separate ed i pacchetti relativi RTCP utilizzano due differenti coppie di porte UDP. Le sessioni RTP audio e video, non sono, perciò, accoppiate direttamente tra loro. Il motivo di questa separazione sta nel voler lasciare ai partecipanti della conferenza più libertà permettendo loro di ricevere soltanto uno dei due media a loro scelta. Nonostante questo è però possibile sincronizzare il playback della sorgente audio/video, usando informazioni di temporizzazione, trasportate nei pacchetti RTCP, per entrambe le sessioni.

Ogni sessione RTP è perciò caratterizzata da una coppia di porte UDP, una per il trasferimento dei dati, ossia la sessione RTP vera e propria, ed una dedicata esclusivamente al traffico di controllo e quindi ai pacchetti RTCP che vengono spediti periodicamente. Ogni pacchetto, oltre ad informazioni riguardanti l'utente, è caratterizzato da informazioni sul timing che ne permettono la giusta collocazione temporale.

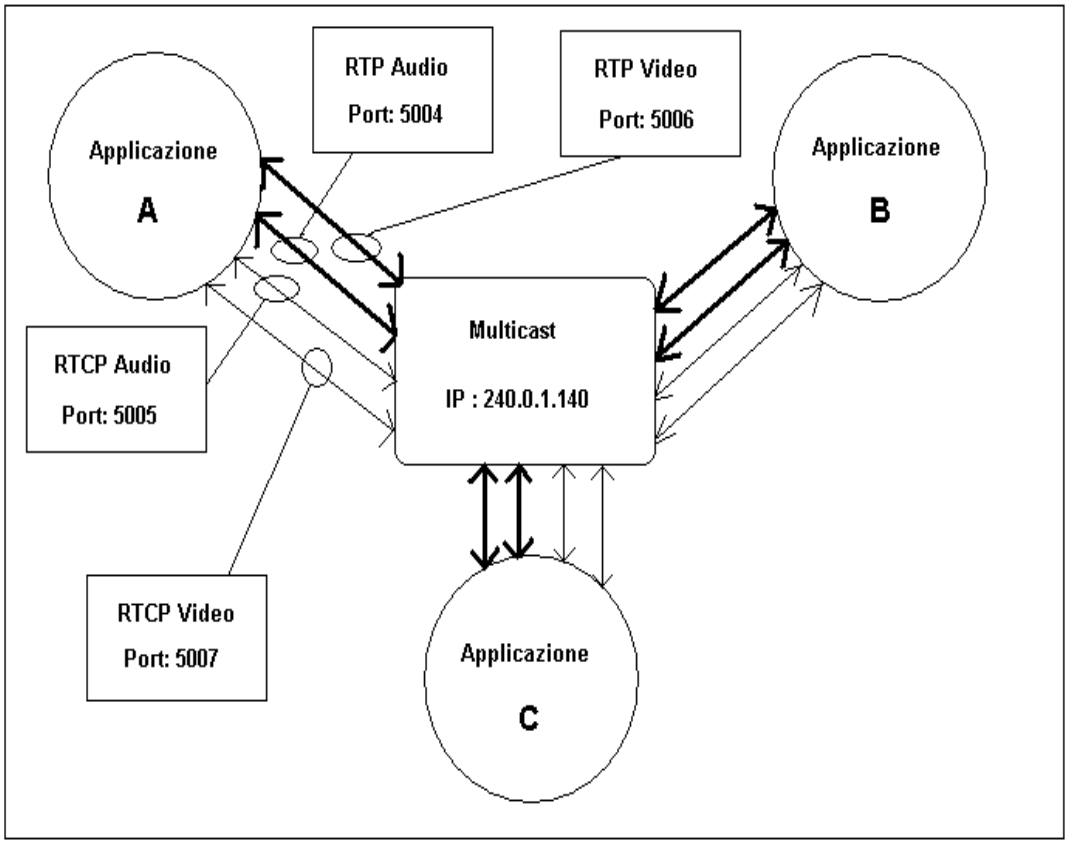

Figura 2.3: Struttura di una conferenza audio/video multicast

Ad esempio, in una conferenza multicast audio/video, come quella di figura 2.3, le sessioni rimangono separate ed utilizzano quindi due coppie distinte di porte UDP. In questo caso l'unico valore che consente il giusto accoppiamento tra dati video e relativi dati audio, è il CNAME (Canonical NAME) che è contenuto nei pacchetti RTCP ed identifica univocamente ogni partecipante.

Il tipico pacchetto dati RTP (fig. 2.4) è diviso generalmente in quattro parti. La prima di esse consiste in un header RTP fisso, dove sono incapsulate informazioni di carattere generale.

Le più significative sono:

- la **versione** di RTP che si sta utilizzando (2 bits);
- il **payload type**, ossia il formato del payload che si utilizza in modo da poterlo in seguito interpretare correttamente (7 bits);
- il **sequence number**, utile per ricostruire la giusta sequenza dei pacchetti (16 bits);
- il **timestamp**, che identifica l'istante nel quale è generato il pacchetto (32 bits);

- **synchronization source (SSRC)**, ossia l' identificatore di una sorgente di uno stream di pacchetti RTP, trasportato nell'header RTP in modo da non dipendere dall'indirizzo di rete (32 bits).

Questa parte fissa di header è seguita da una coppia di campi opzionali.
Il primo esiste solo in presenza di un flusso proveniente da più sorgenti ed è detto **contributing source (CSRC)**. Questi altri non è che una lista, detta appunto CSRC list, dei SSRC delle diverse sorgenti che hanno contribuito a formare uno stream. Il secondo campo opzionale è chiamato **extension** e contiene eventuali aggiunte all'header. Infine c'è il campo di **payload o payload data**, ossia la sorgente dati, vale a dire i campioni audio o video compressi.

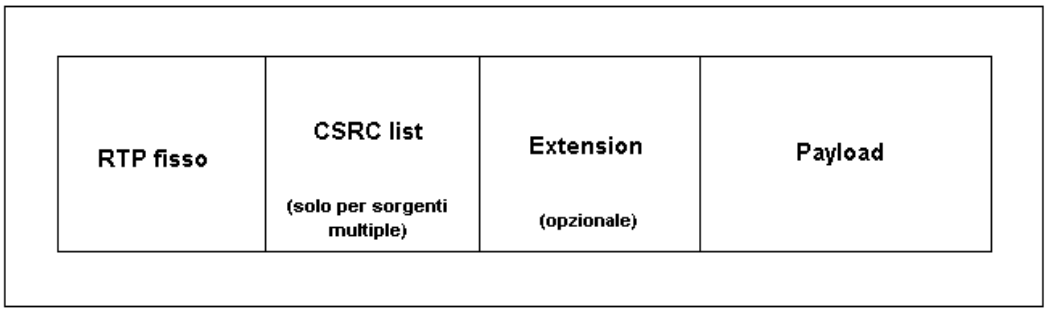

Figura 2.4: Struttura di un pacchetto dati RTP

2.1.2 RTCP: Control Protocol

L'RTP control protocol è basato sulla trasmissione periodica dei pacchetti di controllo a tutti i partecipanti ad una sessione. Il protocollo sottostante deve provvedere alla separazione dei pacchetti di dati e di controllo, usando porte con numeri diversi.
L'RTCP esegue quattro funzioni (fig. 2.5):

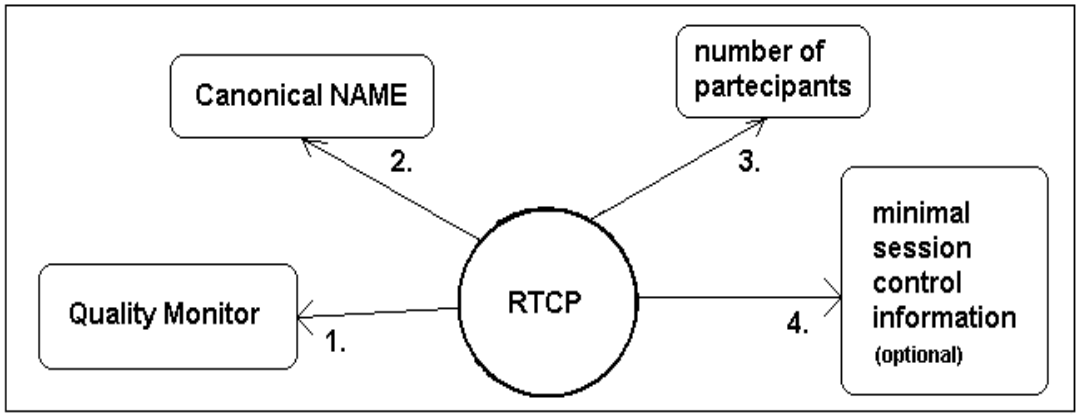

Figura 2.5: Servizi forniti da RTCP

1. La funzione principale è provvedere al **controllo della qualità** della trasmissione dati. Questa funzione è permessa dai reports RTCP. E' anche possibile, per una entità non direttamente coinvolta nella sessione, ricevere informazioni di feedback e di agire quindi da “monitor" di rete diagnosticando eventuali problemi di congestione.

2. RTCP trasporta un identifier di sorgente RTP chiamato canonical name **(CNAME)**. Poiché l'SSRC identifier può cambiare in caso di problemi di conflitto o di restart di programmi, i riceventi richiedono il CNAME per ricondursi al partecipante. Essi richiedono il CNAME anche per riorganizzare sessioni multiple trasmesse da uno stesso mittente (per es.: video e audio).

3. Le prime due funzioni richiedono che tutti i partecipanti spediscano pacchetti RTCP. Per questo la frequenza alla quale vengono spediti i pacchetti, deve essere controllata in funzione del numero di partecipanti. Spedire pacchetti RTCP "tutti a tutti", permette di risalire al **numero di partecipanti**, quindi di permettere il calcolo di tale frequenza (vedi RTCP Transmission Interval).

4. Una quarta funzione opzionale regola un controllo minimo sulle sessioni in atto, dove i partecipanti entrano ed escono senza parametri di negoziazione. Ciò serve a realizzare una **conferenza con libero accesso.**

Le prime tre funzioni sono obbligatorie, mentre l’ultima è opzionale.

Il protocollo RTCP si basa sulla trasmissione di cinque **tipi di pacchetto** (Fig. 2.6):

1. **SR (sender report)**: per portare statistiche di ricezione e di trasmissione effettuate dai partecipanti trasmettono dati RTP.
2. **RR (receiver report)**: per le statistiche di ricezione di un partecipante che riceve solo dati RTP.
3. **SDES (source descriptor)**: per gli elementi di descrizione , incluso CNAME .
4. **BYE** : indica la fine di una partecipazione.
5. **APP** : pacchetto per funzioni specifiche di una applicazione.

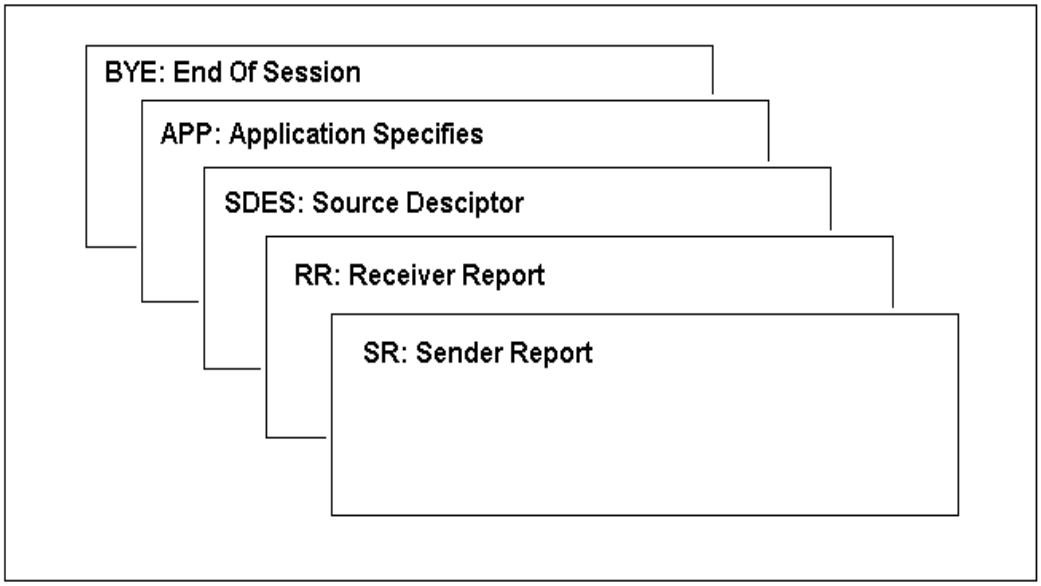

Figura 2.6: Tipi di pacchetto RTCP

Ogni pacchetto è costituito da una prima parte fissa, e da una seconda parte variabile. I pacchetti che più sono interessanti dal punto del monitoraggio sono sicuramente i pacchetti di *Sender Report* e *Receiver Report.* Questi infatti, dopo la parte di header comune a tutti, presentano dei blocchi di dati detti *Report Block*. In questi spazi vengono inserite quelle che sono le informazioni che servono per andare a comprendere quale sia la reale situazione del traffico sulla rete. Il primo campo di questi report block occupa 32 bits ed è quello che si riferisce al **SSRC** del sender (o del receiver) al quale si riferisce il report block. Questo campo è poi seguito da 8 bits che stanno ad indicare il **fraction lost**, ossia il numero dei pacchetti RTP persi dal precedente report spedito. Il terzo campo di questi blocchi è detto **cumulative**

number of packet lost. Questi occupa 24 bits, rappresenta il numero di pacchetti RTP persi tra quelli spediti dalla sorgente definita da SSRC e viene calcolato a partire dal numero di sequenza dei pacchetti RTP. Vi sono poi 32 bits che rappresentano il numero di sequenza dell'ultimo pacchetto RTP ricevuto e sono detti **extended highest sequence number received**. In realtà il numero viene esteso da 16 a 32 bits per evitare ripetizioni cicliche. Un valore molto interessante è l'**interarrival jitter** e cioè 32 bits che danno una stima della varianza statistica dell'intervallo di tempo d'arrivo (latenza) dei pacchetti RTP. Il seguente campo è il **last SR timestamp (LSR)**; occupa 32 bits e rappresenta la parte centrale del timestamp NTP del più recente pacchetto RTCP ricevuto dalla sorgente SSRC. Infine l'ultimo campo dei report block è una stima del ritardo fra l'ultimo pacchetto RTCP ricevuto da SSRC e la spedizione di questo report, espresso in una frazione di secondo pari a 1/65536. Questo campo, il **delay since last SR (DLSR)**, utilizza 32 bits. I due tipi di pacchetti *Sender Report* e *Receiver Report* si differenziano quindi solamente per la presenza di un blocco, nel primo di essi, detto **Sender Info**, che mantiene informazioni di timestamp (in due diversi formati) ed informazioni riguardanti i numeri di pacchetti RTP ed il numero di bytes (solo quelli di payload) spediti dalla sorgente SSRC spediti dall'inizio della sessione.

Più pacchetti RTCP possono essere inviati in uno stesso pacchetto UDP. Ogni pacchetto deve essere però processato dalla applicazione in modo indipendente l'uno dall'altro. Esistono dei **vincoli** che vengono imposti per la trasmissione di questi pacchetti. Innanzitutto i pacchetti SR e RR devono essere spediti tutte le volte che i vincoli di banda lo consentono , cioè devono sempre esserci in un pacchetto composto. Inoltre i nuovi ricevitori devono ottenere il CNAME delle sorgenti il più presto possibile per associare i diversi stream RTP. Infine il numero di pacchetti in un pacchetto composto, inizialmente deve essere limitato per aumentare la probabilità di un successo nella validazione dei pacchetti stessi.

Anche ogni pacchetto composto, di almeno due pacchetti RTCP, dovrà seguire determinate regole seguendo un **formato** standard, come indicato in figura 2.7. Come primo campo troviamo un **prefisso di criptazione** che solo se il pacchetto è criptato è preceduto da una quantità random di 32-bit ricalcolata per ogni pacchetto (vedi RFC1321 e RFC2437 per chiarimenti sulla criptazione). Questi è seguito sempre da un **Sender Report (SR)** oppure da un **Receiver Report (RR)**, in modo da facilitare la validazione.

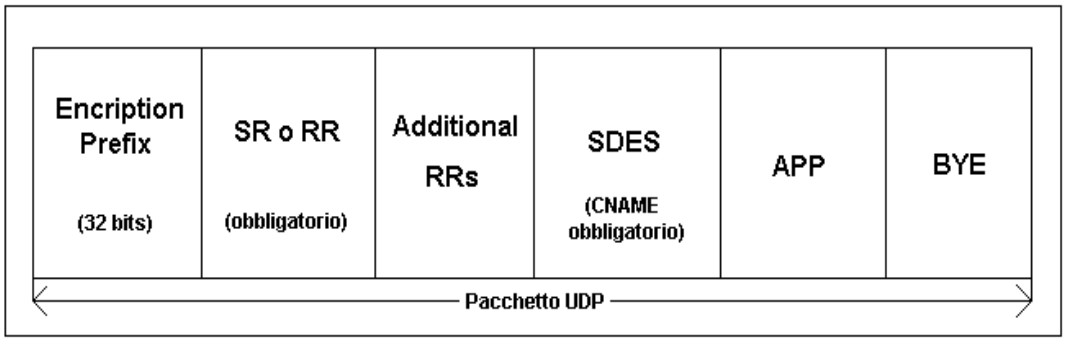

Figura 2.7: Pacchetto RTCP composto

Se le sorgenti di cui si sono fatte le statistiche superano il numero 31 dopo i pacchetti **SR** (o **RR**) ci sono campi detti **RRs addizionali**, che ovviamente mantengono le statistiche sulle sorgenti addizionali. Vi è poi il campo **SDES**, ossia un pacchetto SDES contenente CNAME deve essere presente in ogni pacchetto composto. Altri SDES opzionali possono essere inclusi. Infine vi sono i campi **BYE o APP** ed in particolare il pacchetto BYE deve essere l'ultimo.

