PARIPARI: MULTICAST

RELATORE: Ch.mo Prof. Enoch Peserico Stecchini Negri De Salvi
CORRELATORE: Ing. Paolo Bertasi
LAUREANDO: Marco Malesardi

Corso di laurea in Ingegneria Informatica

A.A. 2010-2011

UNIVERSITÀ DEGLI STUDI DI PADOVA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

TESI DI LAUREA

PARIPARI: MULTICAST

RELATORE: Prof. Enoch Peserico Stecchini Negri De Salvi

CORRELATORE: Ing. Paolo Bertasi

LAUREANDO: Marco Malesardi

A.A. 2010-2011

Sommario

In questa tesi di laurea viene descritta l'architettura e l'implementazione del modulo *Multicast* sviluppato per la rete *PariPari*. Il suddetto modulo ha lo scopo di fornire una connessione di tipo *multicast* ai plugin applicativi che ne richiedono l'utilizzo.

Nel primo capitolo di questo documento viene presentato lo scopo e la struttura del progetto PariPari. Successivamente, nel secondo capitolo, sarà descritto il modulo Multicast nei suoi aspetti progettuali. Nel terzo capitolo sarà quindi approfondita l'implementazione di tale modulo mediante un'architettura centralizzata, denominata *SimpleConference*.

Indice

Introduzione

Nel corso dell'ultimo decennio la rete Internet ha subito una straordinaria espansione. L'evoluzione della rete globale non ha riguardato solo la crescita del numero di utenti, ma anche un aumento della varietà dei servizi offerti. Nuovi servizi quali il *file sharing*, il *VoIP* e, più in generale, la distribuzione di contenuti multimediali sono infatti fioriti accanto a quelli tradizionali del *World Wide Web*, della posta elettronica e dell'instant messaging. In questo scenario, la tipica architettura *client-server* è stata affiancata da una diversa organizzazione della rete, in cui i nodi hanno un ruolo paritario e possono al contempo fornire o usufruire di servizi. Questo tipo di architettura è denominata *peer-to-peer*. I vantaggi offerti da una rete così strutturata sono:

- la condivisione delle risorse, in quanto i nodi mettono le proprie (banda, spazio di memorizzazione e potenza di calcolo) a disposizione dell'intera rete;

- robustezza, poiché i servizi vengono distribuiti su diversi nodi e di conseguenza la loro disponibilità non dipende da un singolo server.

Il progetto PariPari si sviluppa in questo ambito, con l'obiettivo di creare una rete peer-to-peer serverless multifunzionale. La rete implementa tutti i più comuni servizi Internet: file sharing, file hosting, database, DNS, web hosting, instant messaging, chat IRC, NTP, posta elettronica e VoIP. Inoltre, possono essere create nuove applicazioni utilizzando le API messe a disposizione.

Tra le varie applicazioni elencate, alcune di esse necessitano di una connessione che distribuisca la stessa informazione a più utenti simultaneamente. Si pensi, ad esempio, al caso in cui tre o più persone tengono una conferenza telefonica via VoIP: il flusso audio generato dalla voce di una persona va trasmesso a tutti i partecipanti alla conferenza. Questo tipo di connessione tra uno specifico gruppo di utenti è chiamata *multicast*. Essa si distingue dalla più comune connessione *unicast*, in cui un pacchetto è inviato ad un singolo destinatario, e

dal *broadcast*, in cui un pacchetto è trasmesso a tutti i nodi della rete. La maggior parte dei dispositivi che costituiscono l'attuale rete globale non supportano la funzionalità di *routing* multicast. Per questo motivo le applicazioni devono gestire esplicitamente tale tipo di connessione.

In PariPari, l'implementazione del servizio per stabilire connessioni di tipo multicast è demandata all'omonimo modulo. Grazie ad esso, i *plugin* applicativi possono facilmente utilizzare una comunicazione tra gruppi di utenti, lasciando a *Multicast* il compito di gestire i problemi inerenti la connettività.

Capitolo 1

PariPari

In questo primo capitolo verrà introdotto il progetto PariPari. Partendo dall'idea alla sua base, si proseguirà presentando l'architettura della sua implementazione e i componenti che lo costituiscono. L'ultima sezione del presente capitolo è dedicata all'organizzazione e ai metodi di lavoro adottati dal gruppo che sta attualmente sviluppando l'applicazione. Attraverso questa esposizione sarà possibile comprendere l'ambiente in cui è stato sviluppato il modulo Multicast, oggetto del presente lavoro di tesi.

1.1 L'obiettivo del progetto PariPari

Il progetto PariPari, sviluppato nel Dipartimento di Ingegneria dell'Informazione dell'Università di Padova, è stato concepito con l'obiettivo di creare una rete peer-to-peer serverless che[1]:

- sia multifunzionale;
- utilizzi un sistema intelligente di crediti;
- garantisca l'anonimato ai propri utenti;
- renda i propri servizi accessibili anche dall'esterno della rete.

La rete PariPari è **serverless** e quindi completamente decentralizzata. Questa organizzazione ha il pregio di rendere la rete più robusta, dato che la sua accessibilità non dipende da nodi specifici. Inoltre, la scalabilità della rete è maggiore rispetto a quella di un'organizzazione centralizzata. Il principale inconveniente

[1] `http://paripari.it/mediawiki/index.php/PariPari_en#Description`

Figura 1.1: Il logo di PariPari

riguarda la modalità con cui reperire una risorsa all'interno della rete. Per risolvere tale problema PariPari si basa sul protocollo *Kedemlia*[2], che sfrutta una *Distributed Hash Table* per indicizzare e ricercare *host* e risorse all'interno della rete.

Volendo essere **multifunzionale**, PariPari fornisce diversi servizi, da quelli tipici delle reti peer-to-peer (file sharing, VoIP, ...) a quelli più comuni all'architettura client-server (file hosting, database, DNS, web hosting, instant messaging, chat IRC, NTP, posta elettronica). Questi ultimi, invece della consueta realizzazione centralizzata, sono implementati in maniera distribuita all'interno della rete, con i risvolti positivi di cui detto sopra.

L'accorpamento di diversi servizi all'interno di un'unica applicazione ha il vantaggio di consentire una gestione più equa delle risorse in base alle diverse esigenze. Comunemente, infatti, si è abituati ad usare programmi differenti per scopi differenti. Si pensi, per esempio, al caso in cui si riceve una chiamata su VoIP mentre è in corso uno scambio di dati tramite file sharing. Dal punto di vista dell'utente le due attività hanno esigenze differenti. Infatti, se un leggero ritardo nella trasmissione di un file non desta preoccupazioni, lo stesso non si può dire riguardo una chiamata telefonica. Anche un leggero ritardo nella consegna di due pacchetti audio consecutivi è percepita dall'orecchio umano e sopra una certa soglia risulta essere fastidioso per l'utente[3]. A meno che il sistema non possieda

[2]P. Maymounkov, D. Mazières: *"Kademlia: A Peer-to-peer Information System Based on the XOR Metric"*, `http://kademlia.scs.cs.nyu.edu`

[3]il ritardo massimo comunemente accettato per una comunicazione vocale di buona qualità è di 250ms, cfr. *"Understanding Delay in Packet Voice Networks"*, Cisco Technology White Paper, `http://www.cisco.com/en/US/tech/tk652/tk698/technologies_white_paper09186a00800a8993.shtml`

un qualche meccanismo di *QoS*[4], il programma di file sharing potrà interferire con la banda necessaria al programma VoIP, degradando così le prestazioni di quest'ultimo. In PariPari, essendo i vari servizi controllati dalla medesima applicazione, risulta relativamente semplice allocare le risorse in base alle diverse esigenze degli stessi.

PariPari si propone di utilizzare un sistemi di **crediti** intelligente per la monetizzazione delle risorse. L'utilizzo di sistemi di crediti è molto comune nelle reti peer-to-peer. Il loro scopo è quello di incoraggiare la partecipazione ed al contempo scoraggiare comportamenti parassitici. Senza voler approfondire l'argomento, basti sapere che all'interno della rete la fruizione di una risorsa ha un ben determinato costo. In questo modo gli utilizzatori richiederanno lo sfruttamento delle risorse solo per il tempo e nella quantità a loro necessaria. Per guadagnare crediti un utente ha la possibilità di mettere a disposizione le risorse della propria macchina. Così facendo viene incrementata la capacità della rete. Quello dei crediti è quindi un sistema per regolare i rapporti di collaborazione tra i nodi.

