UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Reti di Calcolatori

SUPPORTO PER LA GESTIONE DELL'HANDOFF VERTICALE PER SERVIZI MULTIMEDIALI

Candidato:

Veljko Vuković

Relatore:

Chiar.mo Prof. Ing. Antonio Corradi

Correlatore:

Dott. Ing. Luca Foschini

Anno accademico 2004/2005

PAROLE CHIAVE

Wireless Internet

Context Awareness

Multimedia Streaming

Vertical Handoff Management

WiFi e Bluetooth

Indice

Introduzione

Negli ultimi anni stiamo assistendo a una grande e rapida crescita nelle telecomunicazioni delle tecnologie wireless, ovvero quelle tecnologie che permettono di usare strumenti di comunicazione come laptop, cellulari e PDA, senza essere vincolati ad una postazione di accesso fissa.

Col termine "wireless" si identificano tecnologie che danno la possibilità di effettuare comunicazioni senza fili, non essendo vincolati a una posizione o postazione fissa. Questo introduce una moltitudine di possibilità di sviluppo di applicazioni per utenti "mobili". Le tecnologie wireless consentono la mobilità dell'utente all'interno dell'area di copertura dell'Access Point (AP) senza discontinuità di servizio. Questo porta una maggiore flessibilità della rete e dell'utilizzo consentendo ai terminali di mantenere la connessione pur essendo in movimento.

Uno dei sviluppi che ha avuto maggiore successo è sicuramente quello della rete cellulare che ci permette di effettuare comunicazione quasi da qualsiasi parte del mondo, anche in rapido movimento e con una connessione continua e stabile.

Le tecnologie wireless, come abbiamo accennato, hanno molti vantaggi ma presentano anche alcuni svantaggi. Uno svantaggio che sicuramente si deve affrontare è il problema della continuità e della qualità del servizio offerto quando l'utente si muove durante l'erogazione del servizio. Ogni terminale comunica con la rete wireless tramite un AP che permette ai dispositivi wireless di accedere alla rete fissa. Ogni AP copre un'area limitata, per coprire un'area più vasta sono necessari più AP che cercano di coprire la maggior parte dell'area. Quando usciamo dall'area di copertura di un AP ed entriamo nell'area di copertura di un altro abbiamo una discontinuità, dovuta al processo di un'handoff, cioè un passaggio da un AP ad un altro. Esistono due tipi di handoff : l'handoff orizzontale (il passaggio tra due AP che utilizzano la stessa tecnologia) e handoff

verticale (il passaggio tra due AP che utilizzano diverse tecnologie, e quindi il cambiamento della tecnologia di connessione da parte del client).

Attraverso una rete wireless è possibile offrire praticamente gli stessi servizi che in una rete fissa; in questa tesi però ci occuperemo dei servizi multimediali. Con servizio multimediale intendiamo generalmente la trasmissione di flussi video o audio oppure entrambe le cose insieme. Per le loro caratteristiche i servizi multimediali richiedono la stabilità e la qualità della connessione. Le discontinuità e i disturbi introdotti dall'uso delle tecnologie wireless comportano però una minore qualità del servizio. Per limitare queste situazione si deve fornire un supporto che dia maggiore stabilità e continuità di connessione.

L'obbiettivo di questa tesi è, appunto, fornire un supporto per la gestione dell'handoff verticale per sistemi multimediali con l'obiettivo di migliorare la continuità di servizio su tecnologie wireless. In particolare, si vuole sviluppare un middleware per il recupero delle informazioni di contesto del client, e per la gestione automatica dell'handoff verticale dei servizi multimediali.

Nel primo capitolo viene fatta una breve panoramica sul progetto introducendo lo scenario nel quale si andrà ad operare.

Nel secondo capitolo vengono trattate le reti wireless in generale, lo standard IEEE 802.11 (WiFi) e Bluetooth, ed infine l'handoff verticale.

Nel terzo capitolo vengono trattate le architetture preesistenti per la gestione degli handoff orizzontali rispettivamente con tecnologia WiFi e Bluetooth.

Il capitolo quattro ha lo scopo di evidenziare l'architettura adottata per realizzare l'infrastruttura e le relazioni fra le varie entità che la compongono.

Il capitolo quinto è invece dedicato all'implementazione dove sono evidenziate le varie classi introdotte e le relazioni tra queste. Inoltre, sono state introdotte due classi di test e le loro valutazioni.

Capitolo 1
Panoramica sul Progetto

Con la enorme crescita negli ultimi tempi dell'utilizzo e della richiesta dei servizi forniti utilizzando tecnologie wireless si crea sempre di più la necessità di avere erogazioni con qualità e stabilità sempre maggiori. Oltre a ciò, l'attuale scenario è caratterizzato da un'alta eterogeneità di tecnologie wireless che convivono insieme. Le più note sono sicuramente lo standard IEEE 802.11 noto anche come WiFi, Bluetooth e 3G. Tali tecnologie hanno caratteristiche diverse come larghezza di banda, area di copertura, consumo di energia, potenza del segnale, ecc. Ovviamente non essendo tecnologie a postazione fissa si possono avere diversi tipi di disturbo della qualità della trasmissione e può verificarsi un handoff quando l'utente effettua un passaggio da un'AP all'altro.

Per evitare o ridurre al minimo tutti questi tipi di degradazione si tenta di creare applicazioni che abbiano la possibilità di essere consapevoli del contesto nel quale operano (context-aware), cioè possano conoscere le caratteristiche della tecnologia nella quale operano e possano localizzare gli utenti che utilizzano i servizi. Potere sapere le posizione o le traiettorie dei terminali può infatti contribuire a migliorare la qualità del servizio. Ovviamente questo richiede che le applicazioni siano in grado di reperire tali informazioni e di effettuare un controllo (gestione) necessario, in modo da adattare il comportamento al carico e capacità dell'infrastruttura wireless.

In particolare, diventa sempre più frequente un'infrastruttura wireless dotata di molteplici interfacce di comunicazione wireless, e di tecnologie diverse. Avendo più interfacce esiste la possibilità di effettuare degli handoff verticali (passaggio da una tecnologia all'altra).

L'erogazione di servizi multimediali richiede una connessione stabile e la garanzia di una qualità di servizio (Quality of Service, QoS) per garantire la continuità di

servizio. Nelle reti wireless può però facilmente verificarsi una situazione di discontinuità di servizio dovuta a handoff di varia natura e questo non favorisce la qualità dei servizi multimediali.

L'obbiettivo di questa tesi sarà quello di fornire un supporto di gestione che, ricavando informazioni di contesto e connessione, migliori la qualità della comunicazione fornendo una politica di coordinazione tra diverse tecnologie wireless ovvero ci dia la possibilità di effettuare handoff verticali. Il tutto mantenendo una continuità di servizio tale che per l'utente tutte le discontinuità reali siano trasparenti, cioè lui stesso non si accorga della loro esistenza.

1.1 Scenario Mobile

Fino alla nascita delle tecnologie wireless per utilizzare le informazioni della rete fissa bisognava essere collegati fisicamente ad essa, ad esempio con un cavo. Ovviamente questo vincolo escludeva la possibilità di spostarsi da una posizione all'altra durante l'utilizzo del servizio. La diffusione delle tecnologie wireless consente di rimuovere questa limitazione; si da' cioè la possibilità agli utenti di usufruire dei servizi della rete fissa senza essere vincolati ad una postazione fissa. Questo consente all'utente di muoversi liberamente durante l'erogazione del servizio da lui richiesto. Per collegarsi ad una rete wireless l'utente utilizza una scheda wireless (dipendente dalla tecnologia a cui si vuole collegare) e un dispositivo capace di accedere alla rete o di comunicare con altre schede wireless.

In generale si hanno due tipi di mobilità : *mobilità del terminale*, ossia quando è possibile accedere ad un servizio in modo continuativo anche spostando il terminale da un luogo all'altro; e *mobilità personale*, ossia quando è possibile accedere a uguali servizi indipendentemente dalla postazione. In questa tesi noi ci occuperemo unicamente della mobilità del terminale.

Prima di cominciare a parlare di mobilità del terminale dobbiamo definire due elementi strutturali: la cella e il dominio. La cella è un'area formata da un'insieme

di postazioni wireless controllate e coordinate da un'unica entità detta stazione radio base (RB) che ha il compito di rendere accessibili i servizi messi a disposizione dall'infrastruttura e garantire la comunicazione tra i partecipanti alla rete. Tramite il coordinamento di più celle è possibile estendere la superficie coperta, che è necessariamente limitata, ed è possibile mettere in comunicazione terminali appartenenti a celle differenti.

Esistono tre livelli di mobilità del terminale: la *micro, macro* e *global* mobilità. La *micro* mobilità offre la possibilità a un terminale di muoversi liberamente dentro una celle diverse appartenenti alla stessa sottorete, la *macro* mobilità offre la possibilità ad un terminale di muoversi liberamente tra due celle connesse a sottoreti diverse, ma appartenenti allo stesso dominio, la *global* mobilità offre la possibilità ad un terminale di muoversi liberamente da una cella ad un'altra indipendentemente dal dominio di appartenenza.

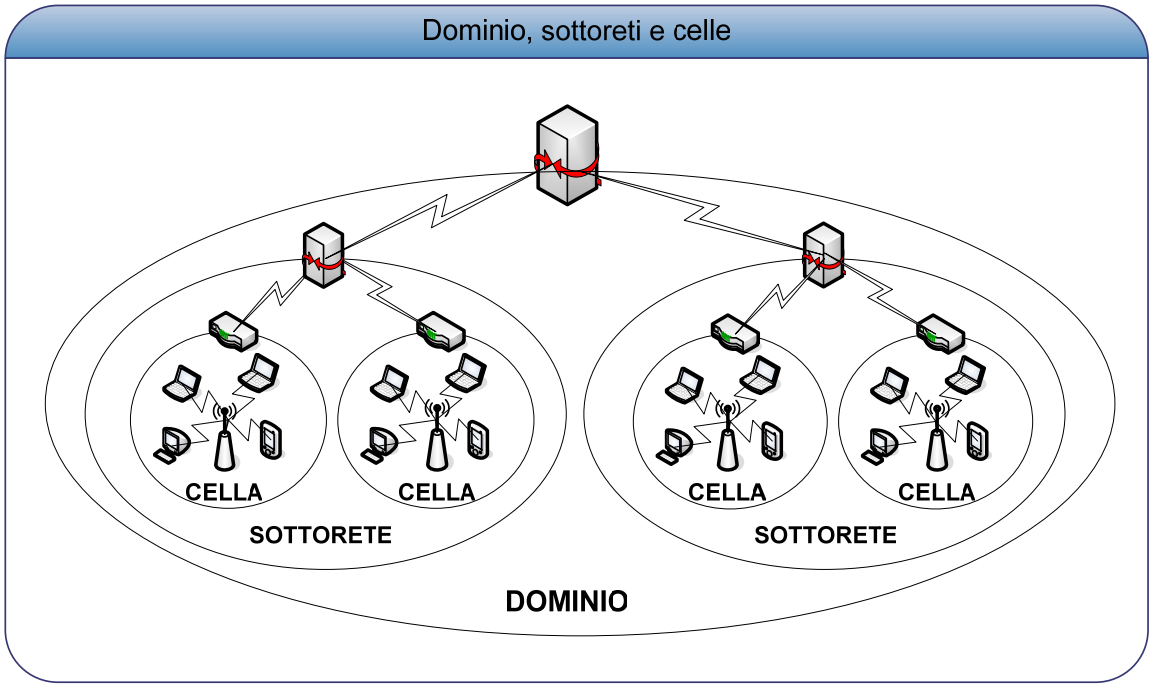

Figura 1.1: Dominio, sottoreti e celle

Per realizzare questi tipi di mobilità è necessaria un'architettura contenente le seguenti entita:

- Mobile Node (MN): rappresenta il terminale mobile;
- Access Point (AP): rappresenta la stazione radio base;
- Distribuition System (DS): rappresenta il sistema che consente l'interconnessione di diversi AP.