2.1.3 La banda impiegata.

RTP è un protocollo progettato per sessioni che vanno da pochi a migliaia di partecipanti, quindi non solo per servizi di videoconferenza ma anche per servizi di video on demand, o trasmissioni "TV" su Internet. Non stupisce quindi l'attenzione che viene data al controllo della banda impiegata. Prima di tutto va notato che la banda impiegata dalle trasmissioni audio sono auto limitate perché in generale parla un solo interlocutore alla volta. Per quanto riguarda il traffico video, esso cresce con il numero dei partecipanti, ma si suppone che in una conferenza il numero dei partecipanti sia limitato mentre in una trasmissione "TV", la sorgente video è una sola. Si conclude quindi che anche il traffico video è auto limitato. Ciò non può dirsi vero per il traffico dei pacchetti RTCP. Si pensi ad esempio proprio ad una trasmissione "TV", in cui ogni ricevente manda il suo pacchetto RTCP. L'intervallo di trasmissione fra un pacchetto RTCP e l'altro va quindi regolato in base al numero di partecipanti. Per ogni sessione si suppone quindi che il traffico sia sottoposto ad un **limite di banda**, come mostrato in figura 2.8, da dividere fra i partecipanti.

Tale limite di banda può essere scelto secondo qualche criterio, come ad esempio il costo della banda o una percentuale della disponibile della stessa. Fatto sta che l'applicazione può obbligare l'impiego di banda sotto un certo limite. Tale controllo si esercita usando tipi di codifica più o meno compressi. La banda è calcolata con gli header introdotti da UDP (8 byte) e da IP (20 byte). L'header del link-level non è incluso in quanto, nel suo percorso, il pacchetto

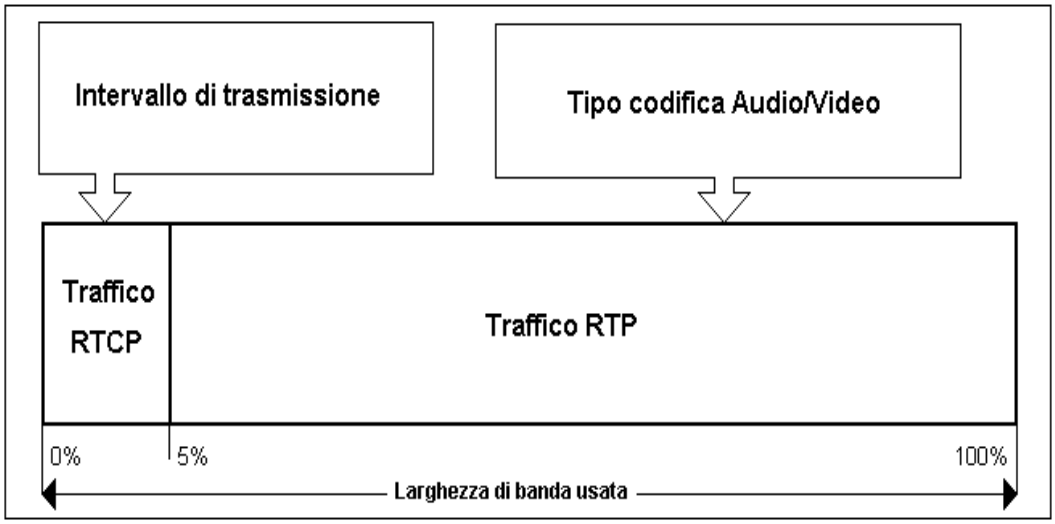

Figura 2.8: Limite di banda e 'leve' di controllo

può essere incapsulato in diversi link-level headers, dal momento che il pacchetto deve poi viaggiare nella rete internet. Il traffico di controllo deve essere limitato ad una piccola frazione della banda impiegata. Viene suggerito nel RFC proprio del protocollo, cioè l'rfc1889, di usare il 5% del limite di banda previsto per la sessione, anche se la frazione può in ogni caso essere specificata nel profile. Ciò viene fatto calcolando opportunamente l'intervallo di trasmissione fra un pacchetto RTCP e l'altro; per questo calcolo occorre tenere conto del numero di partecipanti. Nei pacchetti RTCP oltre ai report e al CNAME possono essere contenute informazioni addizionali, viene quindi suggerito che tali informazioni addizionali non superino il 20% del traffico di controllo per far sì che il CNAME sia spedito con sufficiente frequenza.

2.2 Il protocollo SNMP

Il protocollo SNMP (Simple Network Management Protocol) [Snmp] nasce nel 1988 con l'intento di fornire gli amministratori delle reti di un metodo unico che consenta di accedere alle informazioni di configurazione e di performance in modo diretto. Esistevano già infatti diversi tools che potevano essere utilizzati per la gestione della rete, ma sfortunatamente tali approcci non consentivano di gestire in maniera semplice tutti i possibili dispositivi che possono essere installati all'interno di una rete. Questo protocollo è quindi diventato in fretta **standard de facto** per la gestione delle reti e poiché costituisce una semplice soluzione al problema del network management, molti fabbricanti di hardware lo implementano nei propri prodotti. Un principio fondamentale del protocollo SNMP è quello dell'accessibilità, ossia ogni oggetto gestito attraverso tale protocollo deve poter essere acceduto. Questa accessibilità comporta che le informazioni di gestione vengano memorizzate da qualche parte, e che si possano interrogare e modificare. La struttura delle informazioni di gestione, descritta nell'*RFC 1155*, organizza le informazione ed assegna ad esse dei nomi in modo tale che si possa realizzare un accesso corretto a tali informazioni. Ogni elemento SNMP gestisce oggetti specifici a cui sono associate caratteristiche specifiche. Ciascuna coppia oggetto/caratteristica ha un proprio Object Identifier (OId), consistente di numeri separati da punti. Un Management Information Base consente di accedere in maniera efficiente alle informazioni che si desidera gestire. SNMP si fonda sul **modello Manager/Agent**. Ci si riferisce al protocollo SNMP con il termine "semplice" in quanto il software da installare sull'Agent (nodo della rete da gestire) è minimo. La maggior parte delle capacità di elaborazione risiede, infatti, sulla Network Management Station (Manager). In figura 2.9 è evidenziata la tipica architettura sulla quale agisce SNMP.

Per capire come funziona il Simple Network Management Protocol è opportuno introdurre alcune entità sulle quali si basa il protocollo stesso.

Vi sono innanzitutto i *Network Elements*, che altro non sono che i dispositivi hardware che vengono connessi alla rete. Su di essi risiedono gli *Agents*, moduli software che hanno il compito di memorizzare le informazioni. Queste informazioni vengono memorizzate nel *MIB* o *Management Information Base* che consiste in un insieme di managed objects che risiedono all'interno di un database

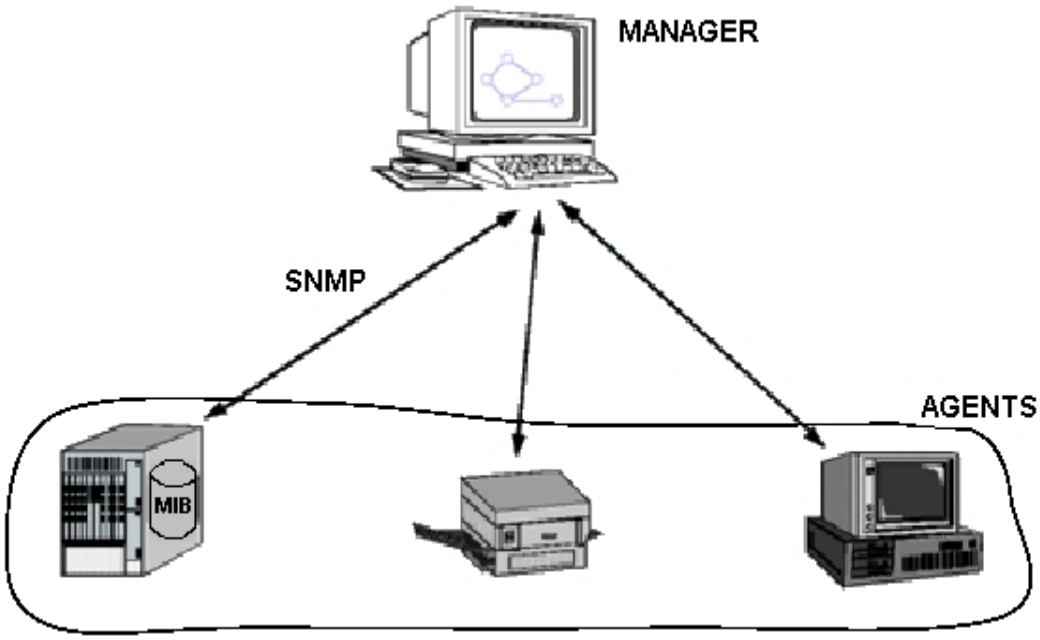

Figura 2.9: **Tipica architettura SNMP**

(il MIB appunto). Ognuno di questi *Managed Object* è una caratteristica che deve essere gestita. Vi è poi il concetto di *Management Protocol*, ossia il protocollo con il quale gli agenti parlano con il manager è che nel nostro caso è proprio SNMP. Infine ci sono le definizioni di sintassi ed SMI. La *sintassi* è il linguaggio usato per descrivere i managed object contenuti in un MIB mediante un formato indipendente dal computer. Si usa un sottoinsieme dello standard ISO ASN.1 (Abstract Syntax Notation) sia per definire il formato dei pacchetti scambiati dal protocollo di gestione che gli oggetti che debbono essere gestiti. Anche il *Structure of management information (SMI)* è definito tramite ASN.1 e serve a definire le regole per descrivere le informazioni di gestione. In particolare questa struttura impone che ogni oggetto gestito abbia un nome, che corrisponde al suo Object Identifier (OId), una sintassi, che ne definisce il tipo (es. intero o reale), ed una codifica, che descrive come le informazioni associate al managed object sono formattate per poter essere trasmesse in rete. Esistono tre successive versioni di SMI tra cui l'ultima, detta SMIng, dove si è cercato di colmare le lacune delle precedenti versioni e si è posto l'obiettivo principale di individuare un linguaggio di definizione dei dati (Data Definition Language o DDL) che sia realmente indipendente dai protocolli. Un MIB (Management Information Base) può essere descritto come un albero n-ario la cui radice non ha nome. Le foglie dell'albero sono costituite da entità individuali. Gli

Object Identifier (OId) consentono di individuare univocamente una entità all'interno dell'albero. Gli OId sono assimilabili a dei numeri telefonici strutturati in maniera gerarchica; ogni organizzazione possiede una propria sequenza di numeri. L'attuale MIB standard per internet è il MIB-II (fig. 2.10) che è stato definito nel rfc1213. Questo albero contiene 171 oggetti che sono raggruppati per protocollo (es. IP, TCP, ecc) o per categoria (es. System o Interfaces).

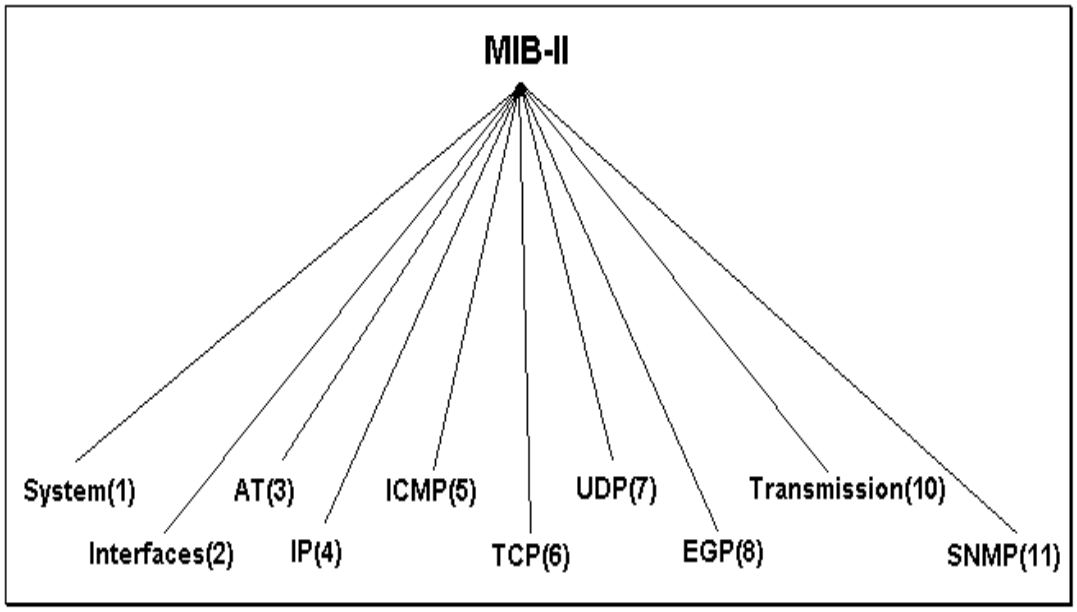

Figura 2.10: Struttura del MIB-II

2.2.1 Versioni del protocollo SNMP

Il protocollo SNMP assume che i canali di comunicazione siano connection less, quindi si utilizza come protocollo di livello di trasporto il protocollo UDP (fig. 2.11). Di conseguenza SNMP non garantisce l'affidabilità dei propri pacchetti. Inoltre i moduli Agent sono sempre in ascolto sulla porta UDP 161, mentre le risposte sono inviate alla Network Management Station (Manager) utilizzando un numero di porta casuale. La scelta di usare SNMP sopra il protocollo di trasporto UDP implica anche un vincolo sulla dimensione massima dei pacchetti. In particolare questa è limitata superiormente dalla massima dimensione del payload UDP (65507 byte). Tutti i messaggi di errore e le eccezioni (Trap) sono spediti dall'Agent al Manager in maniera asincrona utilizzando la porta UDP 162.

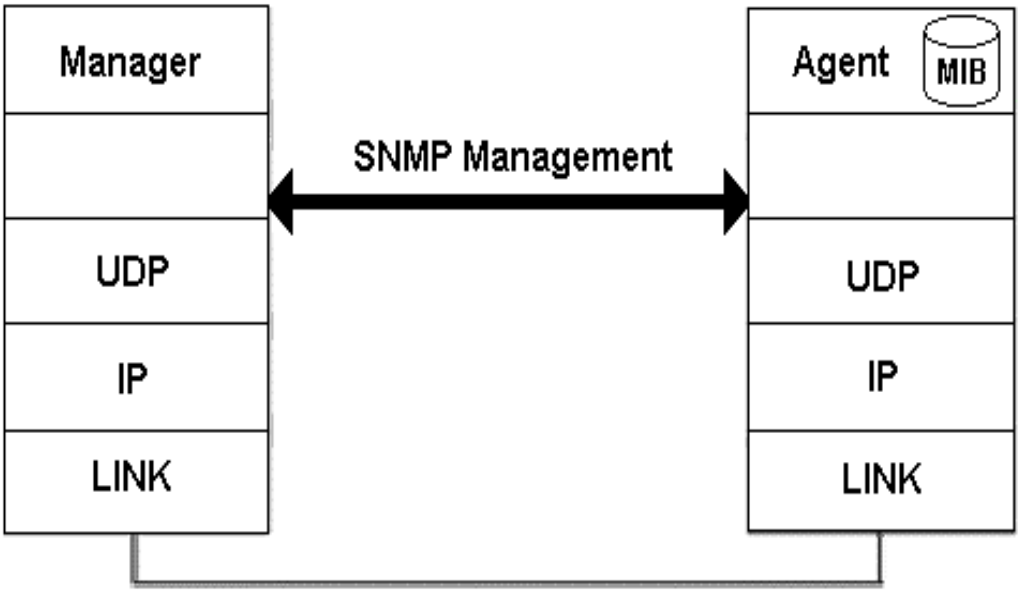

Figura 2.11: Stack protocollare di SNMP

La prima versione del protocollo aveva soltanto quattro primitive: GET, GET-NEXT, SET, TRAP. La primitiva di GET viene utilizzata dal Manager per reperire un valore dal MIB dell'Agent. Questa operazione può anche essere effettuata ricorsivamente dal Manager tramite l'operazione di GET-NEXT. La SET invece serve al Manager per andare a scrivere sul MIB, impostando i valori scelti. L'unica primitiva richiamata dall'Agent è la TRAP ed è usata per inviare al Manager messaggi di errore. Il formato del pacchetto SNMP che consente la gestione di tali operazioni comprende tre campi: un numero di versione; una community, cioè una stringa utilizzata come password; uno o più payload SNMP (fig 2.12). Ovviamente questa versione presentava consistenti limitazioni, come la presenza di regole non documentate, codice di errori e tipi di dato limitati. Inoltre una applicazione di questo tipo presentava poca sicurezza e scarse prestazioni, dipendeva fortemente dal protocollo di trasporto ed era completamente sprovvista di una gerarchia nell'architettura Manager/Agent.