Un altro punto di forza di PariPari è la possibilità di consentire l'**anonimato** ai nodi che lo richiedono. Questa caratteristica si riflette nell'impossibilità di rintracciare l'indirizzo IP del mittente o del destinatario da parte di entità che intercettano la comunicazione. Questa funzionalità è implementata in PariPari usando la tecnica dell'*onion routing* che, attraverso successive cifrature del messaggio, garantisce la segretezza della trasmissione.

Un'importante caratteristica di PariPari, volta a favorirne l'inserimento e la diffusione sul mercato, è la **fruibilità** dei suoi servizi anche da parte di nodi esterni alla rete. Ad esempio, una pagina web, ospitata da un web server distribuito in esecuzione all'interno della rete PariPari, risulta accessibile da qualsiasi postazione connessa a Internet tramite un comune browser. In questo modo la rete può essere sfruttata anche da macchine che non ne fanno parte.

Con quest'ultimo punto possiamo ritenere conclusa la descrizione della caratteristiche alla base del progetto. Per un approfondimento sui concetti fondamento di PariPari si veda [1].

[4]Quality of Service.

1.2 L'architettura dell'applicativo

Passiamo ora ad analizzare la struttura dei nodi che costituiscono la rete.

Per semplificare la gestione della multifunzionalità, i client PariPari hanno una struttura modulare. Questo significa che il programma è suddiviso in componenti non autonomi che interagiscono tra loro. Questi componenti sono detti *plugin* e ciascuno di essi implementa una specifica funzionalità della rete. La modularità consente uno sviluppo separato dei diversi plugin, riducendo al minimo le ripercussioni che la creazione o la modifica di un componente provoca sugli altri. Inoltre, grazie a questa strategia architetturale, un modulo può considerare gli altri come delle *black box* e accedere ai loro servizi semplicemente interagendo con la loro interfaccia. Questo consente anche lo sviluppo di plugin da parte di terze parti.

La comunicazione tra i vari plugin avviene tramite lo scambio di messaggi. Questo operazione, però, non avviene in maniera diretta, bensì attraverso la mediazione di un'entità di controllo, chiamata **Core**. Il Core è il nocciolo centrale del client PariPari ed è un modulo particolare che funge da collante tra i vari plugin. Esso si occupa di caricare i moduli richiesti a *runtime*, gestire le comunicazioni tra i vari componenti e garantire sicurezza. Inoltre, il Core provvede ad un'equa distribuzione delle risorse, incrementando la *user experience*.
La figura 1.2 mostra l'interazione tra i vari plugin all'interno di uno stesso client PariPari.

Oltre al Core, esistono altri plugin con funzione particolare: i gestori di risorse. Questi moduli forniscono l'accesso alle risorse della macchina su cui è in esecuzione il client. Esiste quindi una distinzione tra i plugin che offrono supporto per l'utilizzo della rete e i plugin con finali più applicative. I primi appartengono a quella che è chiamata *cerchia interna* dei plugin, mentre i secondi appartengono alla *cerchia esterna*. Oltre al Core, i moduli facenti parte della cerchia interna sono:

DHT : implementa il protocollo per l'indicizzazione e la ricerca degli host e delle risorse distribuite nella rete; come già riferito in 1.1, l'algoritmo si basa essenzialmente su Kademlia;

Storage : fornisce l'accesso alle memoria di massa; la presenza di questo modulo è sostanzialmente dettata da esigenze di sicurezza: controllando direttamente la scrittura su disco si vuole evitare danneggiamenti dello stesso da parte di plugin malevoli o difettosi;

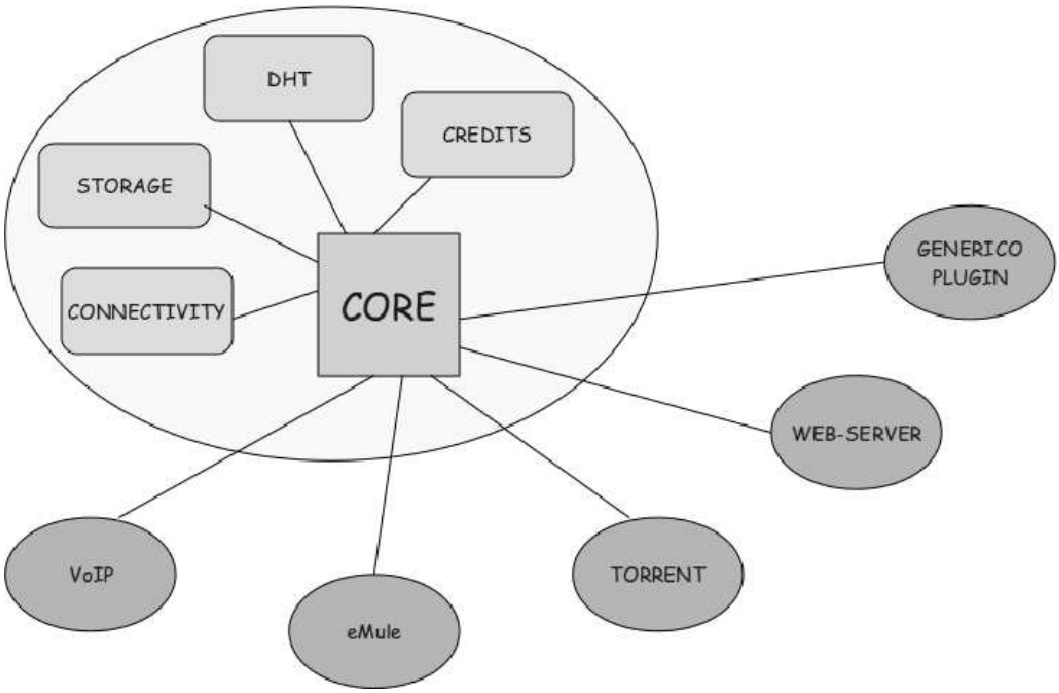

Figura 1.2: Struttura di un client PariPari

Connectivity : fornisce l'accesso alle risorse di rete; i moduli che vogliono accedere alla rete devono utilizzare i metodi messi a disposizione da questo plugin; oltre che per ragioni di sicurezza analoghe a quelle di Storage, l'esistenza di Connectivity è dettata da esigenze di controllo: le comunicazioni su rete sono limitate in banda utilizzando un'algoritmo Token Buket[5] e ciò consente una ripartizione della banda disponibile secondo le necessità dei plugin;

Crediti : implementa il sistema di crediti, il cui scopo è stato illustrato in 1.1.

I plugin della cerchia esterna realizzano servizi applicativi, sfruttando il supporto fornito da quelli della cerchia interna. L'innegabile vantaggio di questo approccio è la trasparenza di cui può godere lo sviluppo dei servizi applicativi. Questi, infatti, possono essere implementati senza doversi preoccupare della struttura di rete sottostante.

Abbiamo così completato la definizione delle caratteristiche di PariPari anche dal punto di vista architetturale. L'ultima parte di questo capitolo è dedicata all'approfondimento del gruppo che sviluppa l'applicazione e alle strategie di lavoro che esso adotta.

[5] `http://en.wikipedia.org/wiki/Token_bucket`

1.3 Il gruppo PariPari

Questa sezione ha lo scopo di illustrare la composizione del gruppo che attualmente sviluppa PariPari. Sarà qui presentata l'organizzazione del personale e i metodi di lavoro che esso adotta. Verranno infine esposte le ragioni che hanno guidato la scelta del linguaggio di programmazione da utilizzare: Java.

Il gruppo di sviluppo di PariPari è composto attualmente da circa cinquanta studenti del Dipartimento di Ingegneria dell'Informazione dell'Università di Padova, che lavorano al progetto per la redazione della propria tesi di laurea. Data questa numerosità, si sono dovute adottare misure specifiche per l'organizzazione e la gestione del lavoro.

Ogni studente è assegnato e contribuisce allo sviluppo di un preciso modulo. Si forma quindi una suddivisione delle risorse umane in diversi gruppi, corrispondenti ai vari plugin presenti in PariPari.

La figura di riferimento per l'intero progetto è quella dell'Architetto, che mantiene una visione complessiva dello stato globale sovrintendendo al progresso dell'applicazione. La necessità di cooperazione tra i diversi plugin ha portato all'esigenza di eleggere un coordinatore all'interno di ciascun gruppo per relazionarsi con gli altri. Inoltre, egli controlla e gestisce l'avanzamento del lavoro nel modulo cui afferisce. La recente crescita del gruppo ha portato ad un'evoluzione della gerarchia interna. Sono stati aggregati gruppi di plugin che offrono servizi affini. Per esempio sono stati riuniti sotto uno stesso *team leader* i moduli Mulo e Torrent che operano entrambi sul file sharing. Queste aggregazioni vogliono favorire la circolazione del *know-how* riguardo alla soluzione di problemi comuni.