In una rete wireless basata sul protocollo IP, la parte di indirizzo di sottorete rimane invariato all'interno di ogni cella (e quindi anche l'indirizzo IP non varia).

1.2 Infrastruttura della rete wireless

Da tempo è nata la necessità di collegare il rete diversi terminali per poter condividere tra loro risorse e servizi. Prima dell'arrivo delle tecnologie wireless questo si faceva tramite collegamenti fisici, ad esempio cavi, e si formavano delle reti LAN (Local Area Network).

Con l'arrivo dei dispositivi wireless e con la possibilità di dare mobilità ai terminali della rete LAN si vengono a formare delle reti wireless denominate WLAN (Wireless Local Area Network). Nelle WLAN è possibile definire servizi Location Aware (consapevoli della locazione nella quale si trovano, cioè riescono a modificare il proprio comportamento in dipendenza della locazione) che nelle LAN non esistono.

Una rete wireless può essere costituita tra entità "pari" cioè i terminali comunicano direttamente tra di loro senza appoggiarsi ad un AP e quindi non hanno bisogno di un'infrastruttura. Questo modo di costituire una WLAN viene chiamato *Ad-Hoc* oppure *Peer-to-Peer*. Un esempio tipico di questa modalità è quando ci sono due terminali e uno si deve connettere all'altro per sfruttare la condivisione della sua connessione LAN.

A noi però interessa di più l'altra modalità di costituire una WLAN ed è quella denominata *a infrastruttura*. Si effettua il collegamento (bridging) di una rete

wireless ad una rete Ethernet fissa. In questa modalità i terminali wireless non possono collegarsi direttamente tra di loro ma devono farlo tramite un AP. L'architettura di una rete WLAN a infrastruttura si basa su due entità: AP e MN. I Mobile Node (MN) sarebbero i terminali dotati di schede wireless di diverse tecnologie (WiFi, Bluetooth, 3G, ecc) che utilizzano i servizi della rete. Mentre l'Access Point (AP) è l'entità che collega i diversi MN ed effettua il bridging alla rete fissa. Tipicamente l'interazione tra le due entità è di tipo Client-Server nella quale più MN si appoggiano a un AP per comunicare con la rete fissa.

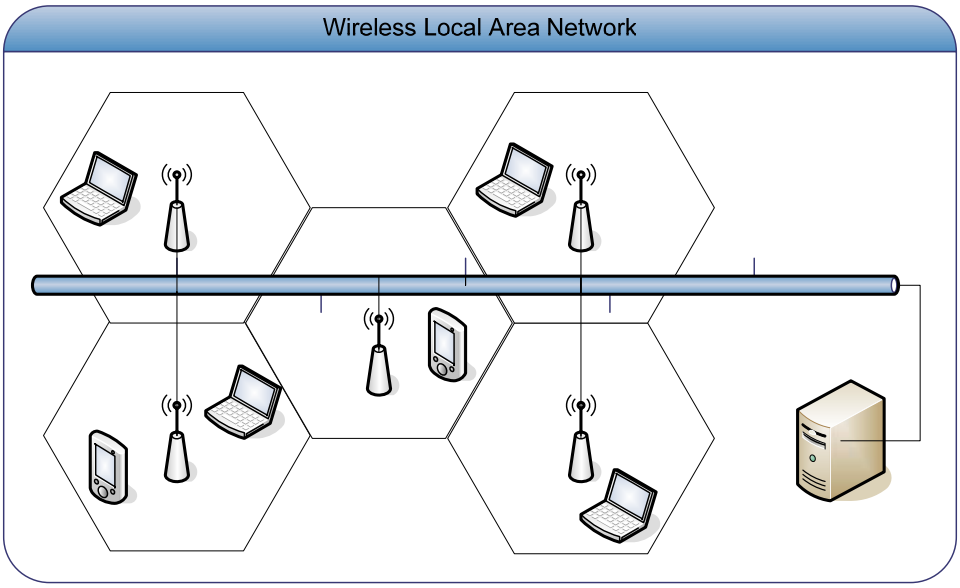

Figura 1.2: Wireless LAN

1.3 Servizi multimediali

Con servizio multimediale intendiamo generalmente la trasmissione di video, audio oppure entrambe le cose insieme.

Esempi tipici di questi servizi sono:

- *Video conferenza o Voice over IP*: in entrambi i casi due o più utenti vengono fatti collegare tra loro per comunicare simulando una conferenza all'interno di una stessa stanza. Chiaramente in questo tipo di applicazioni

il flusso delle informazioni è bi-direzionale, cioè ogni utente trasmette e riceve informazioni potenzialmente verso e da tutti i partecipanti.

- *Video on Demand* (VoD): in questo caso l'utente richiede di visualizzare un filmato che risiede in qualche archivio remoto. In questo caso il flusso dei dati è praticamente mono-direzionale e va dalla macchina che possiede il filmato all'utente che lo richiede.
- *Live*: anche in questo caso il flusso di informazioni è mono-direzionale ma invece di richiedere un filmato o un file audio in archivio, l'utente chiede di poter ricevere le immagini di un evento in diretta.

Servizi multimediali come il secondo e il terzo esempio vengono chiamati servizi di streaming. I dati in modalità streaming vengono trasportati come flusso di dati e quindi non necessitano di essere salvati interamente prima di poter essere riprodotti. I dati in arrivo vengono messi in sequenza in un buffer dal quale il ricevente preleverà i dati da visualizzare, consumando così i dati del buffer. Questo fa intuire che per fornire servizi multimediali streaming (streaming video, streaming audio, ecc) c'è bisogno di una qualità e continuità della comunicazione per evitare ritardi nella trasmissione, i quali riducono di molto la qualità del servizio. La discontinuità porta facilmente a un ritardo o nel caso peggiore blocco dell'erogazione del servizio.

1.4 Handoff

Per poter usufruire dei servizi della rete un terminale wireless deve essere collegato a un AP cioè un punto di accesso alla rete fissa. Ogni AP può offrire questo servizio in un'area limitata e per offrire un'area di copertura maggiore devono esserci più AP posizionati in modo da coprire il maggior spazio possibile.

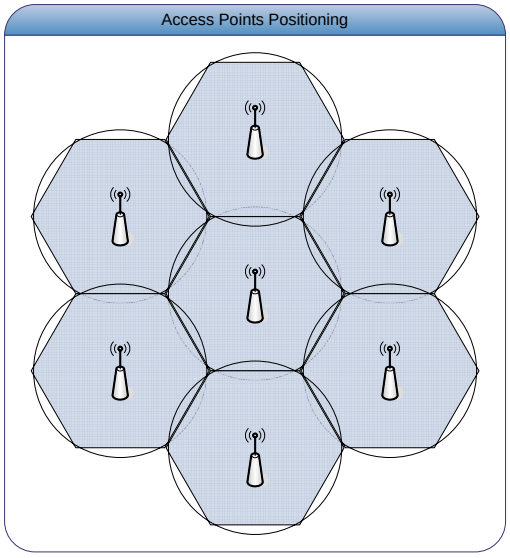

Figura 1.3: Posizionamento degli AP

Quando un terminale esce dall'area di copertura di un AP ed entra nell'area di copertura di un altro AP allora si verifica il fenomeno denominato *handoff*. Questo fenomeno può anche comportare perdite di informazione durante una disconnessione dal primo AP e la riconnessione al secondo AP, come nel caso di handoff di tipo hard (vedi sotto).

Esistono due tipi di handoff: quello *orizzontale*, che si verifica quando un terminale effettua il passaggio tra gli AP della stessa tecnologia, e quello *verticale*, che si verifica quando un terminale effettua il passaggio tra gli AP di diverse tecnologie.

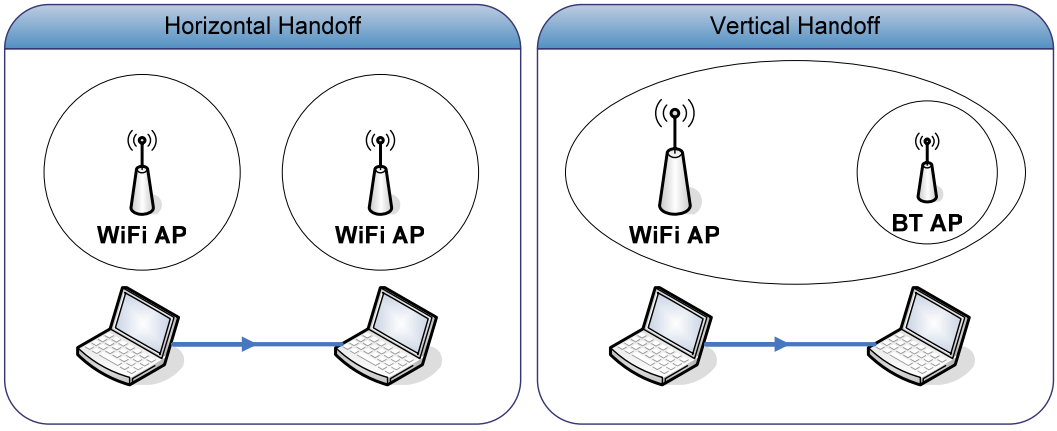

Figura 1.4: Handoff orizzontale e verticale

Esistono diverse tipologie di handoff. Per quanto riguarda l'handoff orizzontale le politiche si possono suddividere in:

- *Soft*: con questa politica durante l'handoff il dispositivo si collega contemporaneamente a più AP ed in questo modo non si perdono dati (pacchetti);
- *Hard*: con questa politica durante l'handoff il dispositivo può essere collegato al massimo a un AP e quindi c'è perdita di dati (pacchetti).

Esiste una ulteriore suddivisione delle politiche, sia per l'handoff orizzontale sia per quello verticale, in:

- *Reactive*: con questa politica, cercando di minimizzare il numero degli handoff, lo si esegue solo quando viene persa la connessione con l'AP attuale. Ovviamente questo porta tempi di handoff più lunghi dato che prima di riconnettersi ad un AP dovrà trovarlo effettuando una ricerca;
- *Proactive*: con questa politica si avvia la procedura di handoff prima che il segnale con AP attuale venga perso, questo viene fatto appena si trova un'AP con un segnale migliore di quello attuale. In questo modo diminuiscono i tempi dell'handoff, ma aumenta il consumo di energia.

Un handoff verticale è ulteriormente classificabile in un handoff:

- *verso l'alto*: un'handoff verso una tecnologia con la dimensione delle celle superiore (es. da Bluetooth a WiFi);
- *verso il basso*: un'handoff verso una tecnologia con la dimensione delle celle inferiore (es. da WiFi a Bluetooth).

La tecnologia WiFi gestisce in modo automatico l'handoff orizzontale a livello Datalink (livello 2 dell'OSI) adottando una strategia di handoff di tipo hard, mentre la tecnologia Bluetooth non lo fa. Nessuna tecnologia wireless gestisce in modo automatico l'handoff verticale.

Nell'ottica dei servizi multimediali l'handoff è una situazione spiacevole perché porta a una discontinuità di servizio dovuta alla disconnessione dal primo AP e la riconnessione al secondo. Questo effetto negativo si amplifica nei servizi

multimediali dato che essi richedono una comunicazione tra l'utente che utilizza il servizio e l'entità che lo eroga in modalità real-time (in tempo reale). Questi problemi sono presenti anche a causa delle caratteristiche intrinseche del protocollo real-time che usualmente si basa sul protocollo UDP (User Datagram Protocol) che è un protocollo non connesso e quindi non offre qualità di servizio (QoS). Viene intuitivo capire che queste discontinuità portano a una riduzione della qualità del servzio. Per capire maggiormente la problematica facciamo un esempio: l'utente guarda un filmato in streaming muovendosi e ad un certo punto avviene l'handoff durante il quale l'utente non riceve più i nuovi dati e quindi la visualizzazione del filmato sarà ferma all'ultimo dato ricevuto, fino al passaggio dell'handoff.