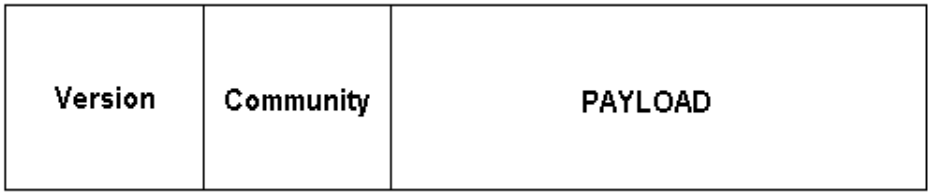

Figura 2.12: Formato dei pacchetti SNMPv1

Si è quindi optato per una scelta che andasse a colmare le lacune che erano presenti nelle prime versioni del protocollo soprattutto nell'ambito della sicurezza delle trasmissioni. In questo modo si è arrivati nel 1999 alla release della terza versione, attualmente la più aggiornata, che è pensata per essere scalabile, duratura, portabile e compatibile con le versioni precedenti (usa infatti gli stessi MIB). Purtroppo però la versione più utilizzata del protocollo resta la prima dal momento che la maggiorazione del numero delle caratteristiche è andato a scapito della sua semplicità. In questa terza versione si è pensato di cambiare anche la classica architettura di tipo Manager/Agent, sostituendola con una più complessa composta da due oggetti detti Motore ed Applicazioni. Infatti una entità SNMP di terza generazione è composta da un SNMP Engine (Motore) e da un SNMP Applications (Applicazione). L'Engine è formato da un Dispatcher (smistatore di messaggi), un sottosistema per elaborare i messaggi, uno per la sicurezza ed uno per il controllo dell'accesso. L'Applications invece contiene un generatore di comandi, un ricettore di notifiche ed un risponditore ai comandi. Oltre a queste modifiche dell'architettura si è anche operato sul formato dei pacchetti SNMP in modo da includere al loro interno anche alcuni parametri di sicurezza ed il controllo dell'accesso. In particolare sono presenti in questa versione appropriate politiche di sicurezza, implementate tramite crittografia, funzioni di hash ed altri strumenti che consentono l'autenticazione dei pacchetti, delle password ed anche delle PDU che a loro volta possono essere codificate. Infine tramite diversi livelli di sicurezza si può stabilire se consentire un accesso senza autenticazione (no pwd/no priv), con autenticazione (pwd/no priv) o con autenticazione e codifica dei dati (pwd/priv).

2.3 Conclusioni

I due protocolli presentati in questo capitolo forniscono gli strumenti necessari ad ottenere le informazioni utili per effettuare un monitoring esaustivo della situazione della banda in ogni momento.

In particolare RTCP permette di "fotografare" lo svolgimento della sessione fornendone valori indicativi quali ad esempio jitter e fraction lost.

SNMP permette invece di fare considerazioni più generali restituendo statistiche relative ai pacchetti in ingresso o in uscita su una determinata macchina, distinguendoli a seconda del protocollo (IP, TCP, ecc) che si desidera controllare.

Capitolo 3

3 Le tecnologie utilizzate: SOMA, MUM, JMF e SNMPApi

In questo capitolo si intendono introdurre al lettore quelle che sono le tecnologie che sono state usate nell'ambito di questo lavoro di tesi. In particolare le prime due costituiscono l'ambiente entro al quale si è andati ad operare, ossia il sistema di gestione dei servizi multimediali del quale il modulo di monitoraggio che si vuole realizzare, diverrà parte integrante. Si andranno poi a presentare le caratteristiche di due prodotti commerciali che sono stati utilizzati, andando a porre l'accento su quegli aspetti che più interessano i nostri scopi nella realizzazione del nostro modulo di monitoraggio.

3.1 S.O.M.A.

SOMA (Secure and Open Mobile Agent) [SOMA] è un ambiente ad agenti mobili realizzato presso il Dipartimento di Elettronica Informatica e Sistemistica (DEIS) dell'Università degli Studi di Bologna. Il nostro scopo, in questa sede, non è quello di fare una trattazione esauriente ed esaustiva di questo ambiente, ma sono di delinearne solo le principali caratteristiche architetturali. SOMA è infatti l'ambiente entro il quale sono state sviluppate le applicazioni di gestione dei flussi multimediali sopra le quali vogliamo andare ad operare. Riteniamo dunque utile questa breve trattazione in modo da rendere più semplice la comprensione dei punti successivi.

L'ambiente SOMA è scritto in linguaggio Java e ne sfrutta tutte le potenzialità. Questo linguaggio, infatti, risulta essere un dei più adatti quando si tratta di realizzare un'implementazione di agenti mobili per diversi motivi:

- Java è un linguaggio interpretato e per questo motivo può eseguire su qualunque macchina che possegga una Java Virtual Machine (JVM) in grado di trasformare il bytecode in istruzioni macchina; risulta perciò portabile su diverse architetture software/hardware;
- Java è un linguaggio object-oriented che permette uno sviluppo modulare del codice per estensione ed ereditarietà, così che l'utente possa scrivere i propri

agenti con un sforzo limitato a partire dalle classi messe a disposizione dalla piattaforma;

- Java prevede un meccanismo di caricamento dinamico delle classi anche da sorgenti remote e di collegamento dinamico all'applicazione corrente;
- Java fornisce la possibilità di serializzare gli oggetti, cioè di rappresentarli come stream di byte e di trasferirli in rete;
- Java ha caratteristiche di sicurezza built-in, la più nota delle quali è quella associata alle applet che, originariamente, potevano essere scaricate da un Web server, ma non avevano diritto di accedere alle risorse del sistema locale. Dalla versione 1.2 del linguaggio si ha un'architettura di sicurezza rivisitata, molto più flessibile ed espressiva, fatta di domini di protezione, controlli d'accesso e permessi da associare sia a codice remoto che a codice locale.

La scelta di un linguaggio porta inevitabilmente ad una scelta obbligata per quanto riguarda il modello di mobilità da usare. Essendo Java un linguaggio interpretato, una parte dello stato di esecuzione rimane incluso nello stato dell'interprete; la cattura di tale stato diventa praticamente impossibile se non andando ad operare modificando l'interprete, ma questo, oltre a problemi intrinseci, porterebbe alla perdita della portabilità, che è una delle caratteristiche fondamentali di un sistema ad agenti.

SOMA, per le considerazioni appena fatte, si può definire come un sistema a **mobilità debole**, dal momento che sfrutta Java e questi non supporta la mobilità dello stato di esecuzione. Secondo una classificazione propria dei sistemi ad agenti mobili, possiamo individuare in SOMA tutti i diversi elementi di questi sistemi:

- il *codice* sono le classi Java
- le *risorse* sono oggetti Java
- le *execution unit (EU)* sono thread Java
- i *siti* (in SOMA chiamati place) corrispondono a macchine virtuali Java
- le *interazioni* avvengono tramite chiamate a metodi, scambio di messaggi e, quando i componenti sono ospitati da JVM diverse, tramite scambio di comandi.

3.1.1 Caratteristiche di SOMA

SOMA presenta diverse caratteristiche interessanti nel campo del supporto alla mobilità del codice. Per prima cosa SOMA fornisce una *gerarchia di astrazioni di località*, adatta alla descrizione di ogni scenario di connessione, mutuata da Internet: esistono diversi domini all'interno dei quali vengono distinti sotto-domini, fino a giungere al semplice nodo. Il mondo in cui vivono gli agenti è formato quindi da *Place* e *Domini*, come descritto in figura 3.1.

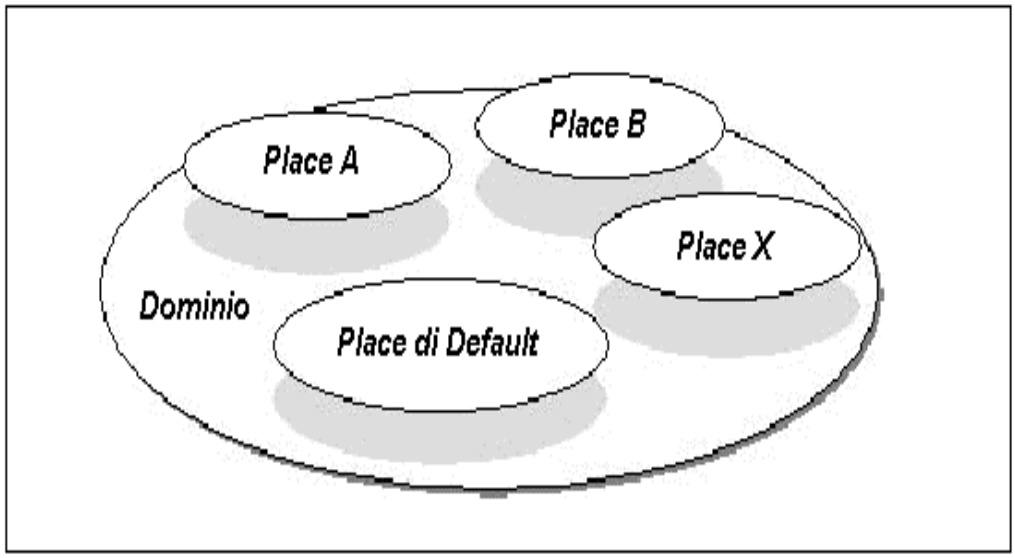

Figura 3.1: Località logiche in SOMA

Il **Place** è il contesto di esecuzione dell'agente ed estende il concetto di nodo: al place può infatti corrispondere ad una macchina fisica, ma su un nodo possono convivere anche più place, permettendo, ad esempio, la definizione di località di protezione delle risorse. Il **Dominio** è una aggregazione di place che può rispecchiare un contesto reale, come una LAN, oppure una caratteristica logica, come ad esempio può essere l'insieme dei dispositivi dello stesso tipo o appartenenti ad uno stesso dipartimento all'interno di una organizzazione. Nell'implementazione il concetto di dominio si riscontra solo nella distinzione tra place generici e place cosiddetti "di default"; questi ultimi hanno conoscenza dei siti costituenti un dominio e sono punto di accesso da e verso l'esterno.

L'organizzazione gerarchica, adottando una topologia ad albero per i domini, permette inoltre l'identificazione univoca di un certo place/default place, tramite il percorso che conduce dal default place "radice" al place/default place stesso.

Inoltre il place è *l'ambiente di esecuzione* dell'agente, realizzato mediante moduli di supporto che forniscono i servizi fondamentali. Questi moduli gestiscono sia gli agenti, permettendone l'esecuzione, l'ingresso e l'uscita da un place, che le interazioni tra i vari place, mantenendo i necessari canali di comunicazione, ed infine anche le informazioni sui place appartenenti ad un dominio.
Tutti gli agenti vengono associati ad un identificatore unico, AgentID, ricavato dalla combinazione dell'identificatore del place sul quale nasce e di un intero progressivo, in modo da conoscere sempre l'origine di un agente. Per identificare Place e Domini si utilizza un sistema di naming che si basa su un identificatore detto PlaceID. Tale identificatore è il risultato della concatenazione del nome dominio e del nome del place. Nel caso di un place di default quest'ultimo sarà semplicemente nullo.

In SOMA ogni agente possiede la capacità di comunicare con gli altri agenti, dal momento che questa è una ulteriore caratteristica di base. Questo avviene tramite scambio di messaggi che risulta essere una soluzione molto flessibile. Vi è inoltre un servizio di localizzazione degli agenti, attivabile su necessità, che permette di avere grande trasparenza.

Infine si deve sottolineare come l'ambiente SOMA garantisca il rispetto di una politica di autorizzazione così che agenti maliziosi non interagiscano in modo incontrollato con le risorse ed i servizi messi a disposizione dall'ambiente. La definizione di diverse astrazioni di località, inoltre, consente di introdurre politiche di sicurezza nelle quali le azioni siano controllate sia a livello di dominio, sia a livello di place. Il dominio definisce una politica di sicurezza globale che impone autorizzazioni e proibizioni generali; ogni place, però, può applicare restrizioni ai permessi consentiti a livello di dominio.

3.2 M.U.M.

MUM (Multimedial Ubiquitous Middleware) [MUM] è un sistema sviluppato, come SOMA, presso il Dipartimento di Elettronica Informatica e Sistemistica (DEIS) dell'Università degli Studi di Bologna. MUM rappresenta il sistema entro il quale dovremo andare a realizzare il nostro modulo di monitoraggio delle risorse. Si ritiene quindi utile andare a delinearne le principali peculiarità, ponendo l'accento su quelli che sono gli aspetti che più degli altri vengono coinvolti nel nostro lavoro. Si deve innanzitutto spiegare che questo prodotto si prefigge l'obiettivo di fornire una

piattaforma per la gestione dei flussi multimediali, che offra un valido supporto all'utilizzo delle risorse e faciliti lo sviluppo di applicazioni di nuova generazione, ispirate a modelli di ubiquitous computing. In particolare si supportano la pubblicazione e la fruizione di presentazioni multimediali di varia natura e qualità diverse su rete fissa e mobile, dove per presentazione multimediale si intende un aggregato di uno o più oggetti multimediali dotati di un attributo di qualità. Ogni oggetto multimediale è l'astrazione di un certo tipo di dato multimediale ed incapsula un solo tipo di media.

La presentazione multimediale è un metadato che viene utilizzato per aggregare diversi oggetti multimediali specificando anche la qualità della presentazione. Inoltre la presentazione contiene anche un attributo che ne identifica la posizione all'interno del sistema. Usualmente, all'aumentare della qualità aumentano anche quelle che sono le richieste di risorse per poter fruire della presentazione stessa.

Ogni presentazione è poi partizionata in sottosistemi mutuamente esclusivi, detti contenuti multimediali. Tale suddivisione è introdotta per raggruppare presentazioni multimediali che portano lo stesso contenuto informativo, ossia che hanno la stessa struttura e lo stesso titolo. Naturalmente vi sono poi delle differenziazioni a partite dai livelli di qualità, ma non sono escluse presentazioni con medesima struttura e livello di qualità. L'identificatore del contenuto multimediale all'interno dell'intero sistema è detto titolo.

Il sistema fornisce agli utilizzatori la possibilità di richiedere il delivery dei titoli presenti, comandare la presentazione e richiedere lo spostamento della sessione verso un terminale diverso da quello che stanno attualmente utilizzando. Se il sistema viene utilizzato da rete mobile, ossia da un device di tipo wireless, questo fornisce un supporto riorganizzandosi per seguire i movimenti dell'utente. Inoltre l'infrastruttura si occupa anche della gestione delle risorse supportando prenotazione e monitoraggio delle stesse, nonché della scelta della presentazione più adatta alla piattaforma computazionale dalla quale un certo utente sta accedendo al sistema. Tale specifica si realizza tramite lo studio delle caratteristiche delle diverse piattaforme, le cui caratteristiche sono salvate in dei descrittori che sono riferiti nei profili degli utenti. Infine si può aggiungere che MUM supporta il downloading e l'inizializzazione di software a runtime.

3.2.1 Fruizione del materiale multimediale in MUM

L'architettura MUM è realizzata in ambiente distribuito ed adotta, come modello computazionale, una integrazione tra il tipico modello client-server ed il modello ad agenti mobili, cercando di utilizzare il meglio di questi due approcci. In particolare in questo middleware si introducono quattro entità: *ClientAgent*, *Client*, *ProxyAgent* e *Server*. Queste entità sono tra loro collegate a formare un percorso, detto path, che va dal client verso il server, attraversando uno o più proxy. Da sottolineare che non tutte le entità sono realizzate come agenti mobili, ma alcune di esse sono realizzate anche entità fisse.

Il **ClientAgent** è l'entry point del sistema dal lato client. Questa entità è realizzata come agente mobile ed il suo compito è quello di inizializzare la sessione per l'utilizzatore e gestire poi le interazioni col resto del middleware distribuito. Per ruolo rivestito, questo agente può essere considerato come una sorta di mediatore che accetta le richieste le richieste fatte dall'utente attraverso una opportuna GUI, per interagire col resto del sistema e richiedere l'esecuzione dell'operazione richiesta. Il ClientAgent è stato pensato come un agente per facilitare l'interazione con gli altri agenti, e per permettere il movimento di queste entità, in modo da supportare gli utenti che si muovono fra i diversi terminali fissi (utenti nomatici, vedi figura 3.2), muovendosi là dove essi si vogliono muovere.

Il **Client** è l'end point che riceve i flussi multimediali richiesti. Questa entità non è realizzata come agente mobile, ma come entità fissa residente sulla macchina dalla quale l'utente effettua l'accesso al sistema. Questo non implica che il software richiesto per l'esecuzione di questo entità debba necessariamente essere presente sulla macchina dalla quale verrà poi lanciato, dal momento che è stato realizzato in modo da poter scaricare questo codice a runtime, seguendo il paradigma del code on demand (COD).

Il **Server** è per sua natura molto simile al Client ed i discorsi fatti in precedenza mantengono qui la loro validità. Infatti anche questa entità è realizzata come fissa ed è scaricabile secondo il paradigma code on demand (COD). Inoltre un certo tipo di server, una volta che sia stato lanciato su di una certa macchina, verrà utilizzato per servire tutte le richieste che da quel punto in poi arriveranno a quella macchina per quel tipo di server. Si può facilmente intuire come questa entità sia definita come l'altro end-point della comunicazione dei flussi multimediali. In pratica è il Server a spedire i flussi multimediali ad un Client che ne ha fatto richiesta.

Il **ProxyAgent** è, differenza delle ultime due entità descritte, realizzato come un agente mobili, in modo da supportare il movimento dei terminali. Per la gestione vera e propria dei flussi il ProxyAgent si avvale di un'altra entità detta **Proxy**.