La complessità del progetto ha richiesto l'adozione di tecniche specifiche per facilitare la cooperazione tra i diversi sviluppatori. In questo senso, un passo fondamentale è stato segnato dall'impiego di una particolare metodologia agile[6] denominata *eXtreme programming*. Questo metodo di lavoro è stato descritto per la prima volta in [7]. La presentazione di tutte le regole su cui questa tecnica si fonda vanno al di là degli scopi di questo documento. Piuttosto, è importante sottolineare le principali ripercussioni che l'attuazione di questa metodologia ha avuto nel processo lavorativo in PariPari, e cioè:

[6]per metodologia agile si intende un particolare metodo per lo sviluppo del software che coinvolge quanto più possibile il committente, ottenendo in tal modo un'elevata reattività alle sue richieste.

- standardizzazione del codice e della documentazione;
- riunioni periodiche (almeno ogni quindici giorni) per la verifica e la pianificazione delle attività;
- progettazione con il cliente: in particolare, la definizione dell'interfaccia di un modulo avviene seguendo le indicazioni fornite dai futuri utilizzatori;
- proprietà del codice collettiva: contribuisce alla stesura chiunque sia coinvolto nel progetto;
- rilasci frequenti di versioni funzionanti del programma e integrazione continua dei cambiamenti del codice;
- *test-driven development*, per verificare la correttezza delle funzioni implementate;
- *pair programming*: coppie di programmatori al lavoro sulla medesima sezione di codice, suddividendosi i compiti di stesura e revisione.

La cooperazione tra i membri è stata favorita anche dall'adozione di strumenti comuni e dall'utilizzo di un sistema di controllo versione quale Subversion[7]. Particolare rilevanza è stata inoltre ricoperta dall'uso di un sistema di *bug-tracking* quale Bugzilla[8], per la segnalazione e la gestione di eventuali *bug*[9].

Il linguaggio di programmazione utilizzato per scrivere l'applicazione PariPari è Java. Questa scelta è stata dettata principalmente dalla confidenza che gli studenti del Dipartimento hanno con questo linguaggio. Ciò ha consentito di creare rapidamente un'ampia base di sviluppatori. Java possiede inoltre l'assai desiderabile proprietà della portabilità. Infatti, una volta compilato, il codice può essere eseguito da qualsiasi macchina che abbia una *Java Virtual Machine* installata. Un ulteriore vantaggio da non sottovalutare è la possibilità di avviare l'applicazione direttamente da un browser impiegando *Java Web Start*[10]. Questa tecnologia rende trasparenti all'utente le fasi di scaricamento, installazione ed aggiornamento del software. Per contro, l'interpretazione del codice a runtime da parte della Java Virtual Machine introduce un inevitabile overhead rispetto ai linguaggi compilati. Ne deriva un degrado di prestazioni, specie per le operazioni

[7] http://subversion.apache.org/

[8] per approfondimenti si veda [3, cap. 4]

[9] un bug rappresenta non solo errori e problemi ma, in in senso più ampio, qualsiasi cosa possa essere modificata dagli sviluppatori di un progetto, e quindi possibili migliorie ed aggiunte.

[10] http://www.java.com/en/download/faq/java_webstart.xml

più costose dal punto di vista computazionale. I problemi sorgono, ad esempio, quando si utilizza la crittografia per garantire anonimato. Fortunatamente, gli stream di byte da cifrare sono di dimensioni ridotte e quindi il rallentamento del sistema non è eccessivo.

Conclusioni Si conclude così questo primo capitolo in cui è stata fatta una panoramica completa sul progetto PariPari, dalle sue caratteristiche concettuali, all'architettura del client, per finire con l'organizzazione e i metodi di lavoro del gruppo di sviluppo. Grazie a questa descrizione è ora più definito l'ambiente in cui è stato sviluppato il modulo Multicast, oggetto del presente lavoro di tesi.

Capitolo 2

Il modulo Multicast

In questo capitolo sarà approfondito il modulo *Multicast* presente in PariPari. La presenza di questo plugin deriva dalla volontà di fornire un servizio di comunicazione tra gruppi di utenti agli sviluppatori che necessitano di connessioni uno-a-molti o molti-a-molti. Questo particolare tipo di connessione, detta *multicast*, sarà introdotta nella prima parte del capitolo. Saranno così illustrate le ragioni che hanno portato alla creazione di Multicast in PariPari. In seguito, verranno esposte le strategie adottate da tale modulo per risolvere il problema del multicasting nella rete.

2.1 Il problema del multicasting

Nell'ambito delle reti di calcolatori, con il termine multicast si intende la consegna di un messaggio ad un gruppo di utenti simultaneamente in una singola trasmissione. Questo tipo di comunicazione si distingue dalla tipica connessione punto-a-punto, detta *unicast*, e dalla trasmissione di un messaggio a tutti i nodi della rete, detta *broadcast*. La figura 2.1 illustra questi tre diversi tipi di comunicazione.

La necessità di instaurare una connessione di tipo multicast sorge per applicazioni che coinvolgono gruppi di utenti. Esempi al riguardo sono le conferenze telefoniche mediante VoIP, chat tra tre o più persone, servizi di *streaming* multimediale e IPTV. Si noti che le prime due applicazioni citate generano una comunicazione molti-a-molti, in cui cioè tutti i membri del gruppo sono al contempo sorgenti e destinazioni di informazione, mentre le altre due sono esempi di comunicazione uno-a-molti, in cui il flusso dati ha origine da una singola sorgente.

Il problema di instradare traffico di tipo multicast risulta più complesso rispetto al *routing* di pacchetti verso una singola destinazione. Si deve infatti costruire

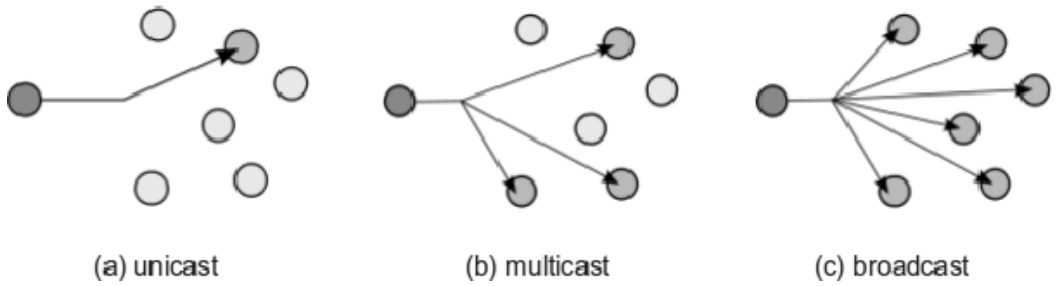

Figura 2.1: Schemi di instradamento pacchetti.

all'interno della rete un albero per distribuire l'informazione in maniera efficiente. I nodi della rete devono replicare i pacchetti per raggiungere i vari destinatari solo dove necessario, ottimizzando così il consumo di banda. L'instaurazione di una connessione multicast pone inoltre il problema di creare e riconoscere un gruppo ed i suoi membri.

La *IETF*[1] ha provveduto ad estendere il protocollo IPv4 originale al fine di introdurre il supporto al multicasting [8] [9]. Purtroppo però, è possibile sfruttare questa funzionalità solo in reti locali. Nella rete Internet globale la presenza di firewall, NAT[2] e proxy, unita alla scarsa diffusione di router che supportino il multicast[3] rende l'IP multicast del protocollo non sfruttabile nella pratica. Per questo motivo, le comunicazioni multicast all'interno della rete PariPari devono essere gestite dagli sviluppatori a livello applicativo. Ciò ha portato all'idea di creare un apposito modulo che fornisca agli altri plugin una connettività di tipo uno-a-molti o molti-a-molti.

Lo spirito del modulo Multicast è implicito nella descrizione che David Clark, uno dei padri fondatori di Internet, ha dato del multicast:

> *"You put packets in at one end, and the network conspires to deliver them to anyone who asks."*—David Clark

Multicast infatti si occupa di trasmettere il flusso dati, generato da un altro plugin, a tutti i destinatari. Il suo scopo è quello di fornire agli sviluppatori un'interfaccia per instaurare comunicazioni multicast in maniera del tutto trasparente alla rete sottostante e ai meccanismi di trasmissione. Il modulo Multicast svolge quindi, all'interno di PariPari, una funzione di supporto per gli sviluppatori, piuttosto che offrire una funzionalità all'utente finale. Dal punto di vista dello stack protocollare, esso si pone come un livello intermedio tra quello

[1] Internet Engineering Task Force
[2] Network Address Translation
[3] tali dispositivi sono chiamati *M-router*.

di trasporto e le applicazioni, costruendo una rete di *overlay*[4] che mette in comunicazione i nodi membri di uno stesso gruppo multicast.