1.5 Context-awareness

Le reti wireless sono contraddistinte dalla loro mobilità e della grande dinamicità con la quale cambiano gli utenti di una rete o la tipologia della stessa. Tutta questa dinamicità può essere gestita solamente dando alle applicazioni la capacità di ottenere e capire le informazioni di contesto nel quale operano. Le applicazioni che hanno questa caratteristica vengono definite applicazioni context-aware. L'efficienza e soprattutto la qualità di un'applicazione aumenta notevolmente potendo sapere il contesto nel quale opera e potendo adattare il suo funzionamento in conformità di tali informazioni.

Le informazioni possono essere di varia natura, sicuramente per le reti wireless le più importanti sono l'identità, informazioni spaziali e temporali, le risorse accessibili, e la conoscenza della tipologia della rete. Altre informazioni possono riguardare l'ambiente, le attività in corso, misurazioni fisiche, ecc.

Un grosso vantaggio delle applicazioni context-aware è dato dalla semplificazione dell'interazione tra utente e applicazione dato che e la stessa applicazione che, per le azioni legate al contesto, recupera le informazioni, esegue le azioni corrette

agevolando, e migliora il funzionamento dell'applicazione evitando all'utente di dover intervenire ogni qualvolta venga modificato l'ambiente di esecuzione.

1.6 Introduzione all'architettura

Nelle tecnologie wireless non si è ancora giunti a una unificazione degli dispositivi e quindi c'e la necessita di poter servire cliente molto eterogenei. Non potendo cablare nel server (servitore) tutte le differenze dei vari client (clienti) si necessita di svincolare il servitore dalla conoscenza dello specifico client. Per fare questo si introduce una nuova entità chiamata proxy tra il server e il client. Questa entità avrà una conoscenza delle particolari esigenze del client. Potrà anche adattarsi dinamicamente al cambiamento delle esigenze con l'interazione. Il client dovrà possedere uno stub leggero che raccoglie informazioni sul contesto e informa il proxy al riguardo. E' importante capire che durante l'erogazione del servizio il proxy è l'unica l'entità che modifica il proprio comportamento in modo da adattare il flusso multimediale alle capacità del client. Questo ci consente di non dover mai cambiare un servitore una volta che ci troviamo a dover servire un nuovo tipo di client. Inoltre, si rende più leggera l'implementazione del server e quindi si avrà una maggiore efficienza da parte del server. Il proxy ha il compito di ricevere i dati dal server e inoltrarli al client nella sua specifica modalità di funzionamento. Potendosi adattare al client il proxy può migliorare la qualità di servizio.

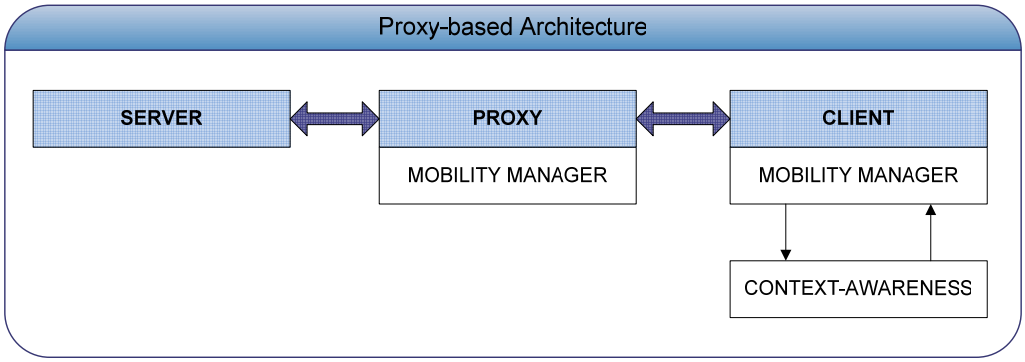

Figura 1.5: Architettura Proxy-based

La Figura 1.5 si basa sulla funzionalità di gestione dell'handoff presente nelle varie entità. Come vediamo sia il client che il proxy hanno un componente chiamato Mobility Manager che si occupa della gestione dell'handoff verticale. Il client ha in più il componente incaricato di recuperare le informazioni di contesto. Nei servizi multimediali basati su tecnologie wireless questa struttura aiuta notevolmente dovendoci essere comunicazione real-time tra il servitore e il cliente. Il server viene completamente svincolato da questa comunicazione con il client delegando al proxy la coordinazione dell'erogazione del servizio. Così facendo si può migliorare la continuità e qualità del servizio facendo adattare il proxy dinamicamente alle necessità del client.

Come già accennato in precedenza l'obbiettivo di questa tesi e di fornire un supporto per la gestione dell'handoff verticale per servizi multimediali tra diverse tecnologie wireless al fine di migliorare la qualità di servizio e ridurre la quantità di dati persi effettuando quando c'e la necessità un handoff verticale.

Il supporto per la gestione dell'handoff verticale si basa sulla possibilità di ricavare informazioni di basso livello sul contesto e sullo stato delle connessioni. Il client può modificare il proprio comportamento in base a queste informazione e può anche far adattare il proxy ai suoi cambiamenti. Applicando l'handoff verticale si cerca di mantenere la continuità di servizio migliorando la qualità della connessione ed evitando possibili interruzioni della rappresentazione dei dati multimediali.

Capitolo 2
Tecnologie Wireless

Le tecnologie che non utilizzano collegamenti fisici per collegare i terminali fra di loro sono denominate tecnologie wireless (senza fili). Per effettuare la comunicazione utilizzano le onde radio o raggi infrarossi e quindi non necessitano collegamenti fisici per il loro funzionamento. Le tecnologie wireless sono abbastanza eterogenee, ognuna con le proprie caratteristiche in base a area di copertura, forza del segnale, frequenza di lavoro, protocollo di comunicazione, ecc. Le più diffuse sono WiFi, Bluetooth, e 3G.

Da tempo è nata la necessità di collegare in rete diversi terminali per poter condividere tra loro risorse e servizi. Prima dell'arrivo delle tecnologie wireless questo si faceva tramite collegamenti fisici (cavi) per il collegamento fra il terminale finale e l'infrastruttura di rete si formavano delle reti locali LAN, Local Area Network. Ovviamente così facendo non si aveva la possibilità di spostarsi da una posizione all'altra durante l'utilizzo del servizio.

Con l'arrivo dei dispositivi wireless e con la possibilità di dare mobilità ai terminali della rete si vengono a formare delle reti wireless. Una rete wireless di ottiene collegando più AP tra di loro. Questi AP si collegano alla rete fissa sottostante fornendo ai terminali mobili tutte le funzionalità dei terminali connessi nella LAN. Esistono diversi tipi di reti wireless: MWLAN (Metropolitan WLAN), WLAN (Wireless LAN), Wireless PAN (Personal Area Network), ecc.

2.1 WiFi

Il marchio "*Wi-Fi*" (Wireless Fidelity) nacque quando la WECA (Wireless Ethernet Compatibility Alliance) introdusse il suo programma di certificazione: ogni strumento che passi i test studiati per garantire il rispetto degli standard può

utilizzare il marchio *Wi-Fi*. In seguito uscì anche il marchio *Wi-Fi5* per indicare quegli strumenti che utilizzando lo standard 802.11a che sfrutta una banda intorno ai 5 GHz. La tecnologia WiFi fa parte dello standard IEEE 802.11 per reti wireless ed esiste correntemente in quattro variazioni, con le loro relative caratteristiche riportate dalla tabella sottostante. Lo 802.11a offre migliori prestazioni rispetto alle 802.11b, ma ha un costo superiore e quindi gli è stata preferita la 802.11b, che è la più diffusa nelle WLAN. Negli ultimi anni si stanno sempre più diffondendo le 802.11g e 802.11n che offrono una maggiore velocità (ampiezza di banda) e qualità sfruttando lo stessa banda di frequenza [WIKI].

Wi-Fi specifications			
Specification	**Speed**	**Frequency Band**	**Compatible with**
802.11b	11 Mb/s	2.4 GHz	B
802.11a	54 Mb/s	5 GHz	A
802.11g	54 Mb/s	2.4 GHz	b, g
802.11n	100 Mb/s	2.4 GHz	b, g, n

La banda di frequenza è centrata attorno ai 2,4 GHz utilizzando tecniche per ottenere una maggiore robustezza nei confronti di segnali interferenti. Tale banda è identificata come **ISM** (Industrial Scientific and Medical purpose) e, tenendo conto delle leggi in vigore nella maggior parte degli stati, per poter trasmettere nella ISM non è necessario ottenere particolari licenze o permessi.

2.1.1 WLAN

Una WLAN (Wireless Local Area Network) può essere vista come un'alternativa o come un'estensione di una normale rete cablata supportando, con un Access Point, la connessione a dispositivi mobili o fissi. Esistono due modalità principali per costituirla delle quali la prima è una rete costituita tra entità "pari", cioè i terminali comunicano direttamente tra di loro senza appoggiarsi ad un Access Point e quindi non hanno bisogno di un'infrastruttura. Questo modo di costituire

una WLAN viene chiamato *Ad-Hoc* oppure *Peer-to-Peer*. La rete nella quale operano viene denominata Indipendent Basic Service Set (IBSS). Un esempio tipico di questa modalità è quando ci sono due terminali e uno si deve connettere all'altro per sfruttare la condivisione della sua connessione LAN.

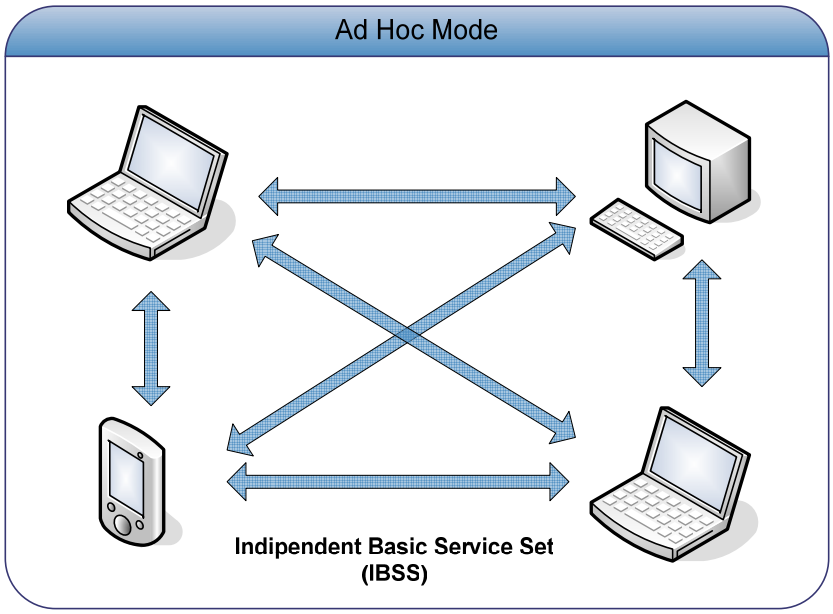

Figura 2.1: Modalità Ad Hoc

Un'altra modalità di costituire una WLAN ed è quella denominata *a infrastruttura*. Si effettua il collegamento (bridging) di una rete WiFi ad una rete Ethernet fissa. In questa modalità i terminali WiFi non possono collegarsi direttamente tra di loro ma devono farlo tramite un Access Point. L'architettura di una rete WLAN a infrastruttura si basa su due entità: Access Point e Mobile Node. I Mobile Node (MN) sarebbero i terminali dotati di schede WiFi che utilizzano i servizi della rete. Mentre l'Access Point (AP) è l'entità che collega i diversi MN ed effettua il bridging alla rete fissa. Tipicamente l'interazione tra le due entità è di tipo Client-Server nella quale più MN si appoggiano a un AP per comunicare con la rete fissa. Invece il Distribution System (DS) è l'entità che consente l'interconnessione di diversi AP [WIKI].