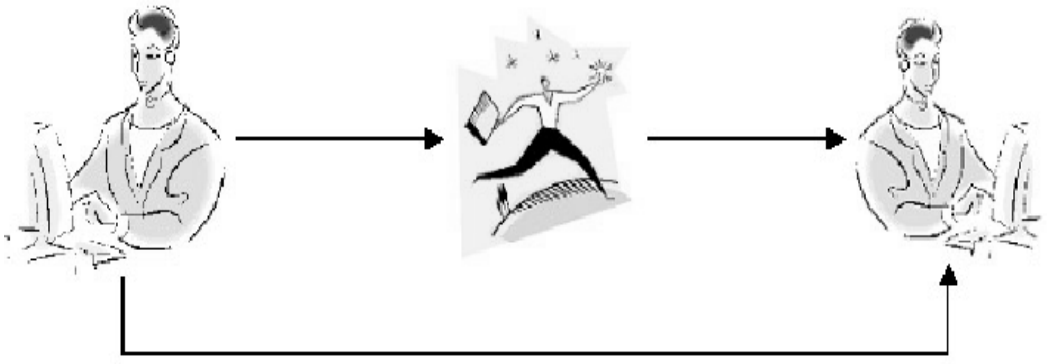

Figura 3.2: Scenario di utilizzo del sistema per utilizzatori nomatici

3.2.2 Inizializzazione e riorganizzazione dinamica in MUM

MUM offre un supporto anche per l'inizializzazione e la riorganizzazione dinamica del sistema, garantendo i principi di modularità e flessibilità. Questo è possibile dal momento che MUM è progettato per interfacce, cioè ogni oggetto interagisce con gli altri tramite interfacce ben definite, e perché da la possibilità di scaricare ed attivare le entità necessarie per lo svolgimento dello streaming. In pratica il sottosistema di inizializzazione scarica e fa partire tutte le entità che partecipano allo streaming, senza doverle conoscere, ma sapendo a quali interfacce esse rispondono. Questa azione è svolta seguendo un piano di inizializzazione da un oggetto in grado di interpretarlo e detto appunto interprete. La riorganizzazione si realizza mediante la generazione di più piani di inizializzazione alternativi, tramite i quali, scelti di volta in volta da un decisore, a runtime, il sistema potrà riorganizzarsi. Questo aspetto di MUM assume particolare rilevanza se si pensano alle implicazioni di una attività di monitoraggio. Infatti la possibilità di riconfigurare dinamicamente il sistema fa in modo che si possano attuare politiche di gestione delle risorse di tipo reattivo. In particolare può verificarsi che, a fronte di una situazione critica rilevata attraverso il monitoraggio, il sistema possa sfruttare questa sua peculiarità per ridurre la richiesta di occupazione delle risorse, con il risultato di decongestionare la situazione garantendosi in definitiva un corretto funzionamento.

3.2.3 Gestione delle risorse

Questa è la parte che più di tutte le altre di MUM interessa in questo lavoro di tesi. È infatti in questo specifico contesto che dovremo poi andare ad operare, per cercare di implementare azioni utili a prevenire, o al limite a correggere, situazioni di scarsa disponibilità di risorse. In particolare il sottosistema di gestione delle risorse in MUM attua politiche che fanno riferimento sia all'approccio pro-attivo che a quello reattivo. Per questo motivo sono presenti sia una entità con il compito di svolgere un compito di assegnazione statica delle risorse tramite **prenotazione** e negoziazione preventiva della QoS, sia una che ha il compito di effettuare il **monitoraggio** dell'andamento della disponibilità delle risorse, rinegoziando eventualmente a tempo di esecuzione la qualità di servizio. Le risorse che vengono controllate in MUM sono **CPU** e **banda trasmissiva** che vengono considerate le più sensibili ai fini della fruizione di contenuti multimediali. Il controllo viene effettuato tenendo conto di ben precisi livelli di QoS, in modo da evitare di incorrere a continui micro-adattamenti, che non portano a sostanziali giovamenti e caricano ulteriormente il sistema, e perché comunque questa classificazione a livelli è già presente in molti oggetti multimediali (come ad es. la bit-rate di un file audio), che sono quelli con cui abbiamo a che fare. Esiste un differente gestore della qualità di servizio per ogni singola macchina (fig. 3.4).

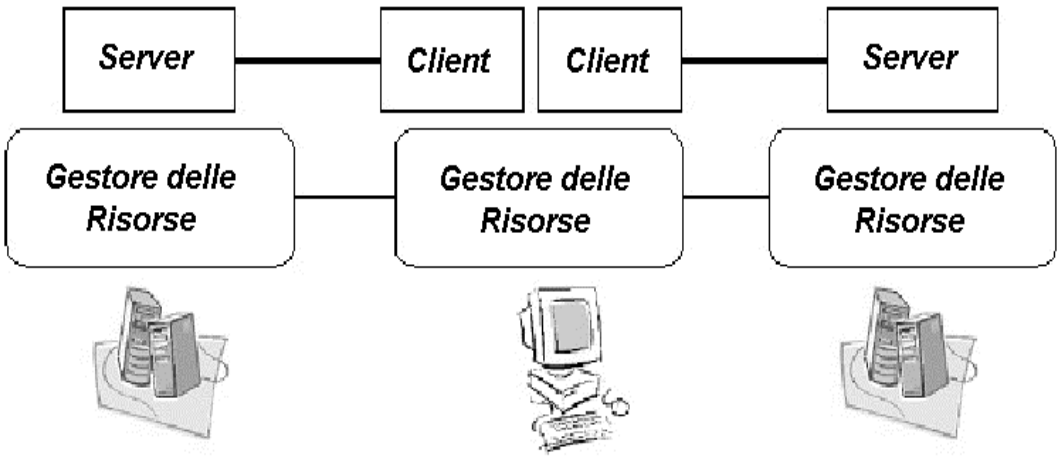

Figura 3.4: Si ha un unico gestore delle risorse per ogni macchina

Al di sopra di esso viene poi realizzata la gestione della qualità di servizio. Inoltre in MUM le richieste di CPU e banda sono incapsulate in un metadato in modo da non rendere necessaria una vera e propria fase di traslazione delle specifiche dal

livello applicativo a quello delle risorse, ma siano direttamente disponibili le richieste delle risorse. In pratica tutte le entità facenti parte della stessa macchina fanno riferimento allo stesso gestore delle risorse, il quale mantiene lo stato delle risorse della propria macchina.

3.3 JMF: Java Media Framework

Il Java Media Framework sono API (*application programming interface*) [JMFH] create per permettere di incorporare tipi di dati Multimediali in applicazioni o applet Java; si tratta di un pacchetto opzionale, installabile per estendere le funzionalità della piattaforma JAVA2SE™. Questo componente è stato sviluppato congiuntamente da Sun e IBM, ed è nato con l'intento di fornire un supporto per i più comuni standard di memorizzazione dei dati multimediali, quali: MPEG-1, MPEG-2, Quick Time, AVI, WAV, AU e MIDI.

Com'è ben noto, la peculiarità di Java è l'utilizzo di una Java Virtual Machine (JVM) che interpreta il byte-code generato dal compilatore Java. Questo meccanismo permette la portabilità del codice Java su più piattaforme ma, nel caso in cui sia richiesta un'elevata velocità di esecuzione, impone dei seri vincoli di prestazioni. Il trattamento di dati Multimediali è uno dei casi in cui è richiesta un'elevata velocità computazionale (decompressione delle immagini, rendering, ecc.), e quindi, dove non è sufficiente la sola emulazione della CPU per ottenere delle prestazioni ottimali.

Ogni sviluppatore che desideri implementare un lettore multimediale in Java, e voglia ottenere delle prestazioni eccellenti deve, necessariamente, ricorrere a codice nativo della piattaforma alla quale è interessato. Questo procedimento comporta due problemi:

- Occorre una specifica conoscenza delle funzioni native da parte del programmatore.
- Un programma Java che utilizza codice nativo non è più trasportabile su piattaforme diverse da quella originaria.

L'API JMF [JMFPG] tenta di risolvere questi problemi, mettendo a disposizione del programmatore una serie di chiamate ad "alto livello" per la gestione del codice nativo. Usando JMF, l'applicazione o applet non ha necessità di conoscere quando e se deve sfruttare particolari metodi nativi per svolgere una determinata azione. JMF

2.1.1 rende disponibili delle classi che permettono lo sviluppo di applicazioni per la cattura di dati multimediali, fornendo inoltre ai programmatori un controllo addizionale sull'elaborazione e la riproduzione dei dati stessi. JMF 2.1.1 è stato progettato per:

- Facilitare la programmazione
- Mettere a disposizione del programmatore un player JMF per la riproduzione dei dati multimediali.
- Semplificare notevolmente l'integrazione di sorgenti multimediali in applet o applicazioni fornendo tutta una serie di classi e metodi per la gestione temporale di stream di dati.
- Connettersi a host remoti e instaurare sessioni http, piuttosto che RTP/RTCP o RTSP (Real Time Streaming Protocol).
- Permettere lo sviluppo di applicazioni di audio e video conferenza in Java.
- Permettere a programmatori avanzati di implementare soluzioni personalizzate basate sulle API esistenti e di integrare le nuove caratteristiche nella struttura esistente.
- Permettere lo sviluppo di demultiplatori, codificatori, elaboratori, multiplatori e riproduttori personalizzati (JMF *plug-in*)
- Mantenere la compatibilità con JMF 1.0

3.3.1 La ricezione dei dati sul Client: il Player

Il Player è la struttura che le API di JMF mettono a disposizione del programmatore per la riproduzione di dati multimediali. Come già accennato, le funzioni messe a disposizione dal Player consentono agli sviluppatori di software di non interessarsi direttamente delle chiamate al codice nativo, di impegnare le risorse necessarie per la riproduzione e di effettuarne un eventuale rilascio, quando queste non siano più necessarie.

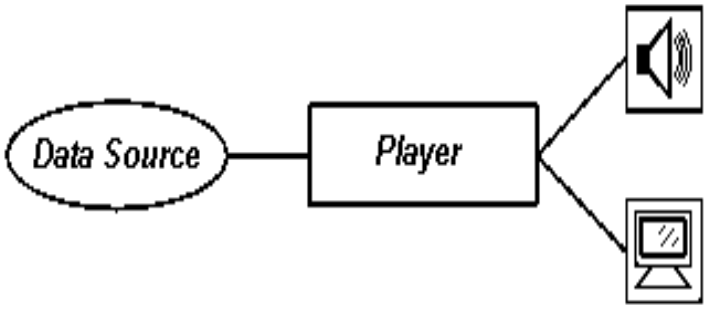

Figura 3.5: Ricezione di flussi multimediali con JMF

Le chiamate a metodi e classi di "alto livello", rendono quindi trasparente al programmatore la connessione che si stabilisce tra Java Virtual Machine e routine specifiche di sistema (proprio secondo la filosofia generale di Java). In figura 3.5 viene rappresentato un tipico scenario di utilizzo del JMF, per la ricezione di dati dalla rete. Per ogni flusso multimediale ricevuto viene impiegato un RTPManager che gestisce la sessione RTP. Ad alto livello dal lato del ricevente possiamo, attraverso l'uso di opportune API ricavare, all'arrivo dei dati, un DataSource che verrà utilizzato per l'inizializzazione del Player. Una volta inizializzato il Player potremo poi ricavare dal Player stesso un componente visuale (se presente), che utilizzeremo per la fruizione del materiale multimediale.

3.3.2 L'invio dei dati su Server e Proxy: il Processor

Per quanto riguarda invece l'invio dei dati viene utilizzato un Processor. Il Processor estende il Player ed in più di quest'ultimo offre metodi per la gestione e la trasformazione dei flussi multimediali e per l'ottenimento di un DataSource. Tale metodo è di fondamentale importanza per la successiva inizializzazione dell'RTPManager; infatti, all'atto della creazione di un SendStream, che rappresenta appunto l'astrazione del flusso dati che verrà trasmesso in rete, bisognerà passare all'RTPManager un DataSource dal quale attingere il contenuto multimediale che sarà, per l'appunto, ottenuto dal Processor. In figura 3.6 rappresentiamo la situazione che si ha per l'invio di dati in rete. Questa figura, come la precedente, mette in evidenza diversi casi di possibile utilizzo delle API.

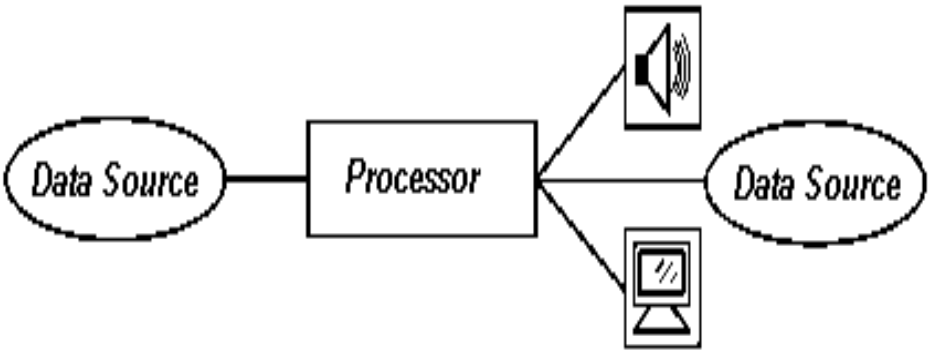

Figura 3.6: Spedizioni di flussi multimediali con JMF

3.3.3 Supporto al protocollo RTP/RTCP

L'ultima caratteristica che si vuole mettere in risalto del JMF è il fatto che offre, insieme al set di API utilizzato messo a disposizione per la riproduzione e il rendering di materiale multimediale, anche un set di API per stabilire sessioni RTP, offrendo al livello applicativo tutte le astrazioni necessarie a gestire tale protocollo in modo piuttosto semplice. Tale caratteristica fa di queste API uno strumento ideale per lo sviluppo di applicativi di distribuzione dei flussi multimediali, ed inoltre, anche per lo sviluppo di applicazioni di tipo audio/video conferenza. La realizzazione dei server e dei client è poi piuttosto semplificata dall'utilizzo di tali API, una volta capita la logica alla base del loro utilizzo. Naturalmente questo set di API risulta molto interessante per i nostri scopi perché permette di ottenere in maniera semplice utili dati che possono essere sfruttati per andarsi a studiare la situazione della corrente sessione RTP, come, per esempio, possono essere i dati di controllo RTCP che viaggiano all'interno dei pacchetti di Sender Report e Receiver Report e che si possono direttamente ottenere tramite le classi del JMF. In figura 3.7 si riporta una figura che mostra l'architettura del JMF.

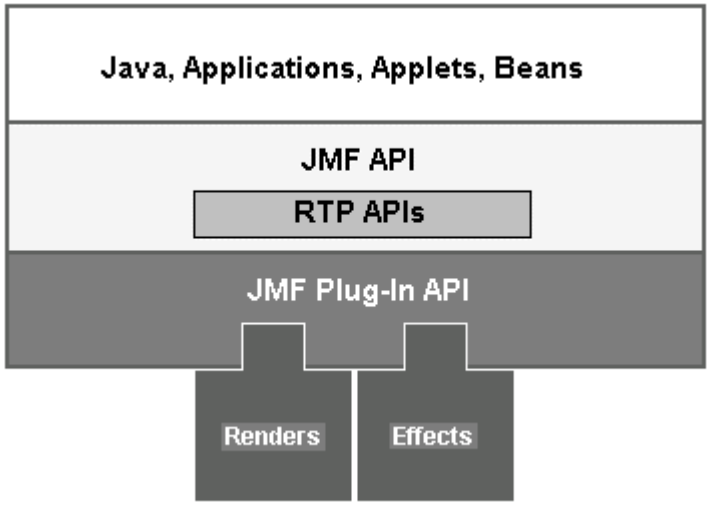

Figura 3.7: Architettura del JMF

3.4 SNMP API

L'ultima tecnologia che andiamo a presentare è una libreria di API sviluppata da AdventNet [Adv] per la gestione dei dati raccolti attraverso il protocollo SNMP. Tale libreria si è dimostrata uno strumento valido per lo sviluppo di applicazioni di network management che vogliono far leva sulla potenza di java e fornisce una buona base per la costruzione di soluzioni di gestione orientate all'utilizzo del protocollo SNMP. Infatti lo sviluppo di questo strumento ha seguito di pari passo l'evoluzione sia del linguaggio Java che, naturalmente, del protocollo SNMP.

3.4.1 Caratteristiche delle SNMPApi

Il pacchetto di API di AdventNet presenta diverse caratteristiche interessanti per lo sviluppo di applicazioni di network management. Esse infatti sono, ad esempio, **indipendenti dalla piattaforma**, dal momento che possono funzionare indistintamente su windows, linux o solaris. Inoltre sono **aperte agli standards**, in quanto seguono tutte le nuove tecnologie e gli standard legati ad Internet, in modo da essere compatibili con quelle con le quali si ha a che fare trattando di network management. In particolare possono interagire con i Java beans, http, RMI, CORBA, EJB e JDBC. In particolare sono completamente conformi con quelli che sono tutti gli standard legati al protocollo SNMP. Queste librerie offrono inoltre strumenti di sviluppi automatici e presentano caratteristiche di **scalabilità**, dal momento che sono

pensate per sistemi distribuiti basati su internet, e **flessibilità**, in quanto hanno una architettura che presenta una gerarchia di package Java con diversi livelli di accuratezza e semplicità.

3.4.2 Architettura delle SNMP Api

L'architettura delle SNMP API di AdventNet è strutturata in modo da presentare una gerarchia di package che garantiscono diverse
funzionalità (fig. 3.8).

Si parte da un livello di astrazione molto basso che fornisce funzionalità molto vicine al protocollo SNMP (Low-Level SNMP API), sino ad arrivare ad un altissimo livello di astrazione (Applications) che fornisce risultati a partire dalle funzionalità dei livelli inferiori.

3.4.2.1 Low-Level SNMP API

Le API di basso livello sono quelle che più ci interessano per quelli che saranno poi gli scopi del nostro lavoro e saranno perciò trattate più in dettaglio. Esse, come accennato in precedenza, sono molto vicine alle funzionalità proprie del protocollo SNMP. Esse sono in sostanza una collezione di quelle classi che implementano il protocollo SNMP secondo quanto stabilito dagli standard delle sue tre versioni, ossia v1, v2c, v3. Queste classi sono raccolte in un unico package, il snmp2, sufficiente per sfruttare tutte questa API.