Per risolvere il problema di creare una rete di overlay tra i membri di uno stesso gruppo multicast, si è cercato di organizzare i nodi in una struttura efficiente in termini di consumi di banda e tempi di distribuzione dei contenuti. Di seguito saranno esposte le soluzioni pensate per implementare il servizio Multicast in PariPari. Si tratta in particolare di una prima architettura, chiamata *SimpleConference*, concettualmente semplice ma limitata nelle prestazioni, e di un suo raffinamento per correggerne i difetti. Questa seconda organizzazione è chiamata *AdvancedConference*. Entrambe le architetture presentate sono orientate alla comunicazione molti-a-molti. D'altro canto, la comunicazione uno-a-molti può essere considerata un caso particolare di quella molti-a-molti e quindi le due architetture possono essere opportunamente utilizzate anche per distribuire traffico multicast generato da una singola sorgente.

2.2 Architettura centralizzata

La prima architettura presa in considerazione è quella denominata SimpleConference. Si tratta di un'organizzazione centralizzata, in cui un nodo eletto agisce da "server" e si occupa di gestire la connessione multicast. Tutti gli altri nodi membri del gruppo fungono da "client". Il nodo server è in comunicazione diretta con i client, riceve i loro singoli flussi dati e invia loro il traffico multicast. Tale organizzazione è illustrata in figura 2.2. Nella sezione 2.4 sarà approfondito il problema riguardante la scelta del nodo server.

In generale, l'informazione da distribuire in multicasting è una specifica combinazione dei flussi dati generati dai singoli membri del gruppo e, in SimpleConference, tale operazione viene effettuata dal nodo server. D'altro canto, la modalità con cui eseguire questo *mixing* dipende strettamente dall'applicazione che utilizza Multicast. Infatti, se ad esempio una audio-conferenza richiede la sovrapposizione dei flussi audio, la stessa operazione non può essere effettuata nel caso di una video-conferenza. Il modulo Multicast demanda perciò il mixing dei dati al plugin che richiede il servizio, occupandosi solo di gestire la sessione di trasmissione e di distribuire il flusso mixato, a prescidere dai contenuti.

[4]per rete di overlay si intende una rete virtuale costruita al di sopra di un'altra rete.

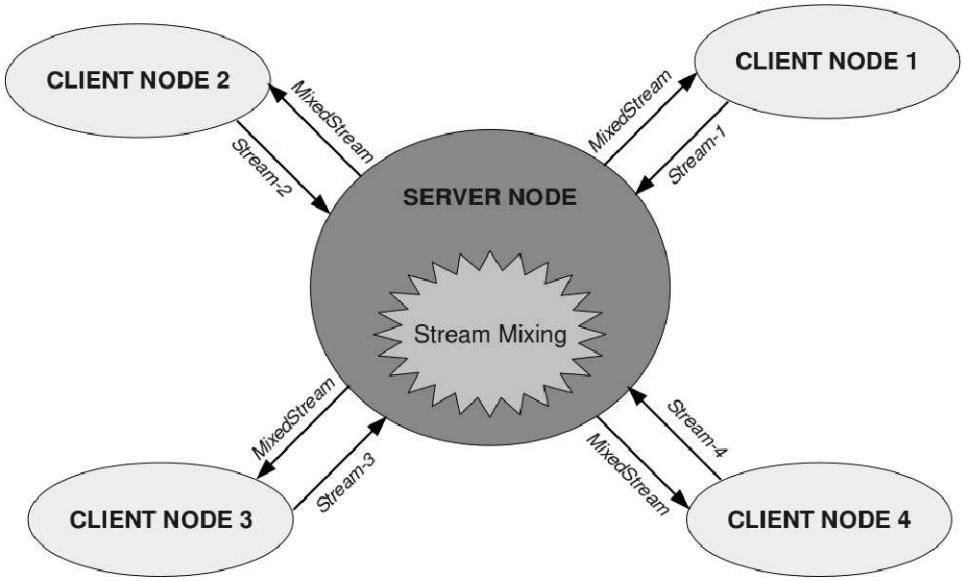

Figura 2.2: Organizzazione della rete in SimpleConference

2.2.1 Limiti dell'architettura centralizzata

L'architettura fin qui descritta ha il pregio di essere relativamente semplice da implementare, in quanto la gestione della comunicazione è centralizzata nell'unico nodo server. Inoltre, SimpleConference mantiene bassa la latenza nella distribuzione dei contenuti, caratteristica questa molto desiderabile ad esempio nel caso di conferenze audio o video.

A fronte di questi due vantaggi, si devono però registrare anche due difetti che ne limitano l'utilizzo.

In primo luogo, la **robustezza** della rete è inficiata dalla sua natura centralizzata. Il funzionamento dell'intero sistema dipende infatti in maniera indissolubile dal nodo server. Se il nodo server cade, l'intera comunicazione viene interrotta a prescindere dallo stato degli altri nodi.

L'altro problema è la scarsa **scalabilità** del sistema, e anch'essa dipende dal nodo server. Esso è di fatto il collo di bottiglia della rete, sia per il carico computazionale a cui è sottoposto nell'eseguire il mixing dei flussi di informazione, sia per la larghezza di banda di cui necessita per trasmettere il flusso mixato a tutti gli altri nodi. In particolare, la necessità di banda cresce linearmente all'aumentare del numero dei nodi nel gruppo. Si pensi, ad esempio, al caso di una trasmissione a rate B costante tra N nodi. Per i nodi client è sufficiente una larghezza di banda pari a B, mentre il server necessita di una larghezza di banda almeno pari a $(N-1)B$.

Visto quest'ultimo punto, si può concludere che SimpleConference è una soluzione che può essere utilizzata per la distribuzione di contenuti in multicasting tra gruppi di nodi non troppo numerosi. Per gruppi multicast di dimensioni medio-grandi è necessario cambiare architettura, organizzando i nodi del gruppo in una struttura che distribuisca più equamente il carico computazionale e i consumi di banda. Una possibile soluzione in tal senso è illustrata di seguito.

2.3 Architettura distribuita

L'architettura distribuita chiamata AdvancedConference è stata concepita per superare i limiti di scalabilità e affidabilità di SimpleConference. A tal fine, è necessaria una distribuzione più equa, tra i nodi del gruppo, del carico computazionale e dei consumi di banda. L'organizzazione a cui si è pervenuti è basata sul protocollo multicast di livello applicativo NICE[5], descritto in [10]. Tale architettura dispone l'insieme dei membri di uno stesso gruppo in una topologia gerarchica. La gerarchia è cruciale per aumentare la scalabilità della rete e viene creata assegnando i nodi ai vari livelli.

Nell'organizzazione gerarchica, i livelli sono numerati sequenzialmente a partire dal più basso, L_0. In ciascun livello, i membri sono suddivisi in *cluster*, che aggregano sottoinsiemi di nodi tra loro "vicini"[6]. Ogni cluster è centrato attorno ad ha un nodo eletto come *leader*[7]. I membri dell'intero gruppo sono assegnati ai vari livelli secondo il seguente principio: tutti i nodi appartengono al livello più basso, L_0; i leader dei cluster di livello L_i formano il livello L_{i+1}. Ricorsivamente, si costituiranno nuovi cluster nel nuovo livello formatosi e verranno eletti i relativi leader. Si noti che i nodi leader appartengono a più cluster, uno per ogni livello in cui compaiono.

Per distribuire i dati, i nodi sorgente inviano la propria informazione a tutti i membri dello stesso cluster. Questa informazione viene propagata dai leader agli altri livelli, diffondendola in tutti i cluster a cui appartengono. La distribuzione dei contenuti quindi avviene in senso orizzontale all'interno di ciascun cluster, e in senso verticale mediante i nodi leader.

La figura 2.3 mostra un esempio di funzionamento del protocollo. Al livello L_0 i nodi sono raggruppati in quattro differenti cluster, i cui leader sono B_0, B_1,

[5] acronimo ricorsivo che significa "NICE is the Internet Cooperative Environment", `http://www.cs.umd.edu/projects/nice/`

[6] il concetto di vicinanza dipende dalla metrica con cui si misura la distanza fra nodi, e sarà approfondito in 2.4.

[7] anche il concetto di "centro" del cluster sarà approfondito in 2.4.