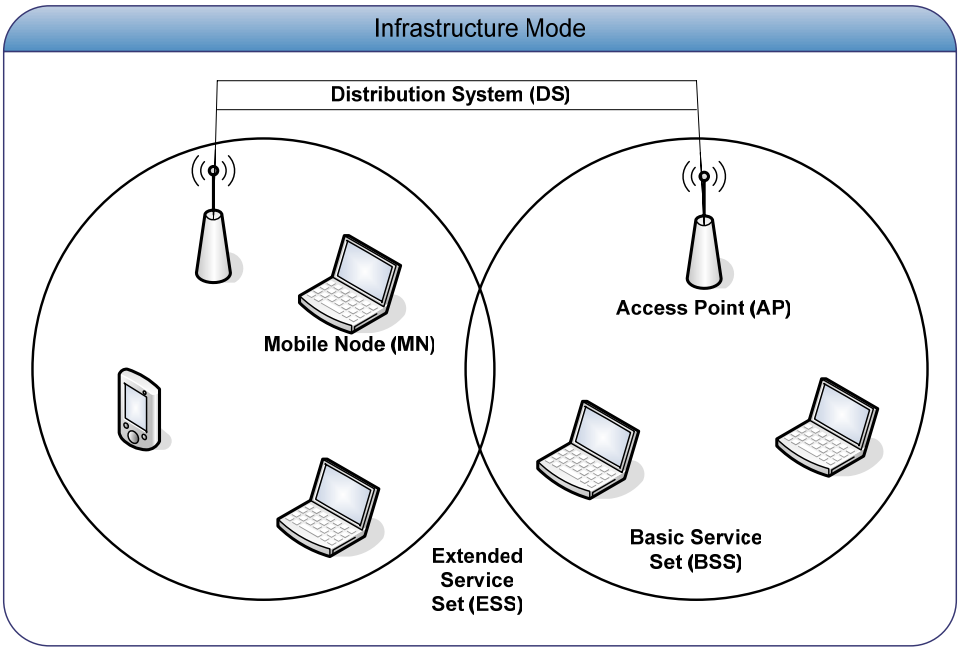

Figura 2.2: Modalità a infrastruttura

2.1.2 BSS e ESS

Una **BSS** (Basic Service Set) è una rete in cui tutti i dispositivi presenti invece di comunicare direttamente tra loro comunicano con un AP. Quindi un dato sarà trasmesso all'AP il quale provvederà ad inviarlo al giusto destinatario. Un AP può eseguire questo procedimento solo se il terminale destinatario si trova nella stessa BSS del mittente.

Una rete BSS può coprire un'area limitata e per questo motivo nasce la necessità di creare più reti BSS che riescano a comunicare tra di loro. Questo si risolve aggiungendo un DS che ha il compito di interconnettere le diverse BSS. Così nasce una rete **ESS** (Extended Service Set), cioè una rete che contiene più BSS interconnesse da un DS.

2.1.3 Servizi del Distribution System

Anche se nello standard 802.11 il DS non è specificato, sono specificati i servizi che il **DS** deve supportare [WIKI]. Questi servizi sono divisi in due sezioni:

- Station Services (SS)
- Distribution System Services (DSS)

La SS vale per entrambe le modalità di costituzione della WLAN mentre la DSS vale solo per le WLAN a infrastruttura.

La DSS offre i seguenti servizi:

- Association
- Reassociation
- Disassociation
- Distribution
- Integration

Questi servizi servono per la gestione delle associazioni che permettono di recapitare correttamente i dati al giusto destinatario. Per fare ciò si utilizza il concetto di associazione (unione dei MAC address del terminale mobile e dell'AP cui è collegato). In particolare il servizio Association è invocato dopo l'autenticazione e permette di creare una nuova associazione. Il servizio Reassociation rende possibile il roaming all'interno di un ESS tra BSS, cioè passaggio tra diverse BSS all'interno di un ESS. Il servizio di Disassociation comporta l'eliminazione dell'associazione. Il servizio di Distribution serve a recapitare correttamente i dati, cioè quando vengono inviati i dati da un terminale a un AP (input AP), questo li passa al DS che li manda all'AP corretto (output AP) che dovrà mandare i dati al terminale che li deve ricevere. Il servizio di Intergration è simile al Distribution, solo che l'output AP è un portale.

La SS offre i seguenti servizi:

- Authentication
- Deauthentication
- Privacy
- MAC Service Data Unit (MSDU) Delivery

Questi servizi hanno come obbiettivo quello di rendere più sicura una rete wireless. Il servizio di Authentication fornisce, ad una postazione, la possibilità di

essere autenticata. Il servizio di Deauthentication è richiesto per revocare l'autenticazione. Il servizio Privacy ha come obbiettivo quello di rendere sicuro il canale wireless mediante appropriati algoritmi. Il servizio MSDU (MAC Service Data Unit) scambia l'unità dati tra due postazioni wireless. Questo servizio è sufficiente per lo scambio di informazioni in reti ad hoc, mentre in una Infrastructure LAN sono necessari anche i servizi della DSS.

2.1.4 Confronto tra WLAN e LAN

Avendo a che fare con una rete wireless (WLAN) ci viene da chiedersi quali sono i principali vantaggi e svantaggi rispetto a una rete fissa (LAN) [WIKI].

Vantaggi:

- *Flessibilità*: la natura stessa della rete wireless, cioè la flessibilità della rete che si ha non avendo vincoli di una postazione fissa, anche durante l'esecuzione del servizio. L'unico limite di mobilità è la limitazione dell'area di copertura dell'AP;
- *Pianificazione*: è ovvio che l'installazione di ogni tipo di rete deve essere pianificata ma, a differenza delle LAN, nelle WLAN la disposizione della rete si può modificare e quindi anche correggere una pianificazione non ottima;
- *Settaggio dell'equipaggiamento:* la WLAN, oltre a poter essere quasi invisibile, può essere implementata anche in posti dove è difficile o molto costoso implementare reti fisse, ad esempio spazi aperti, edifici storici, musei, fiere, ecc;
- *Robustezza:* Se un'AP si spegne gli utenti possono muovere il loro terminali nel range di un altro AP. Non c'e bisogno neanche di muovere i terminali se è presente una sovrapposizione di aree di copertura degli AP. Questa sovrapposizione deve essere ben pianificata per evitare interferenze del segnale dei diversi AP. Ovviamente le performance dell'AP diminuiscono con l'aumentare degli utenti presenti nella cella;

- *Trasparenza delle applicazioni:* le applicazioni che funzionano su rete fissa funzionano anche su rete wireless, pur avendo possibili problemi introdotti dai delay più lunghi e banda meno stabile;
- *Prezzo:* con il diminuire del prezzo dei componenti wireless negli ultimi anni diventa più economica una rete wireless di una fissa.

Svantaggi:

- *Sicurezza dati:* esistono notevoli problemi di sicurezza con il protocollo Wired Equivalent Privacy (WEP). Questo problema ha causato ad esempio il Wardriving, cioè muoversi in machina alla ricerca di reti wireless con lo scopo di recuperare i dati o informazioni che ci transitano. Un protocollo migliore di sicurezza è il WiFi Protected Access (WPA), ma ha ancora un livello di sicurezza inferiore alle LAN, dato dall'accesso condiviso da tutti gli utenti al mezzo fisico;
- *Sicurezza:* esistono posti come ad esempio ospedali che hanno attrezzature che potrebbero subire interferenze da parte della rete wireless e non funzionare correttamente.
- *Limitazioni dovute alle onde radio:* qua ricadono tutti i problemi delle comunicazioni radio, ad esempio l'esistenza di muri o piani degrada il segnale e la comunicazione, oppure interferenza dovuta a più AP sovrapposti.
- *Velocità di trasferimento dati:* normalmente le reti wireless avranno velocità minori rispetto alle reti fisse, dovendo tutti gli utenti dividere la velocità del AP (tipicamente 11Mb/s o 54Mb/s).
- *Standardizzazione:* Per adesso non si è ancora giunti a una standardizzazione delle reti wireless, ma esiste ancora molta eterogeneità fra le diverse tecnologie.

Abbiamo notato che la rete fissa è più sicura, più affidabile e ha maggiore velocità di trasferimento dati. La rete wireless è più conveniente nei casi dove è richiesta la mobilità o dove non sia possibile implementare una rete fissa.

2.2 Bluetooth

Le specifiche della tecnologia Bluetooth sono state introdotte da Ericsson e poi formalizzate dalla Bluetooth Special Interest Group (SIG). La SIG è stata formalmente annunciata il 21. maggio 1999. ed era formata da Sony Ericsson, IBM, Intel, Toshiba e Nokia. La versione più recente di Bluetooth è la 2.0, specificata dalla SIG nel 2004 [WIKI].

Bluetooth è una tecnologia wireless che utilizza onde radio a basso costo avendo un area di copertura molto limitata. Il raggio di copertura dipende dalla potenza del segnale. Ne esistono tre classi diverse ognuna con le proprie caratteristiche, come vediamo dalla tabella sottostante:

Class	Power	Range
1	100 mW	100 m
2	2.5 mW	10 m
3	1 mW	10 cm – 1 m

Lo classe più diffusa è la 2 ed è maggiormente usata per formare reti wireless locali a corto raggio, premettendo di trasferire voce e dati. Queste reti vengono denominate PAN (Private Area Network) o meglio WPAN (Wireless PAN) e sono composte da dispositivi di dimensioni ridotte (computer desktop e notebook, cellulari, palmari, stampanti, ecc). Tutti questi dispositivi hanno in comune il basso consumo di energia in quanto utilizzano una potenza di trasmissione molto bassa, non esigono connessioni veloci, e non necessitano di connessioni ad una rete locale.

2.2.1 PAN Bluetooth

La Personal Area Network è una rete ad hoc a corto raggio tra diversi dispositivi. Può essere usata per scambiare dati interconnettendo diversi dispositivi o per collegarsi a un dispositivo con un'altra rete (WLAN, LAN, ecc). Una Wireless PAN può essere implementata con Bluetooth o IrDA (infrarossi) [WIKI].

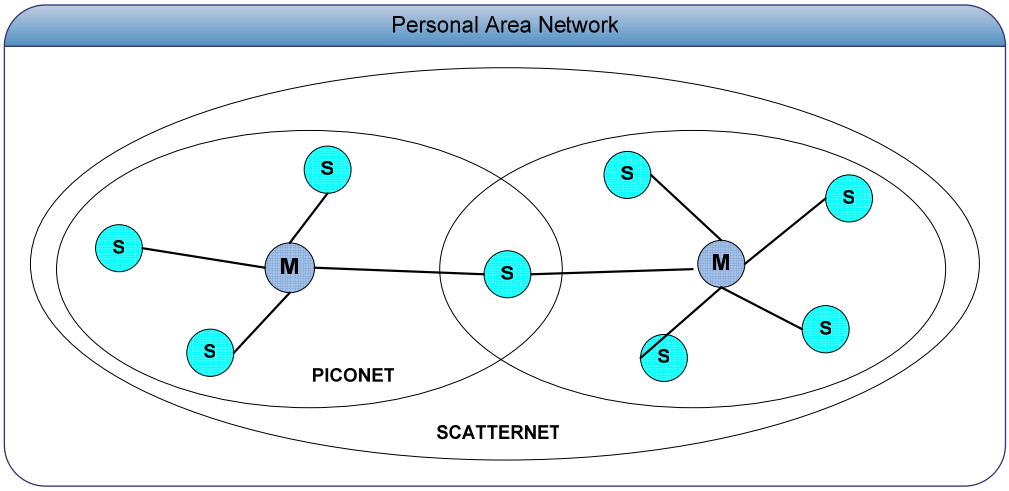

Figura 2.3: PAN

Una PAN Bluetooth è ulteriormente chiamata *piconet* ed è composta da, al massimo, otto dispositivi collegati fra di loro con relazione master-slave (dovuto all'identificatore del dispositivo nella piconet di lunghezza 3 bit). Inoltre possono esserci fino a 255 dispositivi in modalità parked, cioè passiva, e la loro attivazione sarà gestita dal master. Il primo dispositivo è il master e tutti gli altri sono slave che comunicano con il master. I dati possono essere trasferiti tra un master e un slave, ma il master effettua uno rapido switch (cambiamento) da slave a slave così da gestirli tutti. Questo switch è effettuato in modalità Round-Robin, cioè tutti i slave hanno la stessa priorità e cambiano al passare di un certo intervallo di tempo in sequenza. Il trasferimento dei dati da un master a più slave simultaneamente è possibile ma in pratica non viene usato. Se due piconet hanno dispositivi condivisi allora si forma una s*catternet*. Ogni dispositivo può in ogni momenti cambiare il proprio ruolo master/slave.