Le Low-Level API sono suddivise in moduli funzionali, ognuno con le proprie caratteristiche:

- L'**SNMP transport provider framework** fornisce un livello trasparente per le communication classes che trasportano i pacchetti SNMP diretti agli apparati, sopra qualsiasi protocollo. Di default questo viene fatto sopra UDP, ma possono essere fatte anche scelte diverse. In definitiva questo modulo è il responsabile delle comunicazioni tra il manager e gli agents.
- Le **Variable Classes** forniscono l'astrazione dei dati del protocollo SNMP. Ognuna di esse ha infatti una corrispondenza uno-a-uno con i tipi di dato del protocollo. Ad esempio la classe SnmpInt corrisponde al tipo INTEGER del protocollo.

- Le **Security API** implementano le procedure per fornire un controllo sul livello di sicurezza sui messaggi e sugli accessi alle informazioni di gestione.

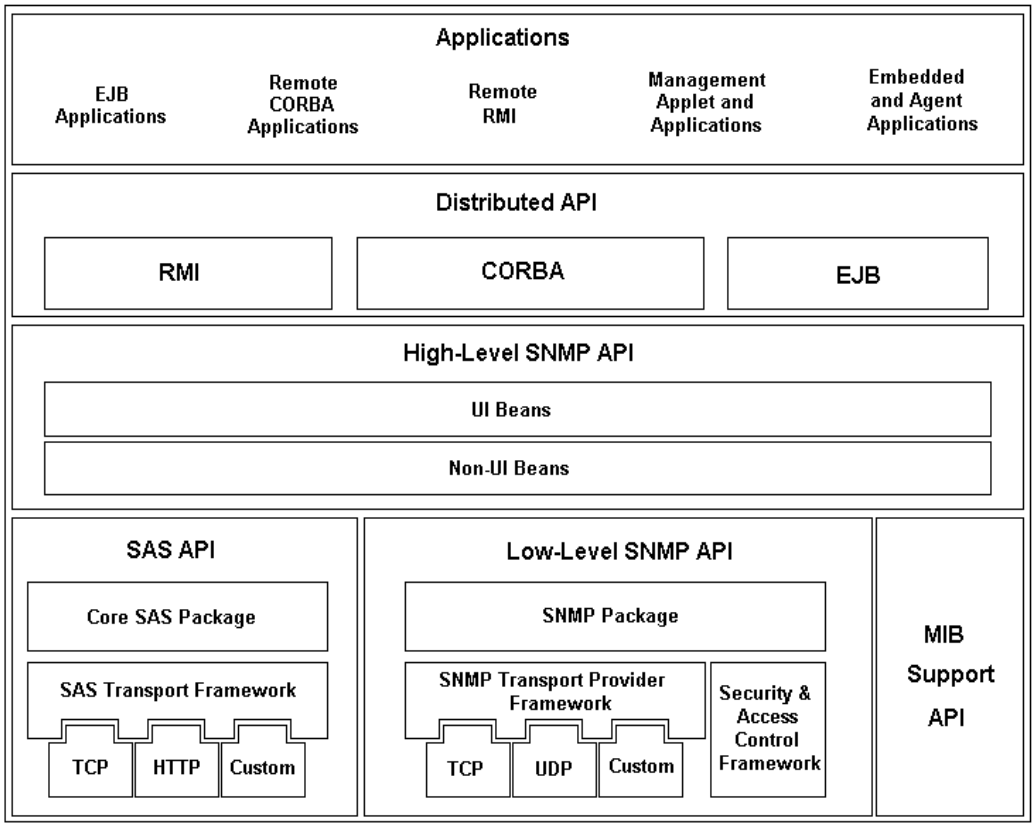

Figura 3.8: Architettura delle SNMP API di AdventNet

- Le **Communication classes**, oltre alla funzione già accennata, sono responsabili di stabilire una sessione SNMP con un'altra entità. Queste classi vengono utilizzate per mantenere i parametri di sicurezza da includere nei messaggi e le varie tabelle associate alla configurazione della sicurezza, permettendone creazione, modifica e persistenza e per settare alcuni parametri della sessione come ad esempio la versione. Inoltre gestiscono time-out e ritrasmissioni per le applicazioni, generano identificatori per i messaggi di richiesta uscenti, correlano le risposte alle richieste e facilitano nella creazione di oggetti PDU per la realizzazione di operazioni SNMP.
- Il **Security and Access Framework API** permette invece di andare a definire, a implementare e ad abilitare livelli di sicurezza personalizzati, lasciando allo sviluppatore molti gradi di libertà.

3.4.2.2 Gli altri livelli delle SNMP API

Le **High-Level API** consistono in UI e non-UI beans che possono essere usate per costruire applicazioni che incorporano le funzionalità fornite della API di basso livello. Gli UI beans possono essere usati per sviluppare applicazioni GUI di management e sono costruiti con un logica propria del protocollo SNMP, permettendo quindi di costruire applicazioni più flessibili. I non-UI beans offrono le funzionalità di base per le API di alto livello, tanto che gli altri beans sono costruiti sopra dei essi.

Le **MIBs API** portano le informazioni riguardo ai dati disponibili sugli agenti SNMP. Queste API facilitano il caricamento e lo scaricamento del MIB nelle applicazioni, in modo da avere in maniera più semplice delle informazioni sulle proprietà dei managed object. Anche queste API sfruttano i servizi delle API di basso livello.

Le **SAS API** forniscono un supporto per le applet Java in modo da superare le restrizioni di sicurezza dei browser.

Le **Distributed API** si pongono ad un alto livello di astrazione sopra le API di alto livello e si dividono in tre diverse categorie: le *RMI API*, poi le *CORBA API* e le *EJB API*. Queste API possono essere sfruttate nello sviluppo di applicazioni distribuite che fanno uso delle operazioni SNMP.

3.5 Conclusioni

In questo capitolo abbiamo presentato le tecnologie coinvolte nel nostro progetto di tesi. Si sono tracciate solo le caratteristiche principali di ogni tecnologia, mettendone in evidenza gli aspetti più rilevanti per i nostri scopi. In particolare abbiamo parlato di SOMA e MUM che sono gli ambienti nel quale il nostro progetto è stato sviluppato. Si è poi passati a descrivere le caratteristiche proprie di JMF, utilizzato nel trattare i dati deducibili attraverso i dati di sessione del protocollo RTP, ed infine delle SNMP API di AdventNet, usate proprio per andare a recuperare informazioni sulla rete per mezzo di questo protocollo.

Nel prossimo capitolo si inizieranno a trattare in dettaglio i passi che hanno portato alla realizzazione del nostro modulo di controllo e monitoraggio.

Capitolo 4

4 Analisi del modulo di controllo e monitoraggio per servizi multimediali

In questo capitolo si inizia a trattare nel dettaglio la problematica che sta alla base di questo lavoro di tesi. Fino ad ora si sono introdotti tutti i protocolli e le tecnologie che sono stati utilizzati, mentre i prossimi capitoli descriveranno il procedimento che si è seguito nel realizzare un modulo di controllo e monitoraggio per servizi multimediali. Si è scelto di seguire i passi propri del processo di ingegnerizzazione del software e si inizierà quindi trattando la fase di analisi del problema, per passare poi, nei prossimi capitoli, alla trattazione delle fasi di progettazione e di implementazione.

4.1 Analisi dei Requisiti

Il nostro modulo di monitoraggio va ad inserirsi all'interno di quello che è l'ambiente SOMA, e più in particolare nell'ambito del middleware MUM. Il nostro obiettivo è infatti quello di dare un supporto alla fruizione dei servizi multimediali tramite MUM, andandoci ad inserire in quelle che sono in MUM le tematiche legate alla gestione della Qualità del Servizio (QoS). Entrando più nel dettaglio, si vuole riuscire ad individuare, attraverso la nostra attività di controllo e monitoraggio, eventuali criticità nella disponibilità delle risorse, in modo che poi il sistema reagisca per garantire un predeterminato livello di Qualità del Servizio.

Come scritto nei capitoli precedenti, le risorse che più delle altre sono impiegate quando si vanno a trattare flussi multimediali sono la capacità computazionale, o **CPU**, e la **banda trasmissiva**. In particolare si vuole porre in maggiore evidenza questa seconda risorsa, e realizzare un modulo che possa gestire le sue fluttuazioni in maniera semplice, segnalando prontamente la presenza di situazioni di criticità e stimolando l'intero sistema ad azioni di riorganizzazione e riconfigurazione atte a risolvere queste situazioni. Quando da ora parleremo di risorse sarà perciò implicito il riferimento alla banda trasmissiva, anche se verranno sempre tenuti in considerazione gli effetti delle nostre azioni anche sulla CPU.

4.1.1 Intrusività del controllo

Naturalmente il prodotto che si cerca di realizzare dovrà seguire i principi elencati nel primo capitolo, ed in particolare il principio di minima intrusione, onde evitare sgraditi effetti di aggravamento nella situazione che si è andati a studiare.

Risulta perciò evidente il ruolo fondamentale che avrà, in questo senso, una larga fase di testing finale, attuata proprio con lo scopo di trovare il miglior compromesso tra frequenza e precisione dei rilevamenti ed intrusività del controllo. Infatti possiamo affermare che, con una frequenza di rilevamento sufficientemente alta, si possono ottenere andamenti della fluttuazione delle risorse sulla rete molto precisi, puntuali e tempestivi. Lo scotto da pagare è una alta incidenza su queste risorse. In particolare per ottenere questi risultati si devono immettere sulla rete una alta quantità di dati di controllo, che concorrono sicuramente alla creazione di una congestione sulla stessa. Inoltre si deve considerare che questa grande quantità di dati di controllo deve, una volta recuperata dalla rete, essere processata, andando a ridurre anche quella che è la disponibilità di CPU. Si deve quindi prevedere di trovare questo compromesso, dal momento che non è ovviamente accettabile neppure l'altro estremo di questa situazione, ossia un controllo con una frequenza molto bassa, per nulla intrusivo, ma anche molto impreciso e molto lento nel segnalare problemi.

4.1.2 Adattabilità e Portabilità

Quando si parla di monitoraggio si fa riferimento ad una gestione delle risorse di tipo reattivo, strettamente legata al principio di adattabilità che si realizza proprio tramite queste politiche di gestione. Come abbiamo visto in precedenza, il middleware MUM integra al suo interno sia gestioni di questo tipo che politiche di negoziazione statica e perciò di tipo pro-attivo. Uno dei nostri obiettivi sarà quindi quello di andare a sviluppare queste politiche, tenendo conto dei loro effetti.

Inoltre la soluzione che si vuole realizzare deve necessariamente essere portabile. Per questo motivo si devono usare strumenti abbastanza generali che possano garantire il loro funzionamento su diverse piattaforme. In questi casi risulta sempre molto vantaggioso ricorrere all'utilizzo di standard, siano essi reali o de facto.

4.2 Analisi

In questa sezione verrà effettuata l'analisi delle varie parti costituenti il modulo, e delle loro reciproche interazioni, e si produrrà un'architettura logica che guiderà la successiva fase di progetto del sistema.

4.2.1 Analisi dei protocolli

Per prima cosa andiamo ad analizzare quelli che sono i due protocolli che utilizzeremo nella realizzazione del nostro progetto, in modo da capire quali sono i loro aspetti che sono utili per gli scopi che cerchiamo di conseguire. I protocolli in questione sono quelli trattati in dettaglio al capitolo 2, ossia il protocollo RTP/RTCP ed il protocollo SNMP.

Questa divisione suggerisce un primo sdoppiamento del problema, dal momento che sembra più sensato realizzare due moduli concorrenti che vadano ad operare sui diversi dati di controllo forniti da questi due protocolli. In particolare dobbiamo notare che il protocollo RTCP fornisce dati che si riferiscono ad una determinata sessione RTP, mentre attraverso il protocollo SNMP si possono ottenere dati che si riferiscono al comportamento di un generico nodo sulla rete.

Alle luce di queste considerazioni possiamo pensare di andare a realizzare un sotto-modulo di controllo RTP/RTCP, con il compito di analizzare i dati relativi ad una particolare sessione di trasmissione dati RTP e, in maniera del tutto duale, un sotto-modulo di monitoraggio che sfrutti i dati ottenibili attraverso il protocollo SNMP e che si riferisca alla situazioni nella quale si trova la macchina sulla quale questo modulo sta operando (fig 4.1).

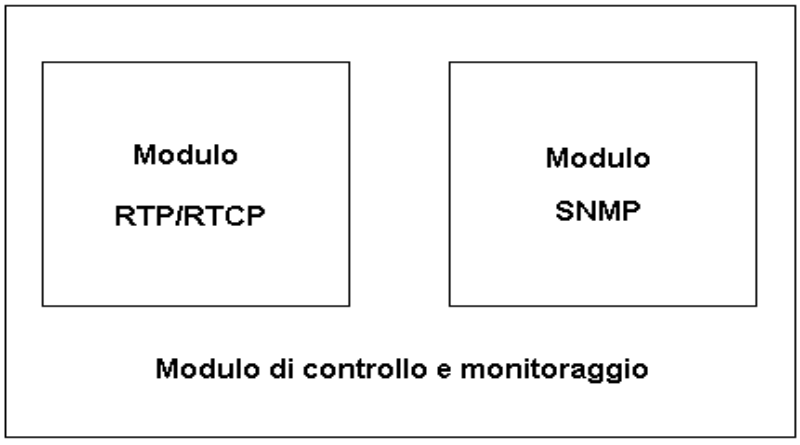

Figura 4.1: Architettura del modulo di controllo e monitoraggio

4.2.2 Analisi del modulo RTP/RTCP

Come detto, i dati di controllo ottenibili tramite RTCP fanno riferimento ad una ben precisa sessione dati RTP. Ne consegue che ogni entità che partecipa ad una sessione potrà effettuare dei controlli su questi dati. In realtà però questo controllo è pensato per essere attuato da ogni proxy ed esiste una corrispondenza proprio tra questo ed un monitor. In particolare si può pensare di istanziare un monitor della sessione RTP per ogni gestore della Qualità del Servizio collegato ad una entità proxy, ossia un monitor per ogni proxy, se si considera una stessa sessione, ed un monitor per ogni sessione che è attivata su un determinato place sul quale esiste un proxy.

Questa situazione è di facile realizzazione se andiamo ad analizzare la struttura di MUM e se si tiene poi conto che le precedenti semplici politiche seguivano questa stessa soluzione.

Questo componente verrà quindi riferito ed interrogato dal manager della qualità del servizio, quando questo vorrà avere informazioni sulla situazione della banda durante la sessione RTP, ed il monitor risponderà a questa invocazione stimolando un controllo sui dati attuali riguardanti tale sessione e generando, qualora fosse necessario, eventi di rilevazione di un determinato problema. Alla ricezione di questi eventi il manager della qualità del servizio deve rispondere stimolando una riconfigurazione dinamica del sistema, in modo da garantirne il funzionamento anche se ad un livello di qualità inferiore.

4.2.2.1 Analisi dei dati RTCP

Per realizzare un controllo funzionale sul comportamento di una sessione dati RTP si possono sfruttare le caratteristiche proprie di questo protocollo andando ad analizzare i dati RTCP ad essa collegati. Infatti la peculiarità interessante di questo protocollo è quella di avere già al suo interno questo flusso di dati di controllo senza che noi dobbiamo andare ad introdurre ulteriori traffici sulla rete per ottenere informazioni. Basterà quindi recuperare questi dati ed analizzarli per avere direttamente il “polso” della situazione.

Occorre però comprendere quali di questi dati possono essere più funzionali per gli scopi che ci siamo preposti, dal momento che molti di essi potrebbero non fornirci indicazioni adatte.

Per prima cosa va sottolineato il fatto che non tutti i pacchetti RTCP ci possono essere utili. Possiamo tranquillamente sostenere che i pacchetti sui quali ci dovremo concentrare sono i pacchetti di *Sender Report* e di *Receiver Report*. All'interno di questi tipi di pacchetti possiamo infatti trovare i blocchi relativi al traffico, ossia i *Report Block*, dove sono già incapsulate molte utili notizie sulla situazione della rete. Una simile situazione ci permette di affermare che si possono fare controlli ogni volta che si verifica l'evento di arrivo di uno di questi due pacchetti. Questa scelta ci toglie un grado di libertà, dal momento che non possiamo più scegliere direttamente la frequenza con la quale attuare il monitoraggio, ma risulta ragionevole se pensiamo che comunque non avrebbe senso stimolare un controllo a frequenza maggiore perché il risultato sarebbe quello di processare due o più volte gli stessi dati, appesantendo la computazione e senza ottenere miglioramenti della precisione del monitoraggio. Questa caratteristica rende il controllo attraverso RTCP intrinsecamente lento, in quanto, per rendersi conto della presenza di un problema, si deve necessariamente attendere l'arrivo di un pacchetto di report che lo individui, introducendo un ritardo, che sarà necessariamente aleatorio, sulla risposta del nostro sistema. Tuttavia questo controllo rimane molto interessante per le informazioni che porta e vale sempre la pena di attuarlo.

Non tutte le informazioni che si trovano all'interno di un report block sono però adatte ad essere analizzate quando si vuole andare ad effettuare un controllo sulla banda trasmissiva. Occorre perciò analizzare ogni singolo campo di questi blocchi per conoscerne pregi e difetti e giustificarne l'utilizzo nella nostra attività di monitoraggio.