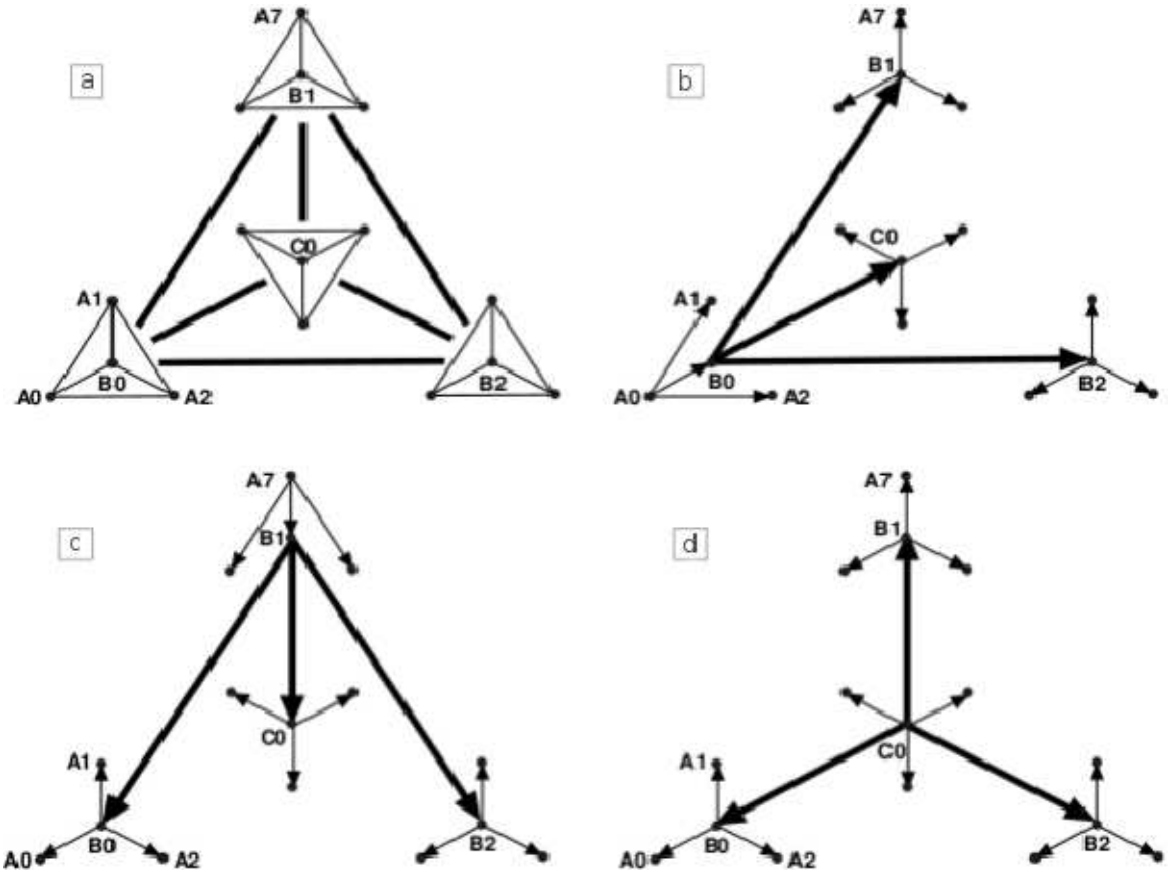

Figura 2.3: Esempio di configurazione e funzionamento AdvancedConference.

B_2 e C_0. Tali leader formano il livello L_1 in cui appartengono tutti ad un unico cluster con leader C_0. Nelle figure 2.3 (b),(c) e (d) sono mostrati i cammini lungo i quali si diffonde nella rete un messaggio inviato rispettivamente da A_0, A_7 e C_0.

In figura 2.3 si può notare come, all'interno di ciascun cluster, sia utilizzata una topologia a grafo completo per la distribuzione dei dati. È possibile, in alternativa, applicare la strategia di SimpleConference all'interno dei suddetti, ponendo il nodo leader come server. In una tale soluzione, i nodi leader dovrebbero occuparsi del mixing dei flussi provenienti sia dagli altri livelli, che dai membri dei cluster a cui appartengono.

Nell'ipotesi di avere N membri totali, cluster di taglia media pari ad n ($1 < n < N$) ed una trasmissione a rate B, i nodi che appartengono solo al livello più basso ($L = 0$) necessitano di banda pari a B, mentre un nodo di livello $L > 0$ necessita di banda (almeno) pari a $(n \cdot B)L + B$.
Inoltre, essendo il numero di livelli esistenti $O(log_n N)$, il diametro della rete[8] (e

[8]per diametro di una rete si intende la distanza massima tra due nodi.

quindi la latenza della comunicazione) è logaritmica nel numero di nodi.

Queste considerazioni dimostrano la migliore **scalabilità** dell'organizzazione AdvancedConference rispetto a SimpleConference, in cui la banda necessaria al server, ed il suo carico computazionale, cresce linearmente con l'aumento delle dimensioni del gruppo.

Anche dal punto di vista della **robustezza** AdvancedConference migliora in parte SimpleConference. Facendo di nuovo riferimento alla figura 2.3, si pensi al caso in cui un guasto provoca la caduta del nodo B_0; questo provocherebbe l'isolamento dei nodi A_0, A_1 e A_2, ma il resto della rete continuerebbe a funzionare correttamente. Se invece il guasto colpisse il nodo C_0, rimarrebbero comunque attive le comunicazione interne ai tre cluster coordinati dai nodi B. La caduta di un nodo leader causa quindi l'isolamento di un sottoinsieme di nodi e su questa base la connettività può essere velocemente ripristinata mediante un'opportuna strategia di tolleranza ai guasti.

In conclusione, l'architettura distribuita AdvancedConference, a fronte di una maggiore complessità nella topologia della rete, esibisce una migliore scalabilità rispetto all'architettura centralizzata SimpleConference e per questo è più indicata per gruppi di dimensioni medio-grandi. D'altro canto, la strategia SimpleConference rimane preferibile per comunicazioni multicast pochi-a-pochi, non solo per la sua semplicità di gestione (riduce il traffico di controllo), ma anche per la minor latenza. Il limite tra le due strategie può essere fissato sulla base delle capacità del nodo server in SimpleConference. Infatti, è possibile sfruttare i vantaggi dell'organizzazione centralizzata fintantoché il nodo server riesce a sopportarne il peso in termini di banda e carico computazionale. Se la dimensione del gruppo multicast è troppo elevata, la strategia AdvancedConference consente di distribuire il carico computazionale e di banda su più nodi della rete.

2.4 Aspetti avanzati

Sia nell'architettura centralizzata che in quella distribuita si devono scegliere particolari nodi per diffondere i flussi dati nella rete. L'oculata scelta di tali nodi, il server in SimpleConference e i leader in AdvancedConference, risulta cruciale per le prestazioni della rete. Si possono individuare tre parametri essenziali sui quali basare la scelta del nodo: posizione relativamente agli altri nodi, banda disponibile e affidabilità. In questo paragrafo saranno presentate considerazioni e spunti di riflessione in merito a questi aspetti. Non saranno illustrate soluzioni finali,

in quanto nella prima fase della realizzazione del modulo Multicast l'attenzione è stata concentrata sull'implementazione di un sistema funzionale. Le considerazioni esposte qui di seguito forniscono tuttavia il punto di partenza per una seconda fase, in cui si vogliano migliorare le prestazioni dell'implementazione esistente.

La posizione del nodo server o leader rispetto ai suoi client si ripercuote sulla **latenza** delle comunicazioni. La distanza tra due nodi è infatti qui intesa come ritardo nella trasmissione tra di essi. Il problema di scegliere un nodo centrato nel gruppo o nel cluster si traduce quindi nella necessità di trovare quel nodo che ha la minor distanza massima da tutti gli altri. L'approccio banale suggerisce di misurare la latenza tra tutti i nodi del gruppo o del cluster, inviando pacchetti di prova. Il problema di questo approccio è chiaramente la sua complessità computazionale, dato che in un gruppo di N nodi si devono eseguire $O(N^2)$ misure. Una soluzione più scalabile risiede nel'algoritmo *Vivaldi*[12]. Senza entrare nei dettagli implementativi, è sufficiente sapere che questo algoritmo assegna determinate coordinate a ciascun nodo. Queste coordinate sono calcolate mediante poche misurazioni di latenza tra i nodi coinvolti e sono definite in modo tale che la differenza di coordinate tra due nodi predice la latenza nella comunicazione tra essi.