2.2.2 Funzionamento Bluetooth

Una delle principali caratteristiche di Bluetooth è quella di dover funzionare in qualunque parte del mondo e per questo motivo la banda di frequenza su cui lavora è di 2,4 GHz. Questa banda di frequenza è utilizzata anche dallo standard IEEE 802.11 e questo causa interferenze nella comunicazione Bluetooth. Per risolvere questo tipo di problemi è adottata una tecnica chiamata FHSS (Frequency Hopping Spread Spectrum), che divide la banda in tanti canali di "salto". Durante una trasmissione, trasmittente e ricevente saltano in frequenza da un canale all'altro secondo una sequenza pseudocasuale, ciò assicura una bassa interferenza sulla comunicazione ed inoltre il protocollo prevede degli algoritmi per la correzione degli errori. Inoltre i dispositivi dotati di tecnologia Bluetooth comunicano tra loro creando e riconfigurando dinamicamente la rete. Ciò rende possibile, ad esempio, sincronizzare i dati di un Pc portatile e di un PDA semplicemente avvicinando i due dispositivi oppure passare in modo automatico al vivavoce in macchina quando si entra e si parla al cellulare. Tutto ciò è possibile grazie all'*SDP* (Service Discovery Protocol) che permette ad un dispositivo Bluetooth di determinare i servizi che gli altri dispositivi Bluetooth mettono a disposizione. In questo modo ogni nodo può conoscere le funzioni ed i servizi di ogni altro nodo nella rete, utile nella sincronizzazione.

Il protocollo *SDP* permette di determinare i servizi presenti e disponibili in una piconet sia in modalità client sia in modalità server. La modalità server consente le interrogazioni sui servizi e protocolli supportati e li renderà disponibili; la modalità client invece consente l'interrogazione dei dispositivi connessi alla piconet per ottenere informazioni.

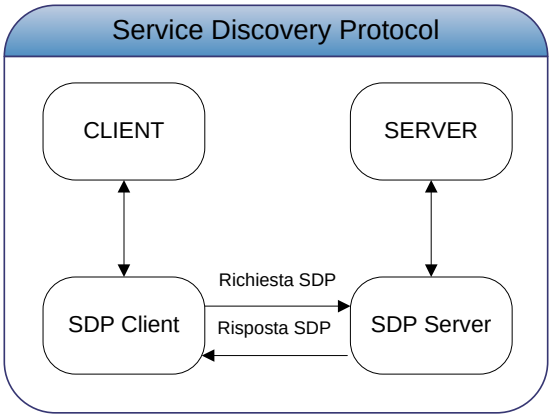

Figura 2.4: SDP

La tecnologia Bluetooth fornisce servizi di rete ai livelli fisico e data-link del modello ISO/OSI di riferimento per le reti di calcolatori, e definisce le specifiche di uno stack di protocolli anche per i livelli più alti come riportato di seguito.

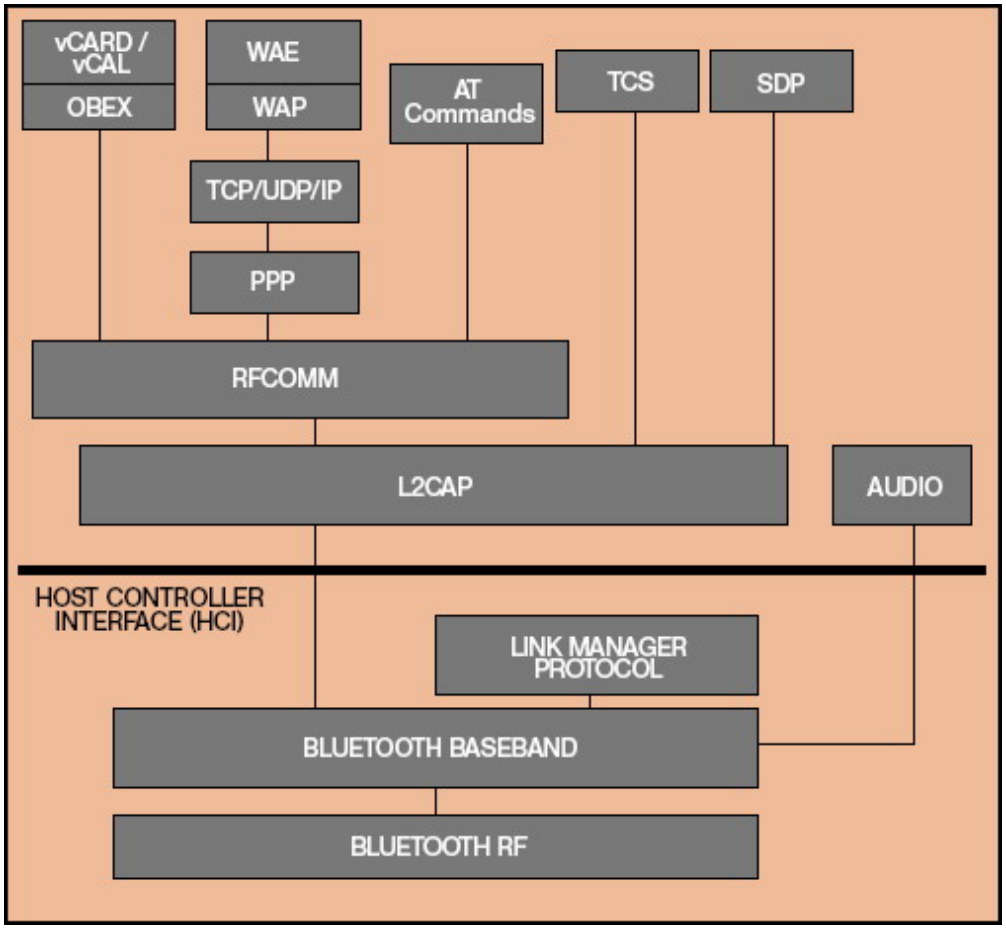

Figura 2.5: Bluetooth Protocol Stack

Protocol Layer	Protocols in the stack
Bluetooth Core Protocols	Baseband, LMP, L2CAP, SDP
Cable Replacement Protocol	RFCOMM
Telephony Control Protocol	TCS Binary, AT-commands
Adopted Protocols	PPP, UDP/TCP/IP, OBEX, WAP, vCard, vCal, IrMC, WAE

In aggiunti ai protocolli sopra riportati [BPA/00]., le specifiche definiscono l'*HCI* (Host Controller Interface) che offre un interfaccia di comando per il controllore baseband, il link manager e l'accesso allo stato dell'hardware e i registri di controllo.

I protocolli base sono richiesti e utilizzati tutti i dispositivi Bluetooth, mentre gli altri sono utilizzati solo quando servono. Questi protocolli sono:

- *Radio Bluetooth*: specifica i dettagli di trasmissione delle onde radio nell'aria quali banda delle frequenze del segnale, uso di salti in frequenza, schema di modulazione e demodulazione, ecc;
- *Baseband***:** consente il link fisico tra dispositivi bluetooth della piconet. Questo link può essere Synchronous connection oriented (SCO) e Asynchronous connectionless (ACL). Con l'ACL vengono mandate solo dati mentre con l'SCO viene mandato solo audio oppure audio e dati;
- *LMP (Link Manager Protocol)*: attiva il collegamento fra dispositivi Bluetooth e gestisce il collegamento. Gestisce anche aspetti quali l'autenticazione, la crittografia, il controllo e la negoziazione delle dimensioni dei pacchetti in banda base;
- *L2CAP (Logical Link Control and Adaptation Layer):* adatta i protocolli superiori al livello in Banda Base. Fornisce servizi orientati alla connessione a livello di trasporto (end-to-end);
- *SDP (Service Discovery Protocol):* oltre a fornire la loro ricerca fornisce le informazioni sui dispositivi, sui servizi e sulle caratteristiche sulle quali

possono essere interrogati per consentire l'attivazione/disattivazione di una connessione fra più dispositivi Bluetooth.

Il protocollo *RFCOMM* emula le caratteristiche della porta seriale RS-232 e rende più trasparente la sostituzione delle tecnologie di trasmissione via cavo. Fornisce un flusso di dati affidabile e controllato e consente connessioni concorrenti multiple. Nello stack sono anche presenti protocolli per la telefonia quali *TCS Binary* (Telephony Control Specification Binary), che definisce i segnali di controllo della chiamata per l'attivazione di collegamenti voce e dati fra dispositivi Bluetooth, e *AT-commands* (Attention commands – parte del Hayes Command Set**),** protocollo di testo che consente la a un DTE (Data Terminal Equipment) di comunicare con un DCE (Data Communication Equipment).

Inoltre sono presenti protocolli adottati da altre architetture come il *PPP* (Point to Point Protocol) e *TCP/UDP/IP* da Internet, *OBEX* per il trasferimento di file e i protocolli tipicamente wireless quali *WAE/WAP*.

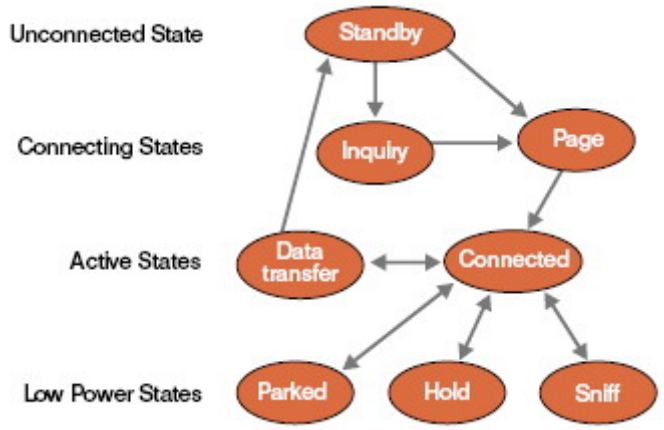

Figura 2.6: Stati di funzionamento

In questa figura vediamo i possibili stati di funzionamento di un dispositivo Bluetooth. Generalmente in un collegamento, tutti gli apparecchi Bluetooth sono in modalità standby, cioè attesa, ed effettuano una scansione ogni 1,28 secondi per verificare la presenza di eventuali altri dispositivi. In questa modalità i dispositivi

Bluetooth sono a basso consumo energetico. I tipi di scansione che possono essere effettuati sono: *Page Scan* (PE) e *Inquiry Scan* (IS).

La scansione PE consente di ricercare un collegamento con un altro apparecchio Bluetooth e può essere in modalità *connectable mode* o *non-connectable mode.*

La scansione IS è simile alla PE e permette di identificare la tipologia di dispositivi presenti nella piconet e di approntare i necessari protocolli per il collegamento. Il comando inquiry è utilizzato quando l'indirizzo o il numero di identificazione di un dispositivo non è conosciuto e dopo aver effettuato il riconoscimento seguirà un comando di page che è utilizzato per risvegliare l'altro dispositivo ed instaurare una connessione.

Una scansione può avere diversi risultati quali:

- *Active*: la connessione risulta essere attiva e può avvenire la trasmissione e la ricezione dei dati, gli slave sono sincronizzati con il master;
- *Hold*: può svolgere operazioni di IS e PS con basso consumo energetico;
- *Sniff*: riduce il carico di lavoro in modalità di ascolto della piconet;
- *Parked*: è la modalità di attesa, rimanendo sincronizzato con la piconet.