Il primo campo che possiamo prendere in considerazione è quello che identifica il ritardo dall'ultimo arrivo di un pacchetto di report. Questa grandezza potrebbe essere scelta in modo da avere una valenza di time-out, ossia si potrebbe pensare di affermare che se non arrivano pacchetti di controllo per un determinato intervallo di tempo, questo significa che vi sono problemi sulla rete tali da impedirlo. Questa soglia deve necessariamente essere molto alta, il che rende questo controllo pressoché inutile. Si può infatti ragionevolmente pensare che, se ci troviamo di fronte ad un problema sulla rete tale da impedire l'arrivo di pacchetti sulla nostra macchina, questa situazione si possa rilevare molto più velocemente attraverso altri controlli, come potrebbero essere quelli fatti tramite il protocollo SNMP.

Un dato che invece porta informazioni funzionali ai nostri scopi è quello contenuto nel campo relativo al jitter. Questo infatti, come già definito in precedenza, rappresenta la varianza, ossia lo scostamento dal valore medio, della latenza dei pacchetti ed un suo aumentare è un sicuro sintomo di una congestione sulla rete o comunque di una situazione che va diventando problematica.

L'ultimo campo di un report block da analizzare è quello che si riferisce alla frazione di persi. Infatti una delle caratteristiche di RTP è quella di riuscire a sopperire ad una eventuale e contenuta perdita di dati, riuscendo a garantirne comunque la fruizione. Naturalmente quando questa frazione diventa troppo grande, non solo ci possono essere problemi per utilizzare i dati in arrivo, ma significa anche che ci sono malfunzionamenti sulla rete che intralciano il corretto delivery dei pacchetti. Questo è perciò un dato da tenere in sicura considerazione quando si effettua un monitoraggio sulla banda.

Sino ad ora abbiamo preso in considerazione solamente dati direttamente ottenibili attraverso i pacchetti RTCP di Sender Report e Receiver Report, ma ci sono anche altre grandezze, facilmente calcolabili, che possono essere controllate. Naturalmente non tutte queste grandezze hanno le caratteristiche adatte per essere utilizzate per un monitoraggio sulla banda trasmissiva.

Per prima cosa si potrebbe pensare di effettuare un controllo sulla bit-rate per avere delle notizie dirette sulla banda. Si deve però tenere conto del fatto che i dati trasmessi tramite RTP sono codificati utilizzando algoritmi di compressione che in generale non mantengono costante il valore della bit-rate. Inoltre questa cambia notevolmente a seconda delle caratteristiche che hanno i dati trasmessi e quindi le sue fluttuazioni sono molto ampie, non solo tra due presentazioni diverse, ma anche tra diversi momenti di una medesima presentazione.

Un discorso analogo può essere fatto se si vuole prendere in considerazione la frequenza di arrivo dei pacchetti. Anche in questo caso non si possono raccogliere informazioni generali da questo dato che dipende notevolmente dalla dimensione dei pacchetti stessi e dal tipo di dato che questi trasportano.

Un controllo che invece potrebbe fare al caso nostro è quello che riguarda la frame-rate. Essa infatti non risente degli stessi problemi delle grandezze appena descritte ed anzi presenta un valore costante in condizioni ottimali, il quale subisce poi un notevole calo quando le condizioni vanno peggiorando, con il risultato di rendere molto difficile il rendering dei dati arrivati.

In definitiva abbiamo individuato quelle grandezze che risultano sufficienti per andare a fare delle considerazioni esaustive sulla maniera nella quale si sta evolvendo una determinata sessione RTP. Abbiamo scelto di utilizzare il **jitter**, per stimare come sta cambiando dinamicamente la latenza dei pacchetti che stiamo inviando, la **frazione di pacchetti persi**, per valutare se e quanti dei nostri pacchetti di dati riescono a raggiungere la loro destinazione, e la **frame-rate**, per capire se la qualità del nostro rendering sta subendo dei peggioramenti o sta continuando ad svilupparsi in maniera lineare e continua. L'analisi congiunta questi tre valori risulta efficace per decidere se il caso o meno di prendere provvedimenti con lo scopo di migliorare la situazione che si è arrivati a riscontrare.

4.2.3 Analisi del modulo SNMP

Il sotto-modulo SNMP opera sopra i dati ottenuti grazie a questo protocollo ed ha una funzione un po' diversa rispetto al precedente. Questo infatti andrà ad operare a livello di gestione locale, ossia all'interno di quel gestore delle risorse che, nel middleware MUM, deve essere presente su ogni macchina. Esso è perciò indipendente dalle sessioni RTP e continua ad operare anche in assenza di esse. In particolare possiamo dire che questo modulo non avrà diverse istanze, ma una singola istanza per ogni macchina sulla rete.

In sostanza questo monitoring si trova ad operare nell'ambito del gestore delle risorse sulla singola macchina e viene stimolato da componenti interni ad esso. In figura 4.2 si esplicita questo concetto, riprendendo quella che è la struttura di MUM, e sottolineando quale sarà il campo dove si andrà ad operare. La frequenza di questo controllo, a differenza del precedente, può essere scelta, mentre non cambia in sostanza il comportamento susseguente ad una rilevazione di un problema. Infatti, analogamente al caso precedente, al verificarsi di questa situazione vengono generati degli eventi per i quali si mette in ascolto il gestore della qualità del servizio.

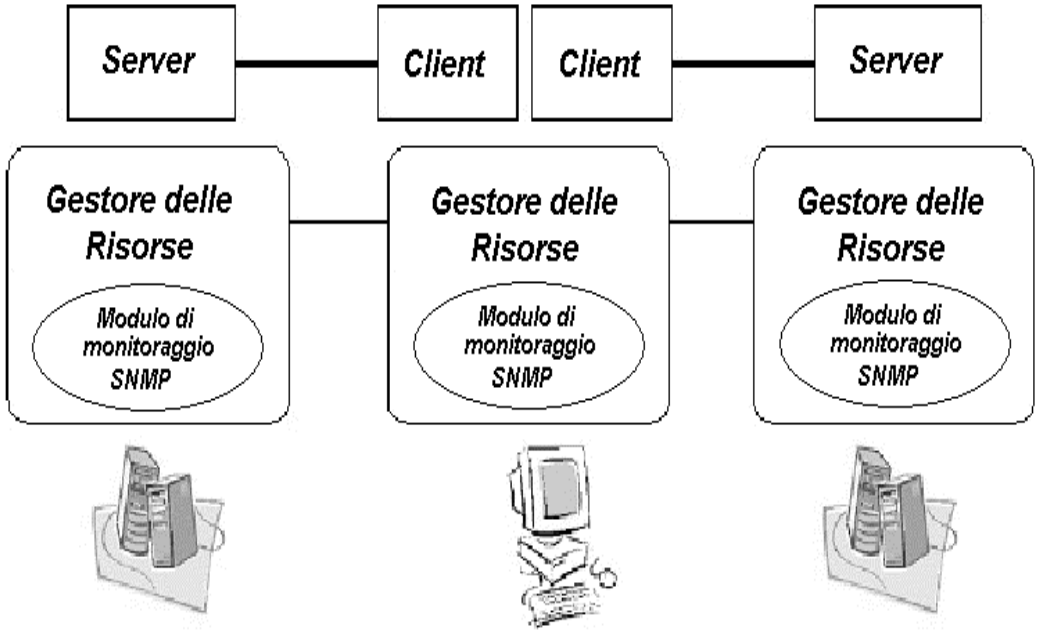

Figura 4.2: Architettura del sotto-modulo di controllo SNMP

4.2.3.1 Analisi dei dati SNMP

Utilizzando il protocollo SNMP si riescono ad ottenere tutte le informazioni residenti su di una struttura dati, detta MIB (Management Information Base), presente su ciascuna macchina. Nel nostro caso non siamo molto interessati all'applicazione delle funzioni del protocollo SNMP per ottenere informazioni riguardanti nodi remoti, quanto più ad utilizzarle per avere un quadro completo di quello che sta accadendo sulla macchina sulla quale ci troviamo. Si poteva quindi ricorrere anche ad altri approcci, ma l'utilizzo di questo protocollo ci garantisce una certa portabilità, dal momento che SNMP è standard de facto in fatto di gestione delle informazioni di rete.

Ovviamente non siamo interessati a tutti i dati residenti sul MIB, ma solo ad alcuni di essi. In particolare per i nostri scopi sono interessanti solamente le informazioni riguardanti il sotto-albero *IP* ed il sotto-albero *interfaces*. Nel primo di questi due sotto-alberi possiamo trovare tutte le informazioni riguardanti i pacchetti relativi al protocollo IP, mentre nel secondo ci sono informazioni per noi interessanti come, ad esempio, il numero totale di byte in entrata ed in uscita. La scelta di andare ad analizzare solo il sotto-albero IP deriva dalla constatazione che la maggior parte del traffico che dobbiamo monitorare si appoggia proprio su questo protocollo. Inoltre il sotto-albero IP del MIBII, ossia la versione del MIB attualmente utilizzata,

ha informazioni molto più dettagliate rispetto a sotto-alberi dedicati ad altri protocolli. Si potrebbe infatti ragionevolmente pensare di andare a prendere in considerazione i protocolli TCP e UCP, ma le informazioni sul MIB relative a questi sono molto più generiche e meno rilevanti, senza poi contare che questi due protocolli si appoggiano sempre, per funzionare correttamente, sul protocollo IP. Questa scelta introdurrebbe quindi una sorta di ridondanza, con il solo effetto di creare un inutile sovraccarico della capacità computazionale.

Naturalmente non tutti i campi relativi ad IP sono per noi rilevanti, dal momento che molti di essi non restituiscono informazioni sui pacchetti. In particolare siamo interessati al numero di pacchetti IP errati in ingresso e uscita, al numero totale di pacchetti ed al rapporto tra queste due grandezze. Queste grandezze ci danno una stima della congestione sul traffico IP che è la stragrande maggioranza del traffico esistente. Tutti questi dati sono contenuti in diversi nodi del MIB. Ad esempio ci sono diversi nodi che identificano il numero di pacchetti errati in uscita, raggruppati ognuno per una diversa motivazione. Ci sono i pacchetti che sono stati scartati a causa di un errore presente nel loro header, mentre altri sono scartati perché presentavano un errore di indirizzo. Vi sono poi pacchetti che non vengono accettati perché si riferiscono ad un protocollo che non viene riconosciuto, altri perché la loro destinazione finale non è raggiungibile, altri ancora vengono scartati senza una di queste precise ragioni, ma per altri motivi, come può essere ad esempio una mancanza di spazio nel buffer di ricezione. Tutte queste motivazioni vanno tenute in considerazione, dal momento che noi siamo interessati al numero totale di pacchetti persi, valore che in definitiva è rappresentato dalla somma dei campi che abbiamo appena descritto. Discorso analogo può essere fatto per il numero di pacchetti errati in uscita, anche se in questo caso sono presenti nel MIB solo due campi che si riferiscono ai pacchetti persi per una causa generica ed a quelli persi perché non si è trovato il router che li inviasse.

Andando ad analizzare il sotto-albero delle interfacce ed in particolare i campi che si riferiscono al numero di totale di byte, sia in ingresso che in uscita, fa in modo che si possa calcolare una stima della reale larghezza di banda trasmissiva presente sulla linea in quel momento. Dobbiamo notare però come questa informazione possa essere interessante in senso assoluto, ma come mal si adatti per essere usata nella realizzazione di un controllo. Infatti da queste informazioni si può trarre una stima della larghezza di banda, ma questa stima sarebbe fatta a posteriori, a seconda del

traffico passato sulla macchina. In pratica, se la macchina non dovesse trasmettere alcun dato per un determinato intervallo di tempo, la banda risultante avrebbe un valore nullo, ma questo non significa assolutamente che sulla linea, in quel intervallo di tempo, si siano verificati dei problemi.

Tutti le altre informazioni presenti sul MIB non hanno molta rilevanza o non aggiungono nulla alle considerazioni appena fatte sul protocollo IP.

4.3 Conclusioni

In questo capitolo sono stati presentati i requisiti che il nostro modulo deve avere. Si è poi passati a descrivere i motivi per i quali si è scelto di sdoppiare questo modulo e, per entrambi questi sotto-moduli, quali sono le grandezze da controllare, evidenziando pregi e difetti di ogni scelta. Nel prossimo capitolo si farà un successivo passo, andando a discutere quelle che sono le scelte progettuali che si sono fatte.

Capitolo 5

5 Progettazione del modulo di controllo e monitoraggio per servizi multimediali

In questo capitolo si andranno a descrivere le scelte progettuali che hanno portato alla realizzazione del nostro modulo di monitoraggio. In particolare si descriveranno in due sezioni separate i procedimenti relativi ai due sotto-moduli descritti al capitolo precedente. Inoltre si faranno espliciti riferimenti alle tecnologie utilizzate e descritte approfonditamente al capitolo 3.

5.1 Metodologie di controllo

Fino a questo momento si è parlato solamente di come riuscire ad ottenere dati da analizzare e quali dati andare a ricercare. Non si è mai spiegato invece come andare a processare i dati così ottenuti, per capire quali politiche di gestione andare ad attuare.

Si è fatta una scelta particolare, mutuata dalla teoria delle carte di controllo. In pratica per ogni grandezza da osservare vengono stabilite due differenti soglie. La soglia più alta rappresenta il valore massimo accettabile per tale grandezza, mentre la seconda è ad un livello più basso, e rappresenta solamente una soglia di allarme. Quando la grandezza osservata supera il primo limite viene scatenato immediatamente un evento di problema, dal momento che abbiamo un valore non accettabile, mentre se viene superata la soglia di allarme, o di warning, viene aumentato un contatore. Se invece il valore osservato rimane sotto entrambe queste due soglie il contatore viene subito azzerato. Quando questo contatore raggiunge un valore prestabilito, per esempio il valore 3, si ha lo stesso effetto del superamento dalla soglia di accettabilità, scatenando un evento di problema. In questa maniera si possono scoprire subito eventuali problemi sulla rete, e si possono anche raccogliere andamenti problematici, dove le risorse vanno lentamente peggiorando. Infatti l'accorgimento dell'utilizzo di un contatore che si azzera quando le condizioni sono buone, fa in modo che l'evento di raggiungimento del valore prestabilito di tale contatore avvenga solamente quando si susseguono diverse situazioni di warning consecutive, che ovviamente stanno ad indicare il progressivo peggioramento delle

risorse. Inoltre, in questa maniera, un singolo valore "sballato" non provocherà mai un evento di riconfigurazione, a meno che non presenti un valore assoluto molto alto.

5.2 Progettazione del modulo RTP/RTCP

Come descritto nel capitolo precedente, il sotto-modulo di controllo RTP/RTCP si andrà ad inserire a livello di gestione della singola sessione RTP. Avremo perciò una classe di monitor per ognuna delle sessioni, in rapporto uno-a-uno con l'oggetto che rappresenta il gestore della qualità del servizio. Questo nostro monitor dovrà quindi avere lo stesso ciclo di vita del manager della qualità del servizio, dal quale sarà poi interrogato. Si è già introdotta la problematica legata alla frequenza del controllo, ma si può aggiungere, a questo punto, che le invocazioni saranno fatte alla ricezione, da parte del QoS Manager, di un evento di ricezione di un pacchetto Sender Report o di un pacchetto Receiver Report.

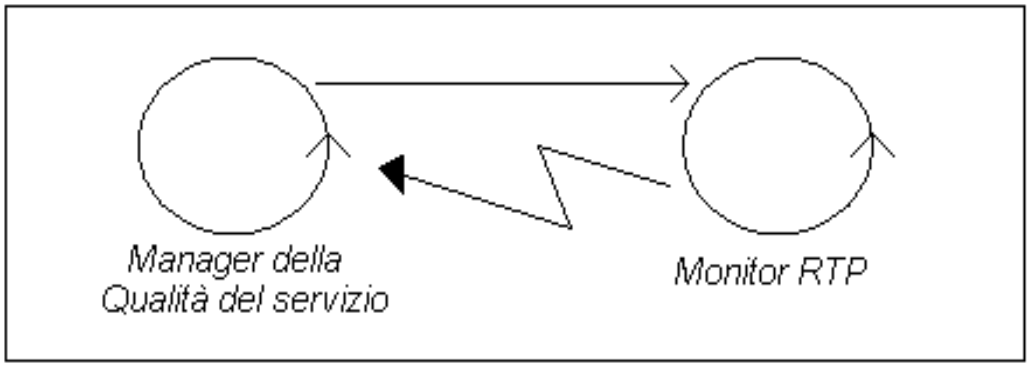

Figura 5.1: Relazioni del sotto-modulo di monitoraggio RTP

Nella figura 5.1 si esplicitano le relazioni che abbiamo descritto sino ad ora, mentre nel diagramma UML in figura 5.2 si evidenziano in maniera formale le caratteristiche che verranno poi trattate nei prossimi punti.