Un'altro parametro da tenere in considerazione per la scelta del nodo centrale è la larghezza di **banda** che questo ha a disposizione. Questo parametro pone un limite massimo al grado[9] del nodo nella topologia di rete. Nell'ipotesi di avere una trasmissione a rate R e un nodo con larghezza di banda B, il grado massimo di tale nodo è $\lfloor B/R \rfloor$. La misurazione della banda disponibile di un nodo è un problema molto delicato, che in letteratura non ha ancora trovato una soluzione definitiva. Un metodo alquanto meccanico di risolvere tale problema è il seguente: si supponga di conoscere la banda B_A di un nodo iniziale A; per cercare di scoprire la banda di un nodo B si cerchi di saturare il canale di trasmissione tra A e B; se il canale si satura ad un *bitrate* inferiore a B_A, allora il collo di bottiglia della trasmissione è rappresentato dal nodo B, che ha banda pari a tale bitrate; altrimenti, se il canale satura ad un bitrate pari a B_A, allora si è trovato un limite inferiore alla banda di B, essendo A il collo di bottiglia. Tuttavia questo metodo trova valori relativi alla banda istantanea disponibile al momento del test, e non un valore assoluto.
La banda disponibile risulta essere un parametro delicato soprattutto per quanto

[9]per grado di un nodo si intende il numero di nodi a cui esso è connesso.

riguarda l'invio di dati. Si tenga presente che gli utilizzatori finali di PariPari saranno per lo più utenti domestici che si connettono alla rete mediante i propri pc. Tipicamente, questo genere di utenti sottoscrive con il proprio *ISP*[10] un contratto per l'accesso ad Internet con banda di *download* maggiore rispetto a quella di *upload*. Questo sbilanciamento è volontariamente realizzato per aumentare la capacità di ricezione dati, sacrificando parte della banda destinata all'invio. L'asimmetricità della connessione ad Internet è favorevole nel contesto di architetture client-server, dato che per i client degli utenti domestici sono molti di più i dati ricevuti rispetto a quelli inviati. Nel contesto di una rete peer-to-peer non esiste una distinzione a priori tra nodi client e nodi server. Di conseguenza, la scarsa capacità di inviare dati diventa un collo di bottiglia per molti utenti, tra cui quelli che fungono da "server" o "leader" in gruppi multicast.

Infine, l'elevata **affidabilità** è una proprietà fondamentale per i nodi server o leader se si vuole una rete soggetta il meno possibile a guasti. Se da una parte il concetto di affidabilità è palese, non è forse altrettanto ovvio come riuscire a quantificare un tale paramentro. In questo senso ci viene in aiuto il seguente risultato di uno studio maturato in [13] e sfruttato dal protocollo mTreebone[14] per identificare nodi stabili: in reti di overlay multicast, nodi più anziani tendono a rimanere connessi più a lungo. L'affidabilità di un nodo può quindi essere stimata sulla base del suo *uptime*, considerando più stabili i nodi da più tempo connessi alla rete.
Si deve altresì prevedere un protocollo di ripristino della connettività della rete in caso di guasto. La rivelazione del guasto può essere facilmente rilevata in conseguenza di un'interruzione del flusso dati. Sarà sufficiente provare a contattare il nodo con cui si era in comunicazione per scoprire se il blocco è dovuto a un guasto oppure alla conclusione della trasmissione. Eventualmente, per poter ripristinare la connessione una possibile soluzione è quella di ricorrere alla ridondanza di informazioni sulla topologia di rete, conservata da ciascun nodo. Ad esempio, nell'organizzazione SimpleConference i nodi client possono mantenere un lista di tutti i partecipanti alla conferenza in modo che, se cade il server, gli altri nodi possono creare una nuova rete tra di loro. Nel caso della strategia AdvancedConference, i nodi di un cluster dovranno mantenere la lista non solo dei nodi paritari, ma anche di quelli al livello superiore, in modo da poter ripristinare la comunicazione con il resto del gruppo se cade il leader del loro cluster. Questa soluzione di ripristino da guasti comporta un aumento del consumo di memoria nei nodi, dovendo tenere traccia di una parte della topologia della rete.

[10] Internet Service Provider.

Conclusioni In questo secondo capitolo sono stati esposti i dettagli di progetto del modulo Multicast per PariPari. Sono state esposte due differenti strategie per organizzare i nodi di un gruppo multicast e distribuire un mix dei flussi di informazione da essi generati. La prima architettura vista è chiamata SimpleConference ed è una struttura centralizzata, con bassa latenza ma che richiede molta banda e carico computazionale al nodo al centro del gruppo. La seconda architettura, AdvancedConference, si basa su un organizzazione gerarchica dei membri per distribuire il carico computazionale e di banda su più nodi della rete. Questa seconda strategia presenta una maggiore scalabilità ed è la soluzione adottata per gruppi multicast numerosi. SimpleConference rimane una soluzione preferibile nel caso di comunicazioni pochi-a-pochi. Nell'ultimo paragrafo sono stati fissati alcuni punti cruciali per realizzare in maniera efficiente le due architetture presentate, ponendo l'accento sull'importanza dei parametri di banda, latenza e uptime per i nodi cardine dei gruppi. È stata infine esposta una possibile strategia per la tolleranza ai guasti nella rete. Con questo si conclude il capitolo relativo alla progettazione del modulo Multicast. Nel prossimo capitolo sarà illustrata una panoramica sulla realizzazione del modulo, in particolare sull'implementazione di SimpleConference.

Capitolo 3

Implementazione

In questo capitolo saranno presentati i dettagli della realizzazione del modulo Multicast. Sarà innanzitutto descritta l'interfaccia che consente ai plugin applicativi di interagire con Multicast e utilizzarne i servizi. Saranno quindi presentate le classi e gli oggetti utilizzati per gestire i gruppi multicast e per consentire la comunicazione tramite la strategia di SimpleConference attualmente implementata.

Per non appesantire troppo la lettura, verranno presentati i punti cruciali ed i concetti chiave per comprendere il funzionamento del modulo. D'altro canto, il codice presente nel *repository* dell'*SVN* di PariPari è accuratamente commentato e di facile lettura una volta che siano state assimilate le caratteristiche qui descritte.

3.1 Interfaccia

L'interfaccia di Multicast fornisce, agli utilizzatori del modulo, tutti i metodi per poter accedere al servizio di comunicazione multicast. Questa interfaccia è stata sviluppata in accordo con le direttive per la comunicazione tra plugin in un client PariPari, presentate in [2, par. 4.2].

L'interfaccia è costituita dalla classe astratta `MulticastConnectionAPI`, appartenente al *package* `paripari.API`. I metodi (astratti) di tale classe sono implementati dalla classe `MulticastConnectionAPIImpl` all'interno del pacchetto Multicast. Un plugin che desidera instaurare una comunicazione multicast, deve inoltrare un'opportuna richiesta al *Core* di PariPari e riceverà in risposta un'istanza di `MulticastConnectionAPI`, attraverso la quale può richiamare i metodi implementati dal modulo Multicast. Nella richiesta inoltrata al Core vengono incapsulati i parametri richiesti dal costruttore dell'interfaccia. Tali parametri sono:

- il tipo di connessione richiesta, che può essere SimpleConference o AdvancedConference;
- il tipo di azione da intraprendere, ovvero `CREATE` se si vuole costruire un nuovo gruppo multicast, `JOIN` se ci si vuole unire ad un gruppo esistente oppure `LISTEN` se ci si vuole semplicemente mettere in attesa di ricevere richieste multicast da altri nodi della rete;
- l'indirizzo del nodo con cui si vuole creare il nuovo gruppo multicast oppure di un nodo membro del gruppo a cui ci si vuole aggregare;
- un numero intero che identifica univocamente il gruppo multicast all'interno della rete PariPari;
- il valore della larghezza di banda necessaria alla trasmissione dati nel gruppo[1].

I metodi forniti dall'interfaccia consentono al plugin utilizzatore la gestione del gruppo e lo scambio dati con gli altri membri. Questi metodi sono:

- `addSource`, per invitare nuovi nodi ad aggregarsi al gruppo;
- `leaveGroup`, per abbandonare il gruppo multicast;
- `getStreams`, che restituisce i buffer sui quali saranno letti i flussi dati provenienti dagli altri nodi;
- `getMixedStreamWriter`, che restituisce il buffer sul quale scrivere i dati da inviare agli altri membri; i nodi che fungono da "client" inviano solo i proprio flusso di informazioni generato localmente, mentre i "server" o "leader" trasmettono il mix dei dati di più membri;
- `getMembers`, restituisce la lista degli indirizzi dei nodi membri del gruppo;
- `getGroupId`, restituisce l'identificatore del gruppo;
- `getStatus`, restituisce lo stato del gruppo (`CONNECTED`/`DISCONNECTED`);
- `getEventNotifier`, restituisce un oggetto attraverso il quale Multicast notifica al plugin utilizzatore eventuali cambiamenti nella composizione del gruppo (aggiunta/rimozione di membri).