2.2.3 Confronto tra Bluetooth e WiFi

Bluetooth e 802.11b ovvero WiFi sono tecnologie wireless eterogenee. Nella tabella [BVW/00] sono ripportate le differenze delle caratteristiche delle due tecnologie:

Tecnology	**Bluetooth**	**802.11b**
Access	Point to Multipoint	Point to point only
Bit Rate	1 Mbps	11Mbps
Spread Spectrum	FHSS	DSSS
Range	10/100 m	150/300 m

Profiles	Almost unlimited	LAN station or access point
Data Distribution	No restriction	Access point only
Current Consumption	60mA	300mA
Audio	PCM channels	Voice over 802.3
Cable Replacement	Serial, USB, UART, Audio	802.3
Horizontal Handoff	Hard, Soft	Hard
Mobility Management	Master	Mobile Station

I svantaggi della tecnologia Bluetooth sono sicuramente la velocità del trasferimento dei dati e il raggio d'azione dei dispositivi. Inoltre i dispositivi Bluetooth sono solo adatti per formare reti di dimensioni minori, cioè PAN. Utilizzando poca potenza del segnale la tecnologia Bluetooth potrebbe essere oscurata se in sovrapposizione con altre onde radio della stessa frequenza e di segnale più forte.

I vantaggi di Bluetooth rispetto a WiFi sono nel minor costo dei dispositivi, migliore gestione della sicurezza, minor consumo di energia e di essere completamente mobili, cioè non hanno un'infrastruttura dalla quale dipendono come in WiFi. Infatti due dispositivi Bluetooth si possono sempre collegare fra di loro, ogni dispositivo può funzionare sia essere in modalità master che in modalità slave.

2.3 Sistemi wireless eterogenei

Con la necessità negli utenti di essere svincolati da una postazione fissa di utilizzo della rete si diffondono sempre di più le tecnologie wireless che offrono questa funzionalità. Lo sviluppo delle tecnologie wireless non era da subito standardizzato e quindi si sono venute a formare molte tecnologie wireless

eterogenee. Queste tecnologie hanno caratteristiche molto diverse fra loro e quindi sono più o meno adatte a specifici contesti d'uso. Si intuisce che riuscire a intergare queste tecnologie eterogenee potrebbe portare al miglioramento dei servizi offerti. Il tutto porta ad introdurre il concetto di *mobilità verticale*.

La figura sottostante rappresenta la tassonomia degli aspetti tecnici legati alla mobilità verticale che si può suddividere in tre parti: gestione delle risorse, ingegnerizzazione della mobilità e la gestione dei servizi [VHM/05].

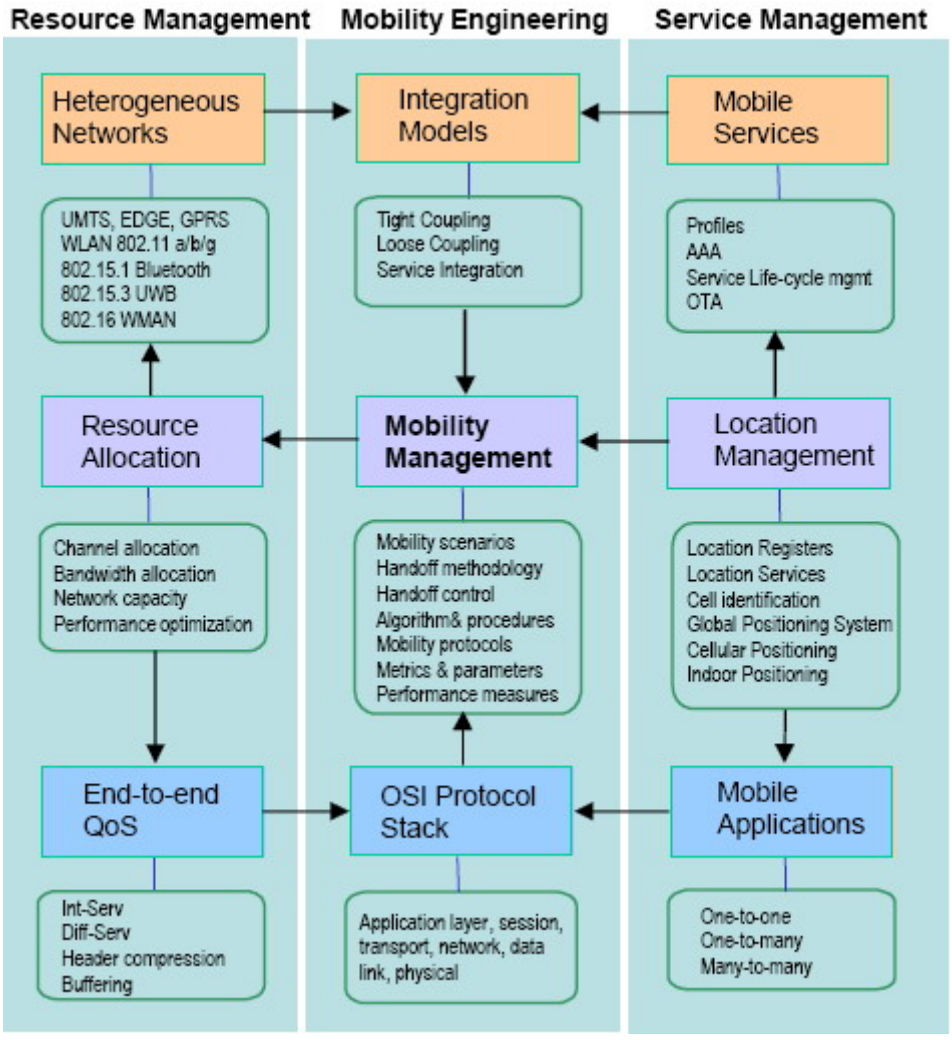

Figura 2.7: Tassonomia degli aspetti tecnici

La gestione delle risorse comprende l'allocazione diretta (canali e banda) e indiretta (ottimizzazione della capacità e delle performance della rete) delle risorse

in una rete con più tecnologie eterogenee. L'ingenerizzazione della mobilità comprende l'intergazione degli accessi e servizi su reti eterogenee, gestione della mobilità, design e implementazione dei vari protocolli e middleware nei differenti strati dell'OSI. Il core di questa parte è la gestione della mobilità, che si basa su diversi aspetti riguardanti gli handoff : strategie di controllo, algoritmi, misurazione delle performance, metodologie, metriche e parametri di mobilità. La gestione dei servizi include i diversi servizi offerti.

Nella figura sottostante vediamo la gestione della mobilità più in dettaglio:

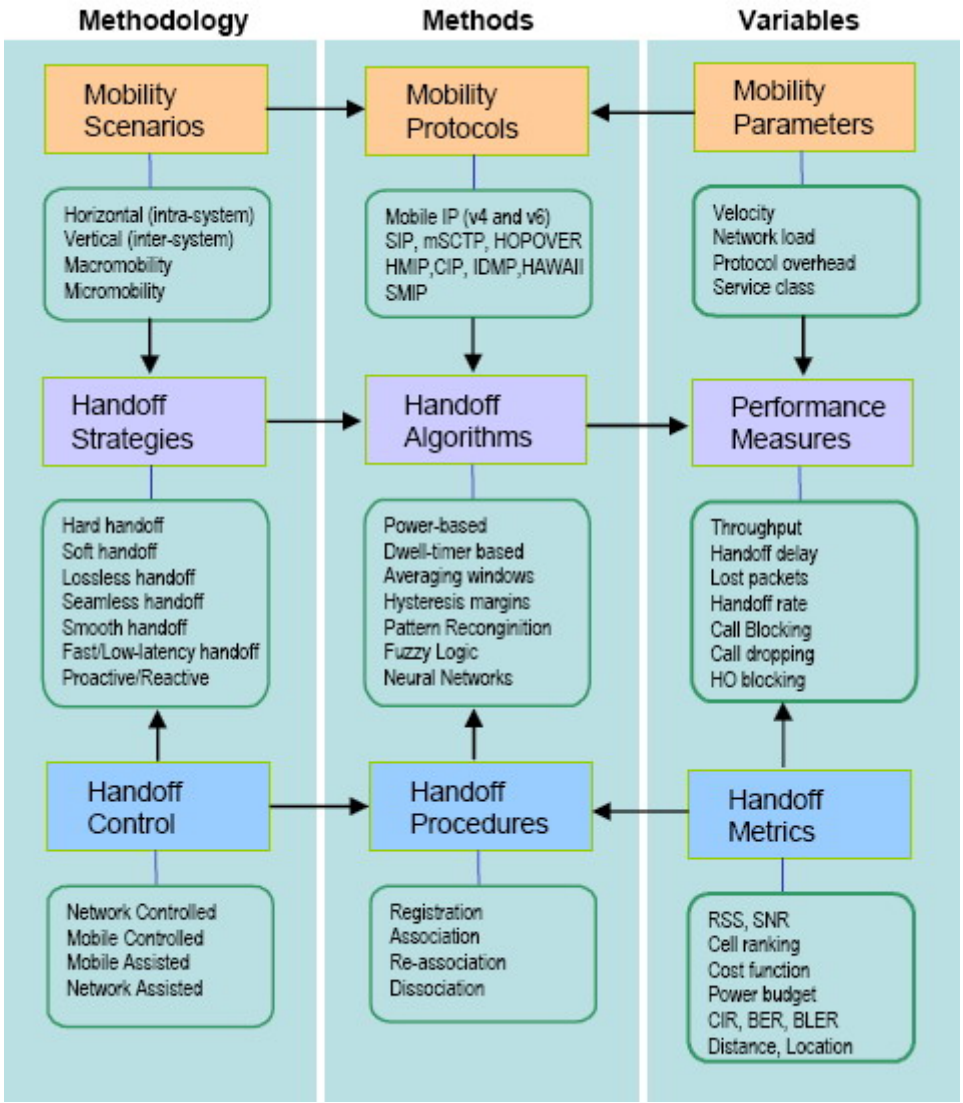

Figura 2.8: Gestione della mobilità

2.3.1 Handoff verticale

L'handoff è l'evento che si verifica quando un terminale mobile si muove da una cella a un'altra. Può essere classificato come:

- *orizzontale* (intra-system): indica l'handoff entro la stessa tecnologia wireless;
- *verticale* (inter-sistem): indica l'handoff tra tecnologie wireless eterogenee.

Il controllo dell'handoff può essere contenuto in un'entità della rete o nel terminale stesso. Questi casi vengono denominati rispettivamente *network executed* handoff (NEHO) e *mobile executed* handoff (MEHO). Esistono anche soluzioni miste dove una entità aiuta l'altra e vengono denominate *mobile assisted* handoff (MAHO), dove il terminale mobile aiuta l'entità nella rete a decidere, e *network assisted handoff* (NAHO), dove l'entità della rete aiuta il terminale mobile a decidere [VHM/05].

Il processo dell'handoff può essere caratterizzato come:

- *Hard:* si riferisce a un'interruzione della comunicazione durante lo svolgimento dell'handoff (brake-before-make);
- *Soft:* si mantiene la connessione a entrambe le stazioni base durante lo svolgimento dell'handoff in modo da evitare l'interruzione della comunicazione (make-before-brake).

In riferimento alla terminologia dello strato di rete è possibile una ulteriore classificazione in:

- *Lossless handoff*: che indica che non vengono persi pachetti durante l'handoff;
- *Fast handoff:* si riferisce a una latenza bassa dei pacchetti;
- *Seamless handoff:* indica che il passaggio a una nuova rete è del tutto trasparente per l'utente.

Esiste una ulteriore suddivisione della gestione, sia per l'handoff orizzontale sia per quello verticale, in:

- *Reactive*: con questa gestione, cercando di minimizzare il numero degli handoff, lo si esegue solo quando viene persa la connessione con l'AP attuale. Ovviamente questo porta tempi di handoff più lunghi dato che prima di riconnettersi ad un AP dovrà trovarlo effettuando una ricerca;
- *Proactive*: con questa gestione si avvia la procedura di handoff prima che il segnale con AP attuale venga perso, questo viene fatto appena si trova un'AP con un segnale migliore di quello attuale. In questo modo diminuiscono i tempi dell'handoff, ma aumentano di numero.