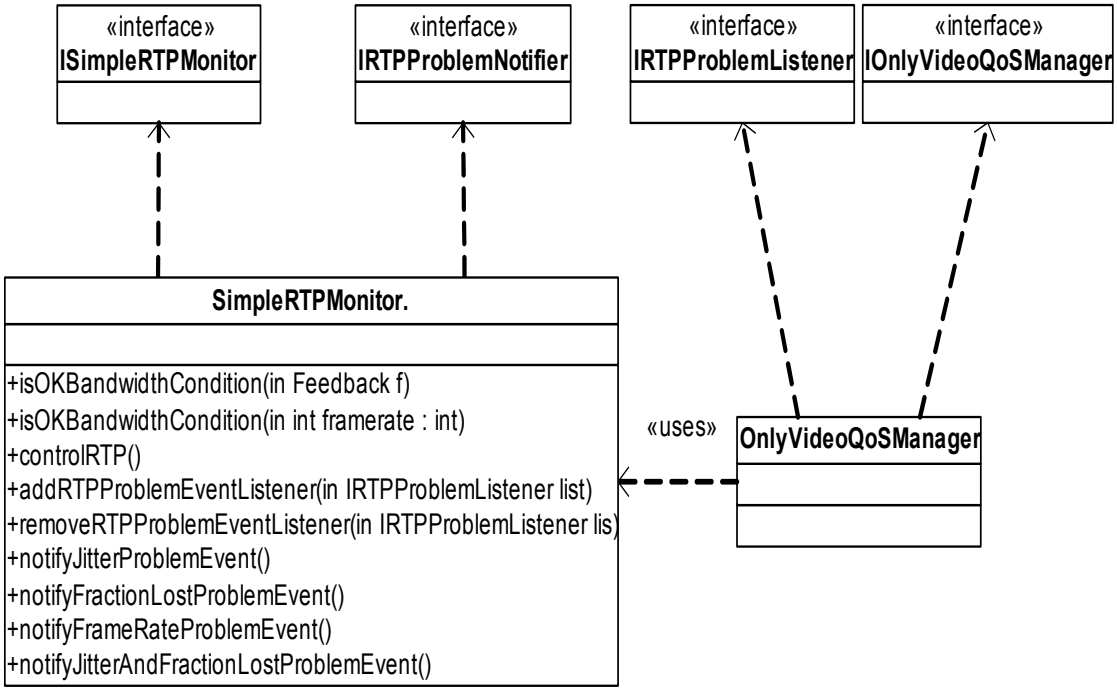

Figura 5.2: Diagramma UML del sotto-modulo RTP

5.2.1 Metodi di controllo

Il nostro monitor, che chiameremo RTP Monitor, dovrà avere la possibilità di recuperare le informazioni che andrà poi a processare ed a controllare.

In particolare si può pensare di avere un metodo di controllo dedicato per ognuna delle grandezze che si è deciso di prendere in considerazione, ossia jitter, frazione di persi e frame-rate. Bisogna però andare a considerare la tecnologia che dobbiamo utilizzare. A tale riguardo abbiamo in precedenza sottolineato come le prime due grandezze vengano ottenute entrambe direttamente dal pacchetto di report e, per questo motivo, sarebbe indicato andarle a processare insieme visto che i loro valori cambiano contemporaneamente, magari andando ad incapsulare queste informazioni in un unico oggetto che si riferisca ad un determinato report. Ha perciò senso, in quest'ottica, andare a dividere il monitoraggio solamente in due, controllando da una parte la frame-rate e dall'altra, congiuntamente, i valori di jitter e fraction lost recuperati dal report. Il codice di questi due metodi dovrà implementare la politica di controllo descritta al punto precedente, stabilendo le soglie e tenendo conto dei livelli di attenzione.

Inoltre possiamo ragionevolmente pensare che questi metodi andranno in overheading tra loro.

5.2.2 Generazione degli eventi

Il risultato del nostro controllo deve essere quello di scatenare la riconfigurazione dinamica del nostro sistema. Occorre quindi che il nostro monitor riesca a comunicare al gestore della qualità del servizio, che lo interroga, il risultato dei suoi controlli. La soluzione più semplice è quella di sfruttare il valore di ritorno dei metodi di controllo. La sua semplicità va però a scapito della potenza semantica. Se questo valore di ritorno fosse ad esempio un booleano sapremmo che si è verificato un problema sulla linea, ma non potremmo sapere di che problema si tratta.

Per risolvere questa situazione, e fornire quante più notizie possibili all'utilizzatore, si è ricorso all'impiego di eventi, ognuno dei quali descrive una diversa situazione di problema sulla rete. In questa maniera otteniamo anche un sistema più flessibile, dal momento che il gestore della qualità del servizio non è più l'unica entità che può essere messa a conoscenza di una problematica situazione sulla rete, ma si può pensare di girare queste informazioni anche ad altri. Inoltre l'uso di eventi permette ad un ipotetico futuro sviluppatore di creare eventi personalizzati, a seconda delle sue esigenze particolari, potendo riutilizzare la struttura creata. Scendendo nei dettagli, il manager della QoS rappresenta il listener posto in ascolto dei nostri eventi, mentre il nostro monitor RTP impersona il ruolo di notifier, ossia ha il compito di generare gli eventi di problema. Si vuole poi far notare che sarebbe consigliabile seguire in questo sviluppo i principi alla base della progettazione detta "per interfacce", in modo da garantire un certo livello di standardizzazione ed omogeneità.

Infine si ritiene utile andare a creare una gerarchia di eventi, ossia avere una classe, che potremmo definire "primitiva", che identifica il generico evento di problema RTP. In questa maniera si possono realizzare ulteriori eventi specializzando questa classe, al fine di concorrere ad aumentare il livello di omogeneità. In particolare si potrebbe pensare di avere una classe diversa per ogni motivazione che potrebbe essere alla base di un evento di problema, come mostrato in figura 5.3.

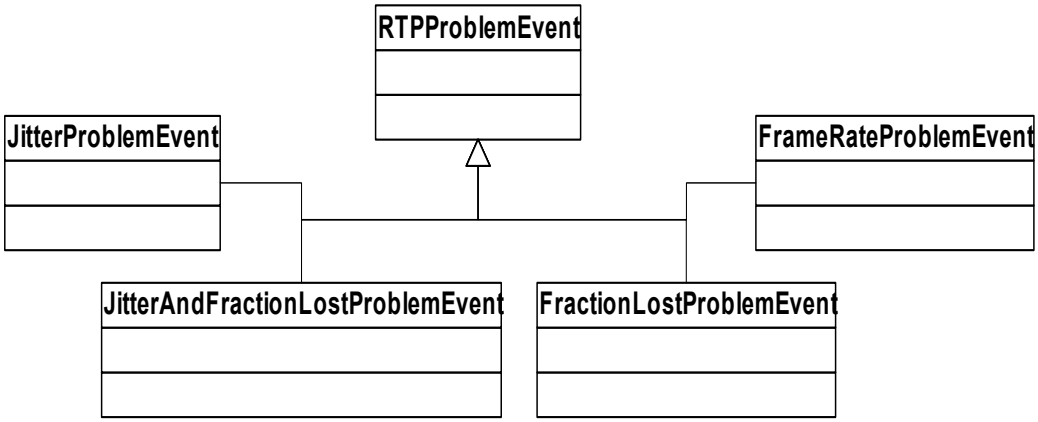

Figura 5.3: Gerarchia degli Eventi di problema legati ad RTP

5.3 Progettazione del modulo SNMP

Il sotto-modulo di controllo SNMP va a collocarsi all'interno del gestore delle risorse locale, presente su ogni macchina. Inizialmente, in questo componente di MUM, vi era un solo oggetto, chiamato *Broker*, che aveva sia la responsabilità di controllare che vi fossero risorse libere da allocare, sia quella di stimolare semplici controlli sulla banda trasmissiva e sulla capacità computazionale.

A questo punto si potrebbe pensare di fare in modo di deresponsabilizzare questo broker, in modo da avere un oggetto con il compito di organizzare il suo lavoro e quello di controllo del monitor SNMP che dobbiamo andare a realizzare. Tale oggetto, che potrebbe essere chiamato *Resource Manager*, diventerebbe il fulcro del componente di gestione locale delle risorse.

Ovviamente sia il monitor che il broker saranno in rapporto uno-a-uno con questo oggetto manager, dal momento che su una singola macchina vi sarà un unico gestore delle risorse.

In figura 5.4 è esplicitata la situazione appena descritta, tenendo poi conto di chi deve realmente ricevere le informazioni trovate dal nostro monitor, ossia il gestore della qualità del servizio.

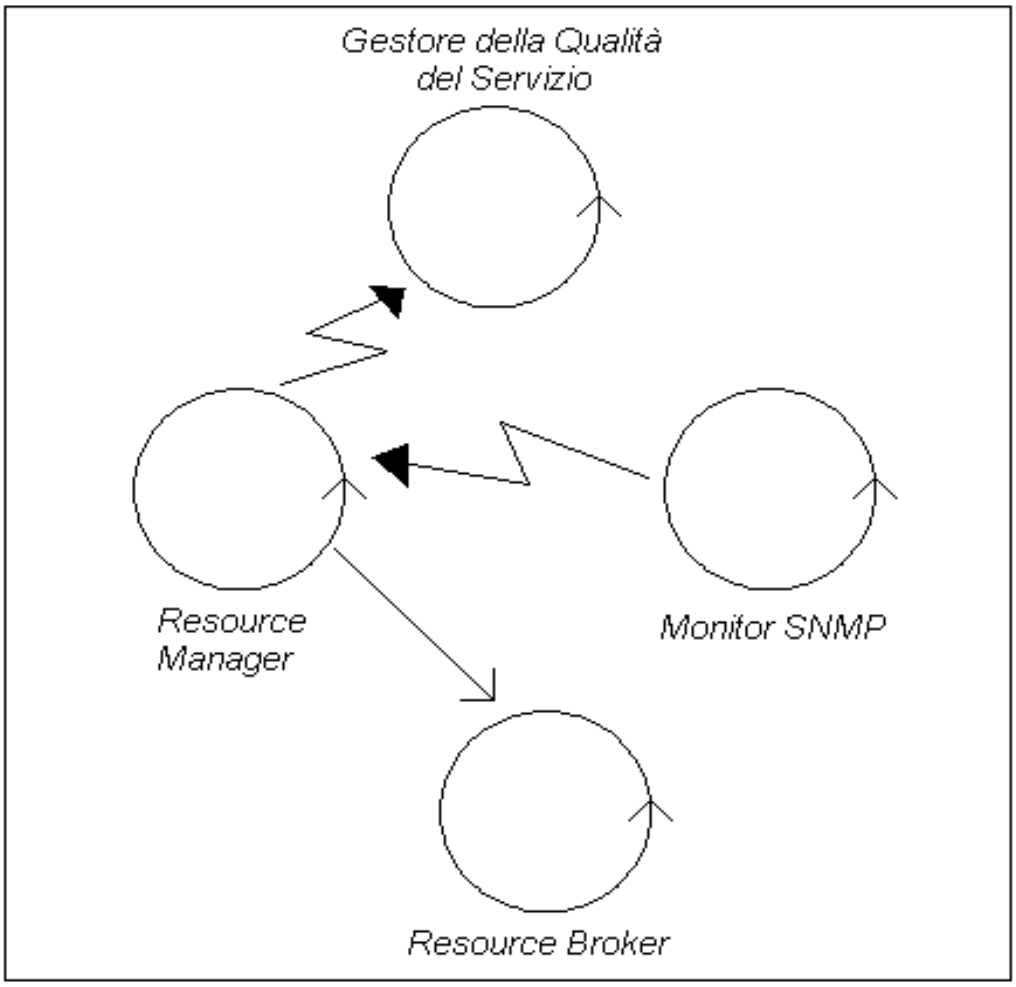

Figura 5.4: Relazioni del sotto-modulo di monitoraggio SNMP

5.3.1 Utilizzo del protocollo SNMP

Per realizzare il monitor SNMP siamo obbligati a raccogliere le informazioni sul MIB in una qualche maniera. Inoltre, come già sottolineato, non siamo interessati alle informazioni su host remoti, quanto più a quelle che si riescono ad estrarre sul MIB locale, in modo da poter comprendere quale sia il comportamento della macchina che stiamo usando.

Dovremo perciò utilizzare una delle primitive che il protocollo prevede ed in particolare potrebbe farci comodo utilizzare la primitiva di GET. Abbiamo infatti già stabilito quali dovranno essere le grandezze sulle quali andremo ad effettuare il controllo e possiamo sottolineare come queste grandezze corrispondano a ben precisi Oid. Non è perciò necessario fare chiamate ricorsive al MIB tramite, ad esempio, una GET-NEXT, dal momento che non tutte le informazioni di un sottoalbero ci interessano e caricheremmo inutilmente la CPU. Si può poi notare come questa azione sia ripetitiva e sempre uguale e si può quindi ragionevolmente pensare di non

farla fare direttamente al nostro monitor, ma di delegare questa responsabilità ad un oggetto dedicato che verrà poi utilizzato dal monitor stesso. Questo oggetto si chiamerà, con estrema fantasia, *SnmpGet* ed andrà a recuperare le nostre informazioni direttamente sul MIB. Per riuscire a trattare in maniera semplice tutte queste informazioni si può prevedere la creazione di un oggetto che le contenga e che permetta di poterle scambiare tra monitor e SnmpGet. Tale oggetto rappresenterà il metadato che descrive la situazione della rete nel preciso momento nel quale è stato effettuato il controllo e potrebbe perciò essere chiamato *Network State*.

Nel diagramma UML in figura 5.5 è riassunta la situazione appena descritta, con gli oggetti trattati e le loro inter-dipendenze, specificando anche le interfacce che dovranno essere implementate da ogni classe, seguendo, come in precedenza uno schema di progettazione proprio ad "interfacce".

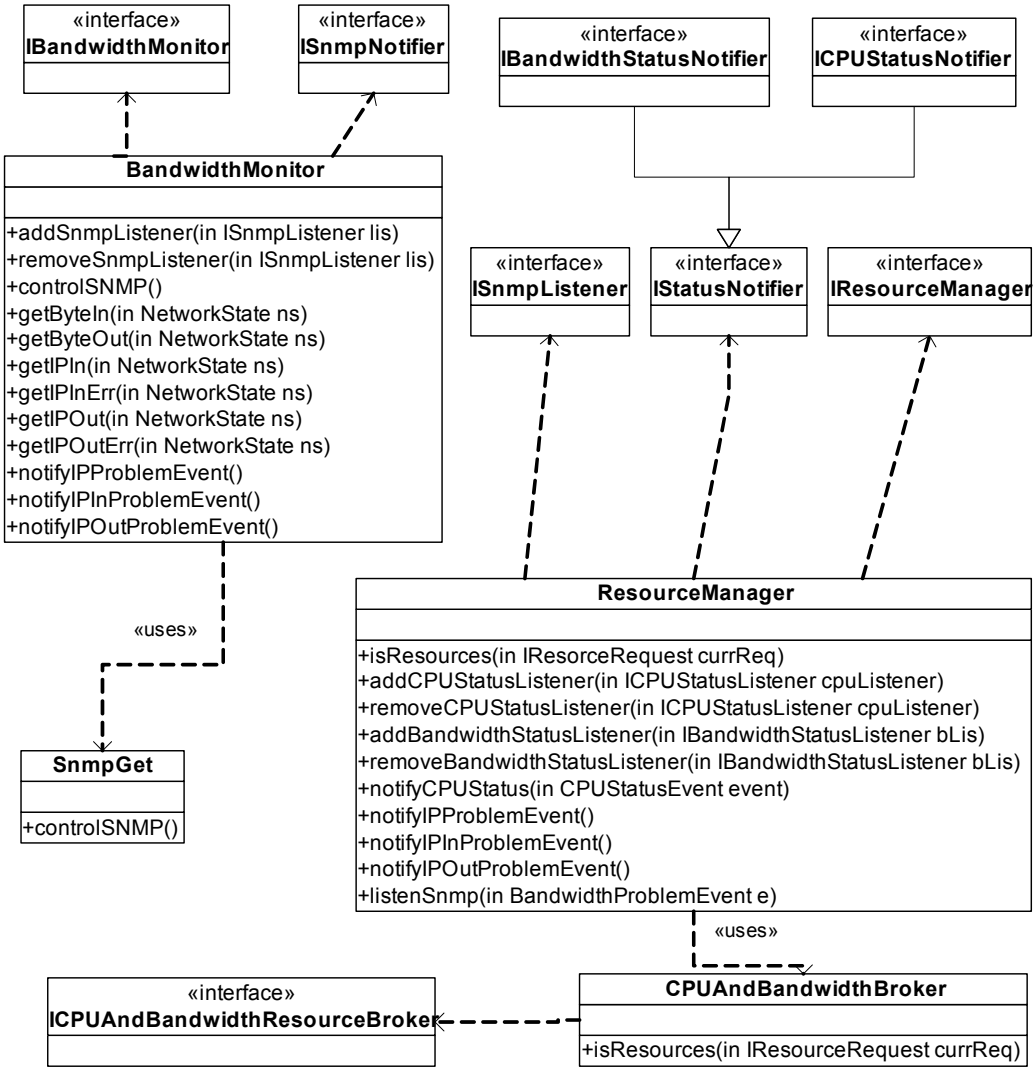

Figura 5.5: Diagramma UML del sotto-modulo SNMP

5.3.2 Generazione degli eventi

Analogamente al precedente, anche questo sotto-modulo di controllo tramite SNMP deve poter comunicare con il gestore della qualità del servizio, ma, a differenza di quello, deve poter inviare i risultati dei suoi controlli a tutti i gestori della QoS di tutte le sessioni attive sulla macchina locale. Risulta perciò parecchio conveniente anche in questa occasione andare a realizzare le notifiche di avvenuto problema sulla rete attraverso la generazione di eventi. Ripercorrendo quelle che sono state le motivazioni addotte in precedenza, si può realizzare una gerarchia di eventi che ne specializzino uno generico, e realizzare delle interfacce che dovranno essere implementate da chi vuole generare (notifier) o ascoltare (listener) questi eventi.

Dal diagramma UML si può notare come la classe *Monitor* interagisca tramite eventi con la classe *Manager*, dal momento che implementano le rispettive interfacce di notifier e listener. Questa soluzione permette al Manager di mettersi in ascolto di molti monitor, per poi generare eventi congiunti da spedire al gestore della qualità del servizio. Infatti, anche se nel nostro caso vi è un solo monitor dedicato al controlli sulla banda, in futuro si potrebbero aggiungere componenti che controllino altre risorse, come per esempio la CPU, senza che il nostro Resource Manager debba essere pesantemente modificato. Inoltre si può notare come il Manager implementi sia l'interfaccia di notifica di problema per la banda che quella per i problemi sulla CPU. Questa scelta deriva dalla volontà di mantenere una sorta di compatibilità con le versioni precedenti dove era il Broker a doversi assumere queste responsabilità. In più possiamo aggiungere che in questo modo basterà andare a lavorare solo su queste due interfacce, che sono già riconosciute, per avere notifiche di eventi congiunti.