[1] tale valore è richiesto per usufruire dei *socket* forniti dal modulo *Connectivity*.

Lo scambio dati tra il modulo Multicast, che accede effettivamente alla rete, e il plugin utilizzatore avviene tramite buffer sincronizzati, in particolare sfruttando gli oggetti `PipedInputStream` e `PipedOutputStream` della libreria standard Java; questo meccanismo sarà meglio chiarificato in 3.3.

Come già detto, questi metodi sono definiti all'interno dell'interfaccia `MulticastConnectionAPI` del package `paripari.API` e sono implementati in `MulticastConnectionAPIImpl` del pacchetto `paripari.multicast`.

Le classi, gli oggetti e il funzionamento descritto nel seguito del capitolo si riferiscono all'implementazione del modulo Multicast realizzata all'interno del pacchetto `paripari.multicast`.

3.2 Gestione di gruppi multicast

Un gruppo multicast è rappresentato all'interno del modulo mediante specifiche classi che ne descrivono la topologia.

Un gruppo è implementato per mezzo della classe `MulticastGroup` e mantiene al suo interno le variabili necessarie a descriverlo, ovvero la lista dei membri e un numero intero che identifica univocamente il gruppo all'interno della rete PariPari. Inoltre, vengono forniti i metodi, utilizzati nello *scope* del pacchetto `paripari.multicast`, per aggiungere o rimuovere membri al gruppo.

Un singolo nodo all'interno del gruppo è rappresentato per mezzo della classe `MulticastNode`. Il modulo Multicast detiene una diversa istanza di questa classe per ogni membro esistente nel gruppo. Si può verificare la relazione topologica tra il nodo locale e il membro remoto mediante il metodo `getNodeRelationship`, che restituirà `FATHER` se il nodo locale è client del nodo remoto, `CHILD` nel caso opposto oppure `PEER` se i due nodi sono paritari. Queste relazioni servono a Multicast per riconoscere i membri ai quali inoltrare il flusso dati, in accordo con la strategia di comunicazione messa in atto.

3.2.1 Traffico di controllo

Le informazioni scambiate tra i nodi per controllare la topologia del gruppo è inviato su un canale dedicato, implementando così un sistema del tipo *out-of-band control* per la segnalazione. Questo traffico passa attraverso una specifica porta aperta da Multicast, chiamata `multicastPort`, e viene utilizzato per contattare gli altri nodi nel corso di operazioni di *create* o di *join*, oppure per segnalare agli

altri membri un cambiamento nella topologia del gruppo. I pacchetti di controllo sono definiti da una specifica classe, chiamata `MulticastDatagramPacket`, e sono trasmessi mediante protocollo *UDP*. Il modulo Multicast mantiene un apposito *thread*, il `RemoteNodeListener`, in ascolto sulla `MulticastPort` per ricevere e gestire i pacchetti di controllo.

Si è scelto di far passare questo traffico su un canale dedicato, distinto da quello per la distribuzione dei flussi dati, per facilitarne il riconoscimento. La strategia opposta, infatti, avrebbe richiesto l'aggiunta di un *header* ai pacchetti per distinguere quelli di controllo dai dati; questa soluzione, oltre ridurre la banda disponibile per l'effettivo *payload*, avrebbe anche comportato un'operazione di lettura e smistamento pacchetti, rallentando la consegna dei dati al plugin applicativo.

3.3 Comunicazione in SimpleConference

Vediamo ora gli aspetti cruciali per la realizzazione di *SimpleConference*. Si ricordi che questa strategia consiste di un'organizzazione centralizzata, in cui un nodo prescelto funziona da server, è in comunicazione diretta con tutti gli altri membri del gruppo ed esegue il mixing dei flussi dati. Tralasciando le considerazioni fatte in 2.4, è designato come server il nodo che dà inizio alla creazione del gruppo.

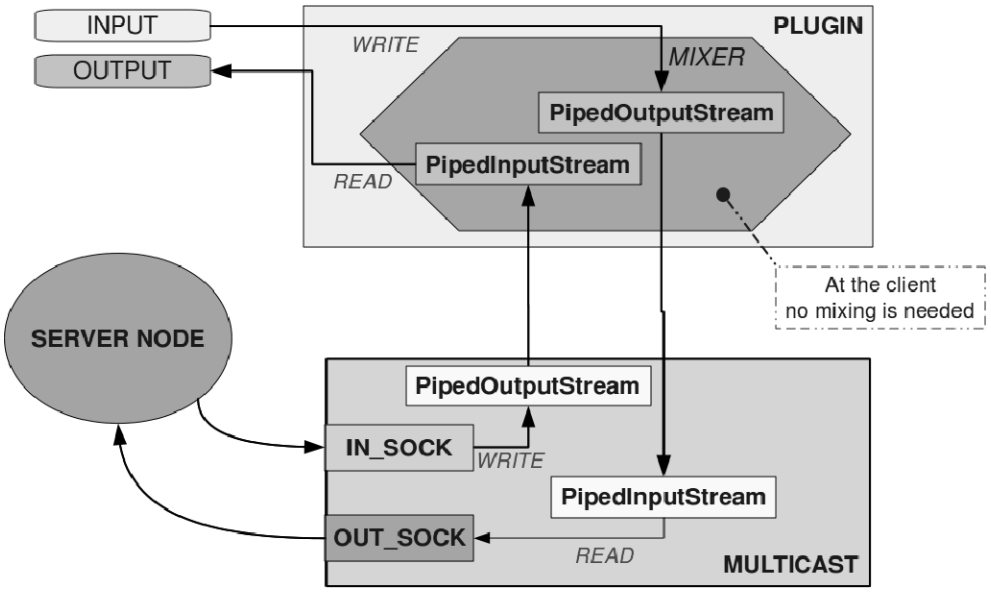

Figura 3.1: Architettura di un client in SimpleConference.

La figura 3.1 mostra l'architettura di un nodo client. Il modulo Multicast mantiene un socket aperto per la comunicazione con il nodo server. Uno specifico thread, il `IncomingDataHandler`, rimane in attesa di ricevere il flusso dati dal server del gruppo. Per passare i dati ricevuti da Multicast al plugin applicativo si utilizzano due buffer di tipo `PipedOutputStream` e `PipedInputStream`[2] tra loro connessi; il primo è detenuto da Multicast mentre il secondo è detenuto dal plugin applicativo. Il funzionamento in fase di ricezione dati è il seguente: il `IncomingDataHandler` copia i dati letti dal socket nel buffer `PipedOutputStream`; eseguendo su quest'ultimo il metodo `flush`, i dati sono automaticamente ricopiati nel relativo `PipedInputStream` e il plugin applicativo viene notificato della possibilità di leggere i dati ricevuti. La necessità di passare i dati da un buffer di Multicast ad un buffer del plugin è dettata dalle politiche di sicurezza in uso in PariPari[2, cap. 8].

Per inviare i dati generati localmente dal plugin applicativo verso il nodo server, il funzionamento è analogo: il plugin deve scrivere il proprio flusso dati su un buffer di tipo `PipedOutputStream` connesso ad un `PipedInputStream` detenuto da Multicast; dal lato multicast, un `OutgoingDataHandler` si occupa di trascrivere i dati del plugin dal buffer di lettura al socket aperto verso il server del gruppo.