Un handoff verticale è ulteriormente classificabile in un handoff:

- *verso l'alto*: un'handoff verso una tecnologia con la dimensione delle celle superiore (es. da Bluetooth a WiFi);
- *verso il basso*: un'handoff verso una tecnologia con la dimensione delle celle inferiore (es. da WiFi a Bluetooth).

Un importante aspetto da considerare nella decisione di eseguire un handoff è ottimizzare le performance di esecuzione dell'handoff. Si deve stabilire il momento adatto per scatenare l'handoff. Queste decisioni sono possibili grazie ad un *predittore*, cioè l'entità che riesce a prevedere dove e quando si verificherà un handoff orizzontale. Senza il predittore non sarebbero possibili gestioni di tipo proactive che possono "preparare" i terminali a gestire l'handoff in modo opportuno. Un esempio è la possibilità di allargare il proprio buffer per compensare una possibile discontinuità di connessione durante l'handoff. Questo offre al terminale di poter mantenere la continuità di servizio anche con discontinuità di connessione più lunghe.

Il predittore usa diversi algoritmi che usano metriche per decidere il momento e l'handoff opportuno. La maggior parte degli algoritmi prende principalmente in considerazione l'RSSI (potenza del segnale ricevuto) per prendere le sue

decisioni, ma sono è possibile anche appoggiarsi su altri fattori come la locazione, la larghezza della banda, la saturazione della rete, ecc.

A differenza degli algoritmi di gestione degli handoff orizzontali che sono ben ricercati e chiari, i quelli per gli handoff verticali sono ancora in fase di ricerca e quindi ci sono molte soluzioni specifiche.

Prendere decisioni relative all'handoff verticale è più complesso rispetto al prendere decisioni in merito all'handoff orizzontale. Un metodo di valutazione per l'handoff verticale richiede infatti la conoscenza di diversi parametri in modo da poter confrontare due reti senza fili, inoltre occorre evitare l'effetto ping-pong, fenomeno relativo al terminale mobile che effettua continuamente l'handoff tra due stazioni base. In ambienti omogenei, se la decisione di handoff è presa solo in base all'RSSI, l'effetto ping-pong si può verificare sul bordo dell'area coperta da due stazioni base. Questo si può verificare analogamente negli ambienti eterogenei se viene preso in considerazione un RSSI simbolico (l'RSSI reale viene astratto suddividendolo in più fasce).

Un'ulteriore considerazione da fare è sul consumo di batteria dato che i terminali mobili sono, di solito, dotati di una quantità di energia consumabile limitata. Una gestione di tipo hard riduce notevolmente il consumo di batteria, dato che durante la gestione dell'handoff verticale ha attiva solo una tecnologia e non entrambe come nel caso di una gestione di tipo soft.

Capitolo 3

Architettura preesistente

In questo capitolo sarà introdotta l'architettura preesistente che si basa su tre entità principali: il server, il proxy e il client. Il compito del server è quello di fornire i dati multimediali necessari a fornire il servizio richiesto dal client. Il client invece è l'entità che richiede e beneficia del servizio. Tra queste due entità è stata introdotta l'entità del proxy che ha il compito di ricevere i dati multimediali dal server e di inviarli al client. Inoltre ha il compito di gestire tutte le richieste del client. Questo rende possibile un'implementazione meno costosa e più efficiente del server che viene svincolato dalla conoscenza dei dettagli di funzionamento dei specifici client; questa scelta architetturale consente di fare fronte alla elevata eterogeneità dovuta all'utilizzo di client wireless potenzialmente molto diversi fra loro. Infine, nell'implementazione preesistente, WiFi era l'unica tecnologia wireless supportata.

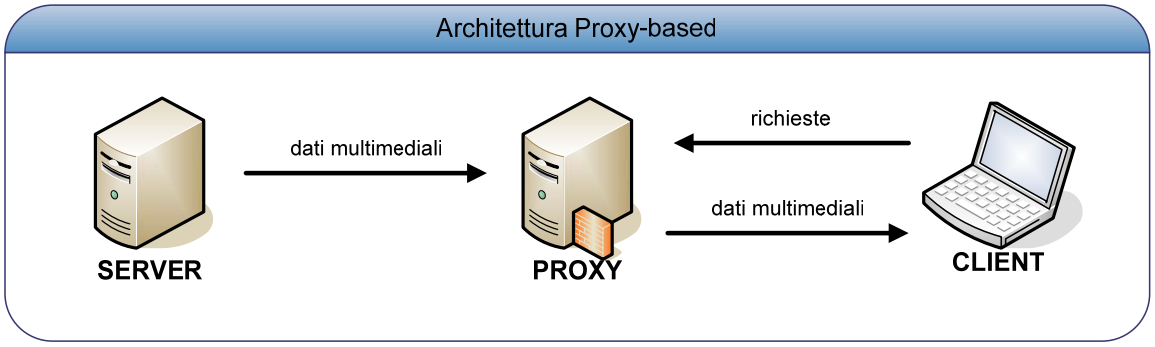

Figura 3.1: Architettura Proxy-based

3.1 Server

Il server, nello scenario descritto, deve soddisfare delle funzionalità piuttosto limitate in quanto la sua principale caratteristica deve essere quella di poter soddisfare più richieste contemporanee possibili e di assicurare un flusso continuo di dati. In particolare, non dovendo trattare direttamente l'adattamento del contenuto erogato, funzionalità svolta dal proxy, deve solo assicurarsi che, una volta ricevuta la richiesta di trasmissione di dati multimediali, l'erogazione avvenga in maniera continua e costante. Avendo una comunicazione di tipo real-time, i dati multimediali ricevuti vengono quasi immediatamente riprodotti e questo ci fa capire l'importanza che i dati arrivino in continuazione e preservando la frequenza di arrivo dei dati multimediale (es. framerate del video).

3.2 Proxy

Il proxy è il componente che si interpone fra il server e il client. Il compito principale del proxy è gestire il flusso di dati multimediali proveniente dal server e inviarlo opportunamente sul client. Inoltre, deve potersi adattare a molti client eterogenei, quindi deve avere la possibilità di modificare il proprio funzionamento adeguandosi alle necessità del client. Come abbiamo gia accennato è proprio il proxy a gestire tutte le richieste dei client svincolando così il server da questo compito. Il proxy provvederà a propagare le richieste al server per richiedere i dati multimediali richiesti dal client.

Il proxy può essere suddiviso in due parti principali: un proxy generale e un proxy specifico per ogni client.

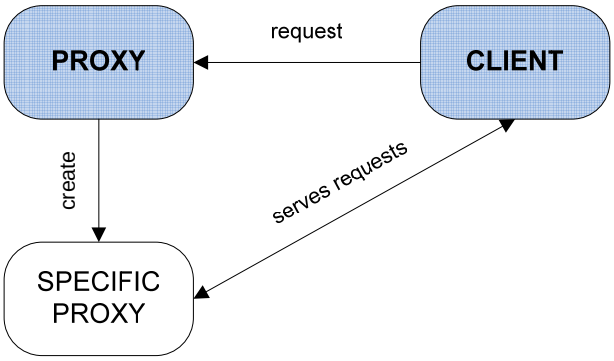

Figura 3.2: Interazione tra proxy e client

Come vediamo dalla figura il client invia la prima richiesta al proxy generale ed esso provede a creare un proxy specifico per quel client e gli delega il compito di servire quel specifico client per tutta la durata del servizio. Da questo momento il client invierà le proprie richieste al proxy specifico.

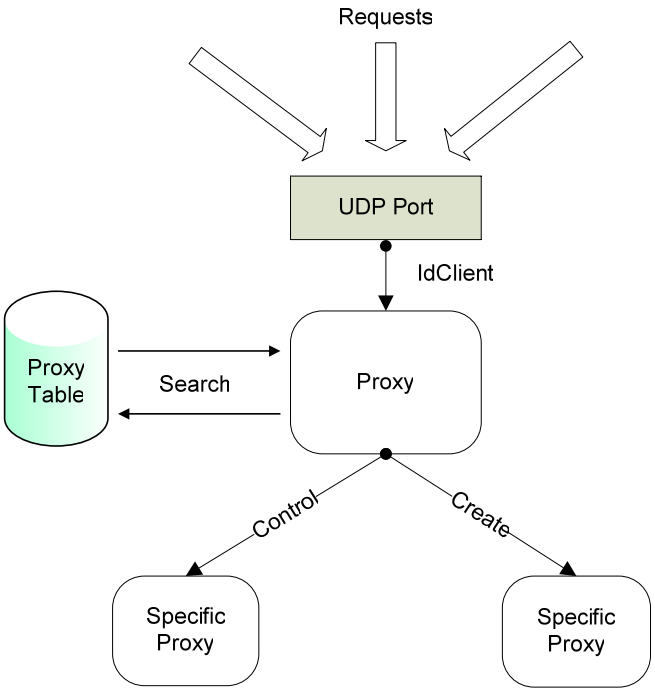

Figura 3.3: Gestione dei proxy specifici

Nella figura viene riportato come viene gestita delega dal proxy generale al proxy specifico. All'arrivo della richiesta del client il proxy generale identifica il cliend dal suo ID (identificatore univoco) e controlla nella tabella se esiste già un proxy

specifico che serve quel specifico client. Se il proxy specifico per quel client esiste già allora lo riassocia al client, viceversa se il proxy specifico per quel client non esiste allora ne crea uno nuovo e lo associa al client. La tabella è implementata dal componente MemoryBroker e contiene i riferimenti a tutti i proxy specifici indicando quale client servono. Mentre il proxy generale è implementato dal componente MainProxy.

3.2.1 ProxyThread

Il ProxyThread è il componente che implementa il proxy specifico ed è il componente che gestisce l'interazione con lo specifico client. Questo componente viene creato dal proxy generale all'arrivo del nuovo client e viene assegnato a servire le richieste solo per quel client.

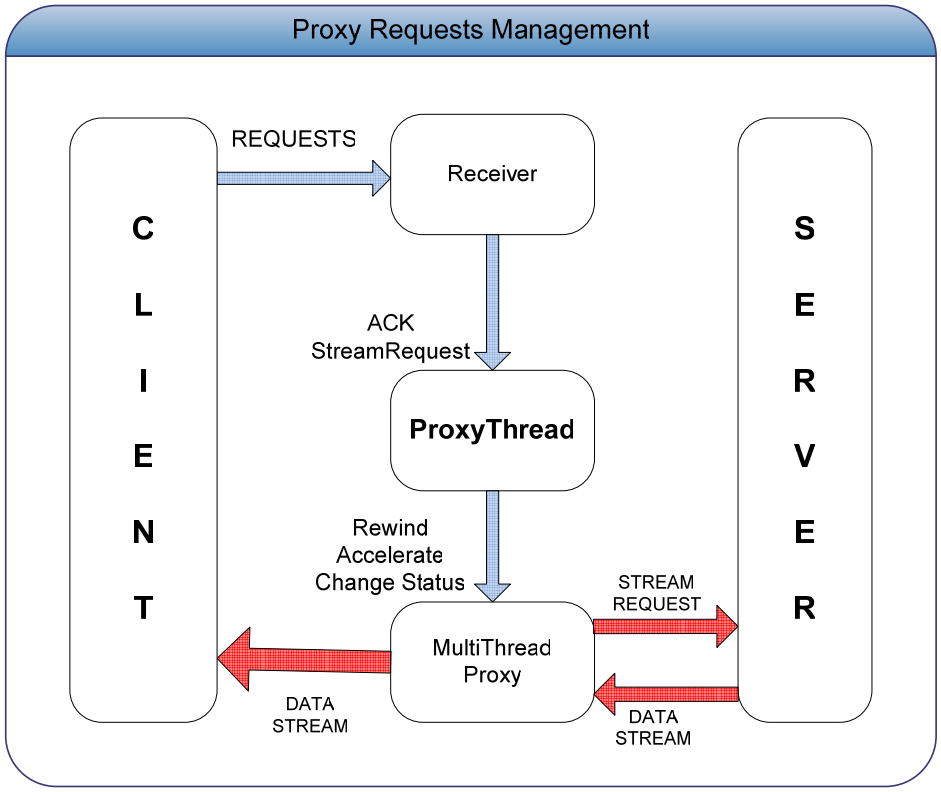

Figura 3.4: Gestione delle richieste nel proxy

Nella figura sono illustrate le richieste del client e la loro gestione nel proxy specifico. Per richiedere l'invio del flusso di dati multimediali il client invia al proxy la richiesta di *Stream Request* che il proxy gestirà inoltrando i dati, richiesti e ricevuti dal server, al client. Ogni proxy specifico ha un buffer dove memorizza i dati ricevuta dal server e poi da esso preleva i dati da inviare al client. L'interposizione di tale buffer, che si va ad aggiungere al buffer abitualmente presente sul client, serve per fare fronte alle perdite di dati dovute agli handoff orizzontali; in questo modo infatti il proxy è in grado di "riavvolgere" il flusso di dati per ripartire la trasmissione dall'ultimo dato ricevuto dal client prima della disconnessione dovuta all'handoff. Questa richiesta prende il nome *Rewind* e viene effettuata dal client non appena viene ripristinata una connessione a seguito di un'handoff orizzontale. La gestione del rewind viene effettuata dal client per semplificare il proxy, cioè per evitare al proxy di dover periodicamente controllare se i dati che manda arrivino veramente al client. In questo modo è il client a doversi preoccupare dei dati multimediali in arrivo e della loro consistenza.