Possiamo infine notare come, nella soluzione proposta, il Manager non faccia altro che andare a ridirigere gli eventi generati dal monitor sul gestore della qualità del servizio, in modo da avere indicazioni precise sui motivi alla base di un problema rilevato.

Anche in questo caso è conveniente costruire una gerarchia di eventi che specializzino una generica classe che definisce un problema sulla banda rilevata tramite SNMP. (fig. 5.6)

5.3.3 Metodi di controllo

Il metodo della classe monitor incaricato di effettuare i controllo è solamente uno, cioè il metodo *controlSnmp()*. L'invocazione di questo metodo scatena un controllo sul MIB e provoca, se necessario, la generazione di eventi di problema, seguendo la gerarchia stabilita.

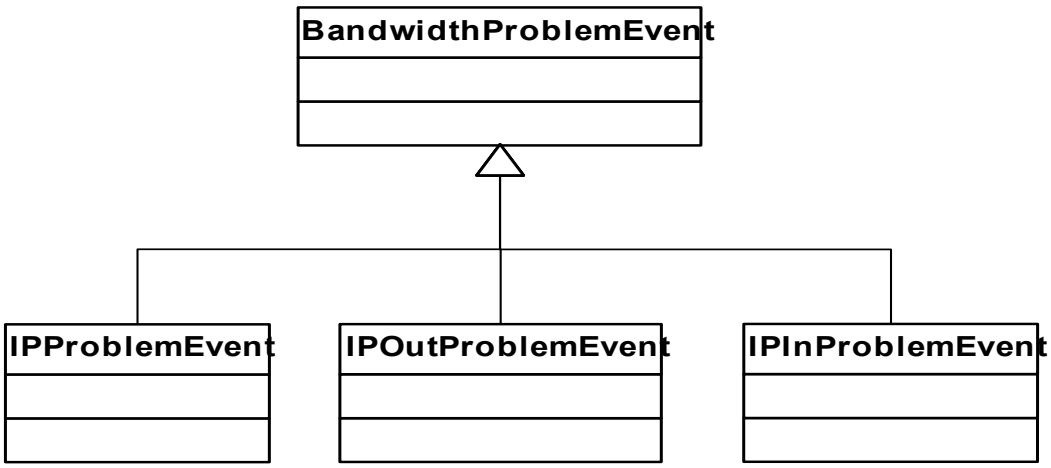

Figura 5.6: Gerarchia degli Eventi di problema legati ad SNMP

La prima azione di questo metodo è l'invocazione al metodo omonimo della classe *SnmpGet*, il cui risultato, una istanza della classe *NetworkState*, ha al suo interno quelli che sono i valori attuali presenti nel MIB dei nodi che si è deciso di scegliere.

Una volta ottenuto il resoconto della situazione attuale la si va a confrontare con quella rilevata all'invocazione precedente. Questo si può fare solamente utilizzando una variabile *NetworkState* interna che va aggiornata alla fine di ogni controllo. Alla fine di questa comparazione si sono ottenuti i dati netti che si riferiscono all'intervallo di tempo che è intercorso tra i due successivi rilevamenti e si possono formulare delle valutazioni su quello che stato il comportamento delle risorse in questo intervallo di tempo.

Naturalmente si seguirà sempre la stessa politica di monitoraggio, stabilendo le due soglie di controllo per ciascuna grandezza e considerando i consecutivi livelli di warning.

5.4 Implementazione e testing

5.4.1 Implementazione

Molte delle scelte implementative che si sono fatte sono risultate praticamente obbligate e hanno avuto una grande influenza sulla fase di progettazione. Innanzitutto si è dovuto tenere conto del fatto che dovevamo andare ad integrare un sistema già esistente che aveva un preciso linguaggio di programmazione, ossia Java. Inoltre anche la scelta delle tecnologie da utilizzare si è dimostrata, in un certo modo, pilotata. Infatti già in MUM era utilizzato il JMF e sfruttare le sue funzionalità è apparso quasi naturale.

In particolare è risultato molto utile per la nostra realizzazione andare ad utilizzare il package che si riferisce al traffico RTCP, ossia il package *javax.media.rtp.rtcp*. All'interno di questo package si sono sfruttate soprattutto le funzionalità rese disponibili dalla classe *Feedback*. Tale classe realizza l'oggetto, citato in precedenza, che rappresentava un generico feedback ricavabile da RTP e che raggruppava tutti i dati che si riferivano alla sezione *Report Block* dei pacchetti *Sender Report* e *Receiver Report*. Da questa classe, realizzata in JMF appunto perché questo è RTP compliant, si sono potuti ottenere i valori correnti di jitter e fraction lost.

Un discorso diverso va fatto per quello che riguarda la tecnologia usata per realizzare il controllo sulle risorse locali tramite SNMP. Si deve creare all'interno del monitor un thread, che faremo diventare un demone, che ha proprio il compito di effettuare il monitoring delle risorse. Questo demone riposa per un intervallo di tempo prestabilito e quindi scatena i vari controlli sulla banda trasmissiva. Andando ad agire su questo intervallo possiamo forzare i controlli e, scegliendolo opportunamente, trovare il miglior compromesso tra frequenza e precisione del monitoraggio ed intrusività del controllo. Ovviamente il monitor verrà creato una volta solamente insieme al manager, in modo da garantire un minor carico sulle risorse del sistema.

Inoltre nell'implementazione si fa largo utilizzo delle SNMP API sviluppate da AdventNet e descritte nel capitolo dedicato alle tecnologie. In particolare si è scelto di far ricorso alle API di basso livello, in modo da caricare il minor numero di classi possibile e da avere un basso livello di astrazione, per lavorare più direttamente con i dati che ci interessano. In quest'ottica si sono dovute scegliere le classi che meglio delle altre potevano svolgere il compito di interrogazione e restituzione di dati dal

MIB, ossia quelle classi tramite le quali si poteva implementare la primitiva di GET propria del protocollo SNMP.

La classe che implementa il monitor e la classe *SnmpGet* sono in rapporto uno-a-uno tra di loro. Infatti per funzionare le API di basso livello richiedono la presenza di due classi particolari *SnmpAPI* ed *SnmpSession*, che al loro interno fanno partire due thread. Per come la libreria di AdventNet è strutturata senza questi due thread non possono essere istanziate quelle classi che sono necessarie per ottenere i dati dal MIB. Per cercare di evitare inutile spreco di risorse, facendo partire e fermando questi thread ad ogni ciclo di controllo, li si usa come fossero demoni, facendoli partire nel costruttore della classe e riutilizzandoli sempre senza chiuderli. Per questo motivo c'è una sola istanza di *SnmpGet* per ogni *Bandwidth Monitor*, perché, alla luce di quanto appena detto, non avrebbe senso creare una nuova istanza di questa classe (e quindi anche due nuovi thread) ad ogni richiesta di controllo.

Nella realizzazione sono stati coinvolti solamente due package di MUM, *repository.qosManagement* e *resourceMngService*, oltre a qualche altra classe all'interno della quale sono stati fatti piccoli aggiustamenti.

5.4.2 Testing

In seguito all'implementazione si è proceduto ad una accurata fase di testing con lo scopo di andare a definire tutti i valori delle soglie relative a ciascuna grandezza da osservare. In questa fase si sono anche riviste ed aggiustate alcune politiche di controllo che erano risultate poco robuste. Ad esempio si è potuto notare che i valori di jitter e fraction lost attraversano un transitorio iniziale entro il quale assumono valori non esatti e notevolmente più alti delle soglie stabilite. Si è dovuto procedere quindi a modificare il controllo su queste due grandezze in maniera da ovviare a questo problema. Dobbiamo poi sottolineare come non si sia potuta verificare la validità del controllo tramite SNMP, dal momento che questo si basa sulla percentuale di pacchetti persi ed in una rete locale la situazione di perdita di un pacchetto si verifica molto di rado.

Oltre ai test legati alla scelta delle soglie di controllo, si sono svolti altri studi sperimentali con lo scopo di controllare l'intrusività del nostro modulo di monitoraggio. A questo fine, si è proceduti a fare funzionare il sistema MUM in

diverse condizioni per poi analizzare l'incidenza sull'utilizzo della CPU del nostro modulo.

5.4.2.1 Configurazione dei test

Due computer sono stati coinvolti nell'esecuzione dei test. Essi erano così equipaggiati:

PC1: portatile Packard-Bell
-processore AMD Duron 1200 MHz, 256 MB RAM
-sistema operativo Windows XP Home

PC2: fisso
-processore AMD Atlon 1400 MHz, 256 MB RAM
-sistema operativo Windows XP Professional

Rete: i due computer sono stati poi collegati da una rete LAN Ethernet da 10 Mbps

Sul PC2 si va ad istanziare il nodo radice, ossia quello che avrà la funzione di server, mentre sul PC1 si sono istanziati due place con il ruolo, rispettivamente, di proxy e di client.

Si sono messe a confronto diverse situazioni, che sono sembrate più significative. In particolare si è pensato di confrontare il sistema a riposo con e senza il controllo tramite il protocollo SNMP. Quindi si è fatta partire una presentazione su un place avente un proxy, in modo da poter controllare anche l'intrusività di questo controllo sulla CPU.

5.4.2.2 Risultati dei test

I primi test sono risultati molto incoraggianti. Se si va a considerare il controllo SNMP si può facilmente vedere (figura 5.7) come il sistema con o senza questo tipo di controllo abbia un comportamento per lo più indistinguibile.

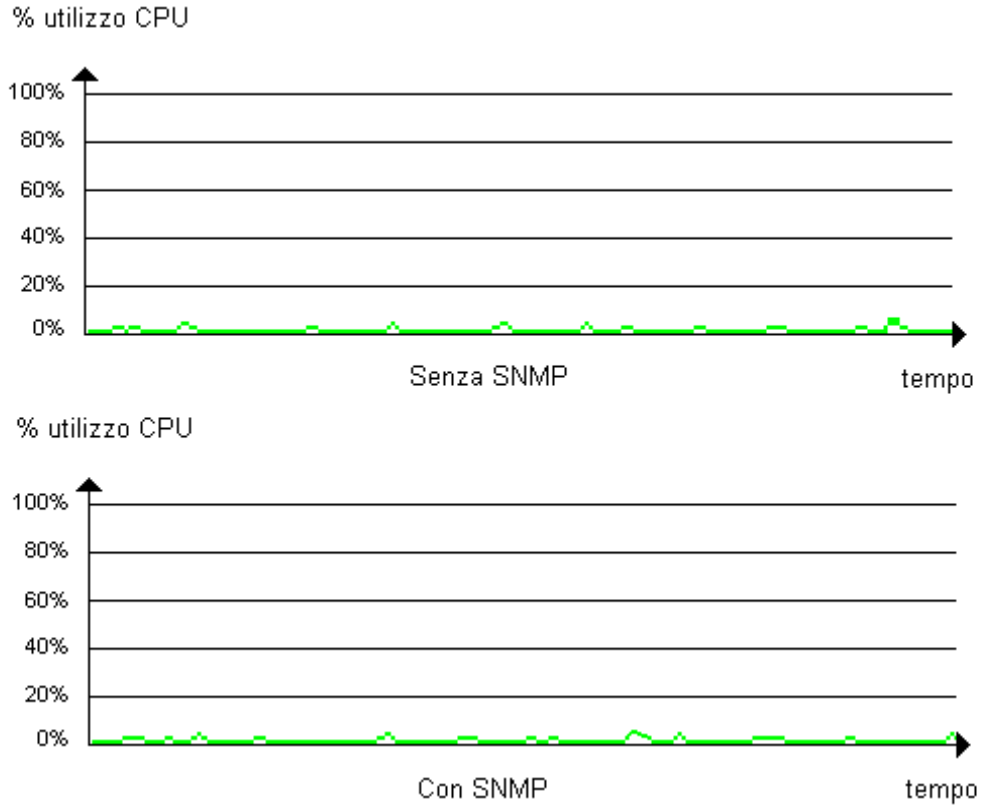

Figura 5.7: Il comportamento del sistema con o senza il controllo SNMP

Per quanto riguarda il controllo tramite RTCP, possiamo asserire che la condizione è molto simile alla precedente. Infatti, come possiamo vedere in figura 5.8, i comportamenti sono molto simili. Si può vedere come questo grafico presenti dei picchi, i quali hanno un andamento instabile e possono essere anche molto pronunciati. Si è potuto notare come questi picchi si verifichino in concomitanza agli eventi di arrivo di un pacchetto RTCP di Sender Report, ma non siano in diretta relazione con il nostro controllo, dal momento che si ripresentano anche quando questi è disabilitato.

Figura 5.8: Il comportamento del sistema con o senza il controllo RTP

Possiamo concludere che il nostro modulo ha rispettato, secondo i primi risultati qui analizzati, l'obiettivo di essere il meno intrusivo possibile sulla risorsa della CPU. Ovviamente si sono stabilite delle soglie che sono sembrate appropriate, ma nulla vieta che un futuro utilizzatore possa decidere di scegliere, a proprio piacimento, soglie completamente diverse, a seconda dei suoi bisogni. Per quanto riguarda il discorso relativo ai picchi dovuti ai pacchetti di Sender Report, ci si ripromette di andare a studiarne le cause, in modo da avere la possibilità di eliminarli o quantomeno di riuscire a ridurre il più possibile la loro ampiezza.

5.5 Conclusioni

In questo capitolo si è sviluppato il progetto di un modulo di controllo e monitoraggio per servizi multimediali, e si sono approfondite le tematiche relative a ciascun aspetto. Si sono inseriti all'interno della trattazione anche dettagli implementativi, dal momento che la scelta relativa al linguaggio era obbligata e le

scelte fatte riguardo le tecnologie da utilizzare influenzavano fortemente la fase di progetto.

Conclusioni

Nel corso del nostro lavoro abbiamo toccati diverse tematiche legate al controllo ed al monitoraggio delle risorse per una applicazione multimediale.

Per prima cosa si sono studiati approfonditamente i protocolli RTP/RTCP ed SNMP tramite i quali si riesce ad ottenere interessanti informazioni sul traffico sulla rete e quindi anche sulla banda trasmissiva. Seguendo questa divisione dualistica si è disaccoppiato il problema e si è scelto di sfruttare le peculiarità di questi due protocolli andando a realizzare il nostro modulo di monitoraggio in due sotto-moduli, uno incentrato su RTP/RTCP e l'altro su SNMP, con caratteristiche e compito diversi e ben precisi, con lo scopo di realizzare una soluzione più flessibile e portabile. Si è adottata per entrambi i componenti una stessa politica di controllo. Tale politica è mutuata dalla teoria delle carte di controllo e permette, tramite l'utilizzo di diverse soglie ed il controllo dei livelli di warning, di realizzare un controllo complesso e poco intrusivo. Il vantaggio di questo approccio sta nelle sue caratteristiche di previsione, dal momento che si riescono a percepire anche andamenti critici che preludono ad una situazione di problema che ancora si deve palesare.

Si è poi passati ad una fase di prova e testing del componente per capire come il controllo implementato interagisca con il sistema, quanto sia valido e quanto vada ad influire, in termini di intrusività, sulle risorse del sistema stesso. Da queste prime prove si può rilevare un comportamento più che accettabile del nostro componente, sia in termini di precisione del controllo che in termini di sua intrusività al livello delle risorse, in particolare sulla CPU.

Dobbiamo ora sottolineare come il nostro lavoro si sia andato ad inserire in un contesto molto ampio, ovvero la problematica di gestione delle risorse necessarie ad un flusso multimediale, e come siano stati scelti con precisione i limiti del nostro campo di azione. In definitiva si possono già indicare alcuni aspetti che dovrebbero essere oggetto di studi futuri. In particolare si potrebbe pensare di rendere molto più modulare la nostra soluzione, realizzando un componente capace di inizializzare i parametri interni ai nostri due sotto-moduli, stabilendo soglie e frequenze dei controlli, a seconda delle esigenze dell'utilizzatore finale. Inoltre meriterebbe un

accurato studio la problematica legata al controllo sull'utilizzo della CPU, in modo da integrare il nostro componente con un altro affine, che implementi anche per questa risorsa un controllo complesso.

Bibliografia

[OSI] Basic Reference Model of Open Distributed Processing, International Standard Organization, 1992.

[RFC1889] RTP: A Transport Protocol for Real-Time Applications. RFC1889, si veda http://www.ietf.org/rfc/rfc1889.txt

[SOMA] Bellavista P., Corradi A., Stefanelli C., "Mobile Agent Middleware to Support Mobile Computing", IEEE Computer, March 2001

[MUM] L.Foschini, tesi di laurea, 2003

[JMFPG] Java Media Framework, Programmers Guide: http://java.sun.com/products/java-media/jmf/2.1.1/specdownload.html

[JMFH] Java Media Framework home page: http://java.sun.com/products/javamedia/jmf/

[Adv] AdventNet: java SNMP API. http://www.adventnet.com/ .

[Snmp] A.Calarco, "Il protocollo SNMP" http://www.aipa.it/attivita[2/formazione[6/corsi[2/materiali/Reti%20di%20Calcolatori/welcome.htm

[Rete] A. Calarco, "Problematiche di gestione della rete" http://www.aipa.it/attivita[2/formazione[6/corsi[2/materiali/Reti%20di%20Calcolatori/welcome.htm