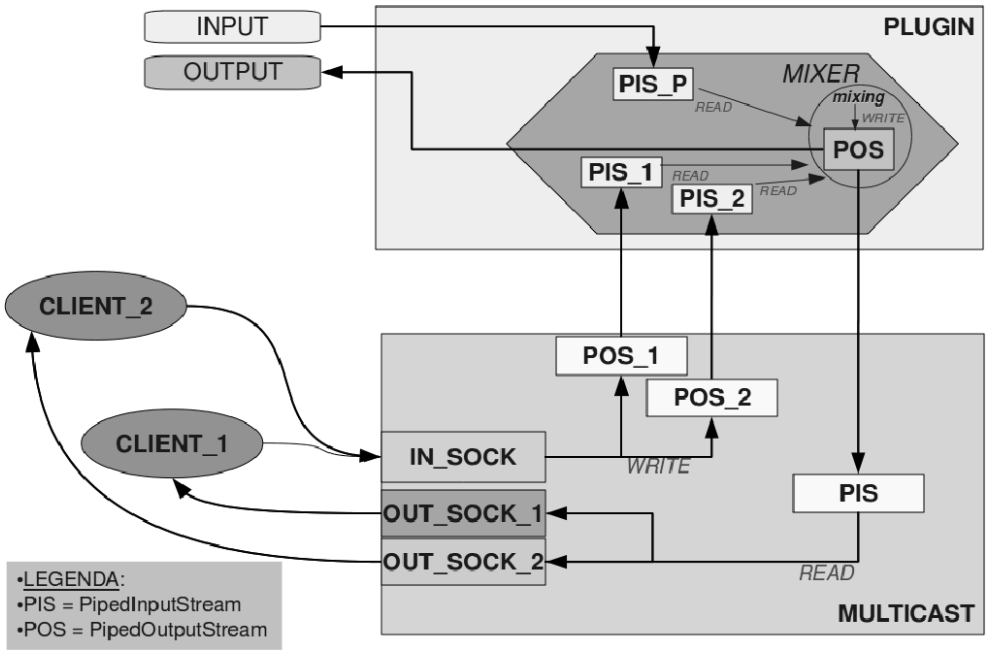

Figura 3.2: Architettura di un server in SimpleConference.

[2]appartengono al pacchetto `java.io` della libreria standard Java.

L'architettura del nodo server, illustrata in figura 3.2, è molto simile a quella del client. La differenza all'interno del modulo Multicast risiede nel fatto che questo nodo è in comunicazione diretta con tutti gli altri membri del gruppo. Di conseguenza devono essere gestiti più flussi dati, sia in ingresso che in uscita, utilizzando un socket (ed un relativo buffer `PipedOutputStream`) per ogni altro nodo del gruppo. Dal lato del plugin che utilizza multicast, la differenza rispetto al client risiede nella responsabilità di effettuare il mixing di tutti i flussi dati ricevuti. Tale operazione è demandata al plugin in quanto dipende strettamente dal tipo di contenuti della comunicazione; non è compito di Multicast occuparsi del tipo di dati trasmessi, bensì solo della loro diffusione all'interno del gruppo. Il flusso dati mixato sarà passato a Multicast attraverso un'opportuna coppia di buffer `PipedOutputStream`/`PipedInputStream` e un thread `OutgoingDataHandler` provvederà a spedire tale flusso a tutti i membri del gruppo.

Conclusioni Con questo possiamo ritenere concluso il presente capitolo in cui sono stati esposti i punti cruciali dell'implementazione del modulo Multicast in PariPari. Il modulo Multicast si propone come un livello intermedio tra la rete fisica e i plugin che desiderano creare una comunicazione in multicasting. L'implementazione del modulo Multicast provvede pertanto a fornire supporto per tale tipo di connettività. Ciò avviene mantenendo le informazioni relative alla topologia del gruppo, scambiando opportuni pacchetti di controllo tra i nodi della rete e facendo da tramite, al plugin utilizzatore, per l'invio e la ricezione di dati.

Conclusioni

Il lavoro svolto per la redazione di questa tesi di laurea ha costituito un'esperienza estremamente formativa. Il progetto PariPari, per la sua complessità, consente agli studenti partecipanti di confrontarsi con problemi di progettazione, sviluppo, pianificazione e gestione delle risorse non comune ai lavori di tesi redatti da singoli individui o, tutt'al più, da piccoli gruppi.

Personalmente, oltre a migliorare le capacità di programmatore, il progetto mi ha permesso di sviluppare conoscenze nell'ambito del collaudo di programmi tramite testing. Inoltre ho potuto comprendere le problematiche derivanti dal coordinamento e la gestione di un gruppo di lavoro numeroso ed eterogeneo.

Lo sviluppo del modulo Multicast ha comportato un notevole impegno nell'ambito dell'ingegneria del software, vista la necessità di implementare un servizio *ex novo* all'interno di un progetto già avviato. Si è dovuto progettare il modulo in conformità agli standard esistenti nel gruppo e ponendo particolare attenzione alle esigenze dei potenziali utilizzatori. In questo senso la fase di progettazione si è distinta rispetto allo sviluppo di un programma autonomo e relativamente semplice a cui ero stato abituato nel corso dei miei studi.

Dal punto di vista della programmazione, PariPari ha favorito un raffinamento della tecnica. La condivisione del codice tra molti sviluppatori ha dettato l'esigenza di imporre degli standard per lo stile di programmazione e la documentazione del codice. Questo ha insinuato nei programmatori l'abitudine a scrivere codice "pulito" e ben documentato, facilitandone la lettura e operazioni di modifica quali ad esempio il *refactoring*.

Oltre a questo, il progetto PariPari mi ha fatto comprendere l'importanza dell'utilizzo di strumenti che facilitano lo sviluppo in software piuttosto complessi, quali l'ambiente di sviluppo Eclipse[3] e il sistema di controllo versione Subversion[4].

La filosofia eXtreme programming adottata ha imposto uno sviluppo test-

[3] `www.eclipse.org/`

[4] subversion.tigris.org/

driven, portando di conseguenza gli studenti ad acquisire conoscenza delle procedure di collaudo di software. Questa tecnica viene raramente insegnata nel corso degli studi accademici e pertanto PariPari ha rappresentato, anche sotto questo aspetto, un'opportunità unica.

Oltre all'incremento delle conoscenze tecniche, PariPari mi ha portato a confrontarmi con i problemi di pianificazione ed organizzazione all'interno di un gruppo di lavoro numeroso. Nell'arco della mia permanenza all'interno del gruppo ho ricoperto il ruolo di coordinatore per il gruppo di Connettività. Tale responsabilità ha comportato la gestione, in media, di sei persone al lavoro su differenti progetti correlati al plugin. I miei compiti hanno riguardato la pianificazione degli obiettivi a breve e medio termine, la supervisione del progresso dei lavori e il mantenimento delle relazioni con il resto del mondo PariPari. La principale difficoltà incontrata è derivata dall'eterogeneità del gruppo. Gli studenti, infatti, non afferivano in genere al medesimo ordinamento; questo generava la necessità di calibrare i compiti assegnati sulla base delle capacità individuali e del tempo di permanenza nel progetto. Questo processo di pianificazione, e di supervisione del progresso per rispettare le scadenze fissate, è migliorato con il passare del tempo e la maturazione dell'esperienza.

Per i motivi appena riportati, il progetto PariPari è stato un'esperienza unica di formazione e crescita: non solo per migliorare le capacità tecniche che erano state acquisite nel corso degli studi accademici, ma anche da un punto di vista manageriale, affrontando problemi di gestione del progetto che tipicamente non si riscontrano in ambiente accademico, ma solo successivamente nel mondo del lavoro.

Bibliografia

[1] P. Bertasi: *"Progettazione e realizzazione in Java di una rete peer to peer anonima e multifunzionale"*, 2006.

[2] M. Bonazza: *"PariCore"*, 2009.

[3] F. Castellana: *"PariPari:Multicast"*, 2009.

[4] M. Padovan: *"PariPari: Testing"*, 2008.

[5] A. S. Tanenbaum, M. Van Steen: *"Sistemi Distribuiti"*, Prentice Hall, 2nd ed., 2007.

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *"Design Patterns: Elements of Reusable Object-Oriented Software"*, Addison-Wesley, 1995.

[7] K. Beck: *"Extreme Programming Explained"*, Addison-Wesley, 1999.

[8] *"Host Extensions for IP Multicasting"*, RFC 1112, Agosto 1989.

[9] *"IANA Guidelines for IPv4 Multicast Address Assignments"*, RFC 3171, Agosto 2001.

[10] S. Banerjee, B. Bhattacharjee, C. Kommareddy: *"Scalable Application Layer Multicast"*, in Proceedings of ACM Sigcomm, Agosto 2002.

[11] S. Banerjee, B. Bhattacharjee: *"A comparative study of application layer multicast protocols"*, in Submitted for review, 2002.

[12] F. Dabek, R. Cox, F. Kaashoek, R. Morris: *"Vivaldi: A Decentralized Network Coordinate System"*, in SIGCOMM '04, 2004.

[13] M. Bishop, S. Rao, K. Sripanidkulchai: *"Considering priority in overlay multicast protocols under heterogeneous environments"*, in IEEE INFOCOM, 2006.

[14] F. Wang, Y. Xiong, J. Liu: *"mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast"*, 27th International Conference on Distributed Computing Systems, Giugno 2007.

Indice delle figure