Oltre a ciò, la dimensione del buffer non è fissa, ma può variare nel tempo adattandosi al contesto del client. La variazione della dimensione del buffer ci offre la possibilità di gestire al meglio le risorse del sistema, ad esempio la memoria del proxy. Il proxy in una situazione di stabilità di connessione mantiene bassa la dimensione del buffer, mentre con l'arrivo di un possibile handoff allarga il buffer fornendo la possibilità di un rewind maggiore. Questo per evitare che i dati presenti nel buffer del proxy non siano sufficienti per effettuare il rewind. Ad esempio, il ridimensionamento del buffer del proxy si verifica al cambiamento della probabilità di handoff orizzontale del client e viene gestita con la richiesta *Change Status*. Viene poi data la possibilità di cambiare frequenza di invio dei dati dal proxy al client con la richiesta di *Accelerate*. Questo può essere utile nel caso, dopo un handoff orizzontale il client abbia consumato molti dei pacchetti, con il pericolo di svuotare il proprio buffer e fermare la riproduzione dei dati, e voglia riempire velocemente il proprio buffer. Ovviamente una volta ritornato a una

situazione normale il client chiedera al proxy di rallentare ad una frequenza di invio normale. Infine la richiesta di *Acknowledgement* (ACK) non è una vera richiesta, ma in pratica è una segnalazione di presenza del client. Se il proxy non riceve l'ACK per un tempo maggiore a 2,5 volte l'intervallo previsto tra un ACK e l'altro invoca la funzione di chiusura sul Proxy.

Oltre a gestire le richieste del client il proxy specifico contiene due importanti entità: UDPReceiver e MultiThreadProxy. Utilizza il componente UDPReceiver per ascoltare le richieste del client. UDPReceiver è l'entità attiva che ascolta le richieste in arrivo dal client. All'arrivo di una richiesta, UDPReceiver invoca il metodo di gestione della richiesta del ProxyThread passandogli la richiesta stessa. Questa viene servita dal ProxyThread oppure passata al MultiThreadProxy che provvederà a servirla, come evidenziato in figura 3.4.

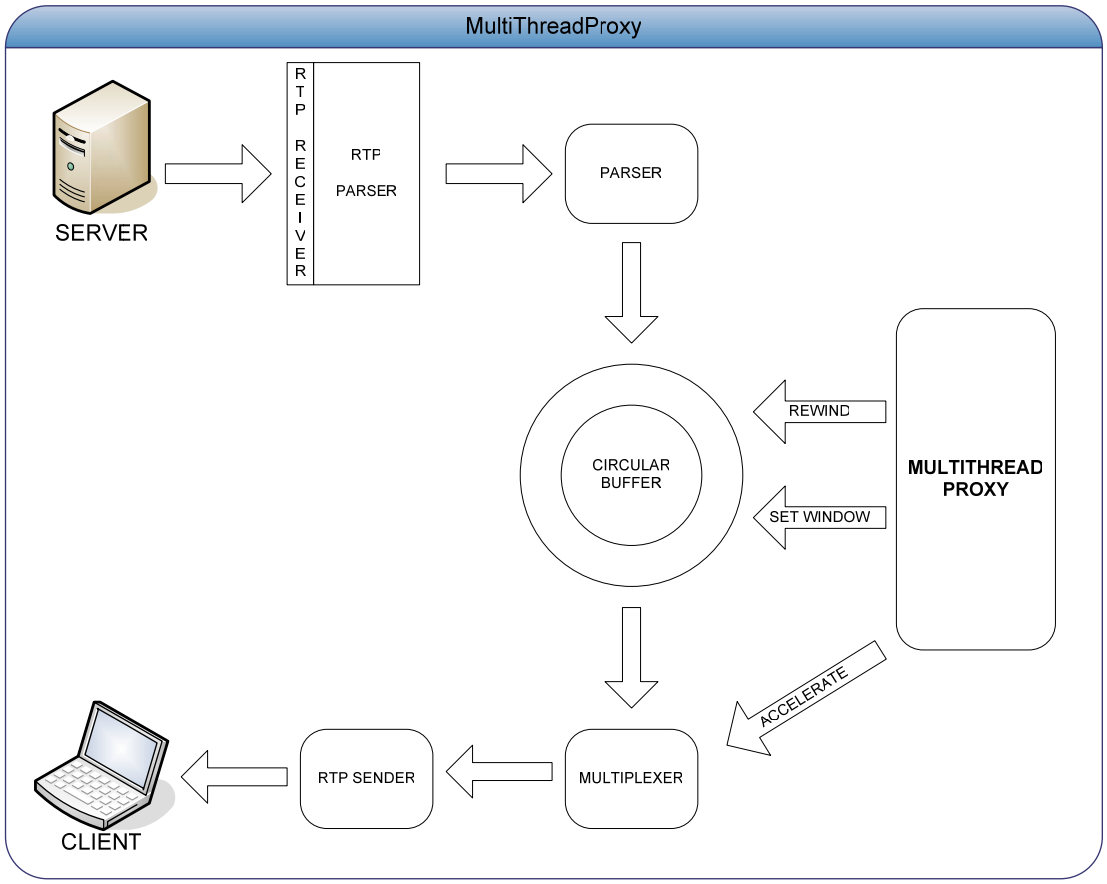

Figura 3.5: MultiThreadProxy

3.2.2 MultiThreadProxy

Il MultiThreadProxy è un componente del proxy specifico e il suo ruolo è di gestire il flusso di dati multimediali. Quindi deve contenere una parte che si incarica comunicare con il server per ricevere i dati (RTPReceiver), una parte che si incarica di comunicare con il client per inviare i dati (RTPSender) e un buffer dove salvare i dati ricevuti dal server prima di inoltrarli al client (CircularBuffer). Poi esistono un componente per estrarre i vari dati dal flusso ricevuto e inserirli nel buffer (Parser) e un componente che estrae i dati dal buffer e forma il flusso da inviare al client.

Il buffer è realizzato come buffer circolare in modo da poterlo "riavvolgere" per servire la procedura di rewind. Il comando *SetWindow* che vediamo nella figura serve a modificare la dimensione del buffer.

3.3 Client

Abbiamo sottolineato l'importanza di avere un proxy che si adatti ad ogni tipo di client in modo da poter gestire al meglio le sue risorse e le richieste del Client stesso. Il Client non vuole essere così adattivo, ma certamente dovrà favorire il compito del Proxy e quindi comunicargli quante più informazioni utili possibile sulle proprie esigenze. Il client è composto da una parte applicativa e un middleware che è un componente indipendente dal problema applicativo che fornisce informazioni sul contesto nel quale si opera (stato della connessione, probabilità dell'handoff orizzontale, qualità del segnale, AP presenti, ecc). Questo middleware potrà essere riutilizzato in altre applicazioni dipendenti dalle informazioni sul contesto. Sappiamo che il client deve essere consapevole del proprio contesto di esecuzione, e quindi il supporto middleware presente, denominato Stub, dovrà essere in grado interrogare il sistema sulle risorse disponibili oppure ricavare informazioni riguardanti la scheda wireless utilizzata. Ovviamente, in base a queste informazioni, l'infrastruttura di supporto distribuita

dovrà essere in grado di adattare il funzionamento del client e del proxy specifico a lui associato. Il middleware è Lato client, inoltre è presente il componente in grado di riprodurre i dati multimediali ricevuti dal proxy. Questo componente applicativo è denominato Player. Esistono due modalità di riproduzione dei dati multimediali: prima ricevere tutti i dati e poi riprodurli, oppure riprodurre i dati durante la ricezione del flusso multimediale. Il client realizzato adotta questa seconda modalità. Avendo a che fare con una riproduzione di questo tipo si capisce la necessità di avere un buffer interno al client. Durante gli handoff orizzontali c'e una perdita di comunicazione con il proxy e quindi il client durante questo periodo non riceve nuovi dati. Avendo dentro il proprio buffer dei dati ricevuti, ma non ancora riprodotti, il client può evitare di interrompere la riproduzione dei dati utilizzando quelli contenuti nel proprio buffer nella speranza che la comunicazione con il proxy riprenda prima dello svuotamento del buffer stesso. Questa situazione si deve verificare in maniera completamente trasparente per l'utente. Per gestire e coordinare questi due componenti deve esserci un ulteriore componente denominato Controller.

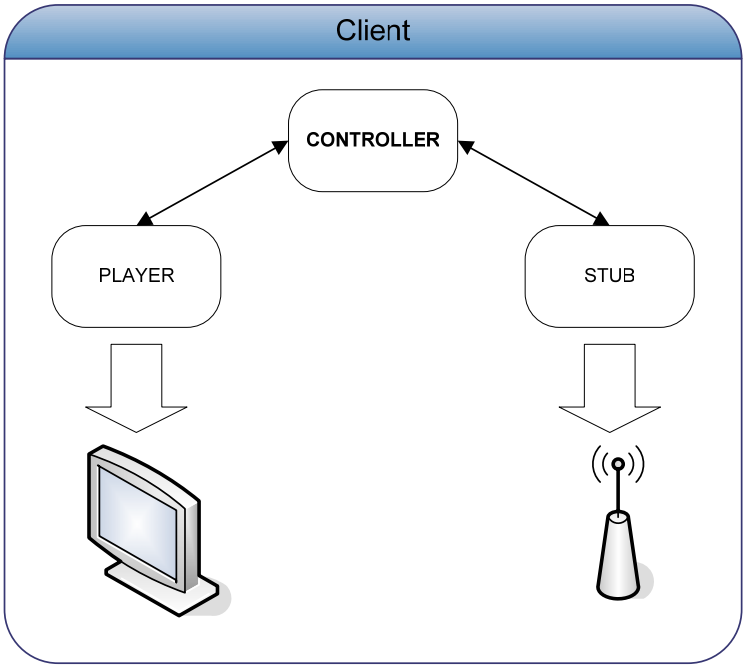

Figura 2.6: Client

3.3.1 Player

Il player è il componente del client che si occupa della ricezione del flusso di dati multimediali dal proxy e della loro riproduzione all'utente. Per effettuare questa funzione si avvale dei seguenti componenti:

- Listener (RPCClient)
- Receiver (ReceiveStreamReader)
- Buffer (QueableCircularBuffer)
- Renderer (BufferRenderer)

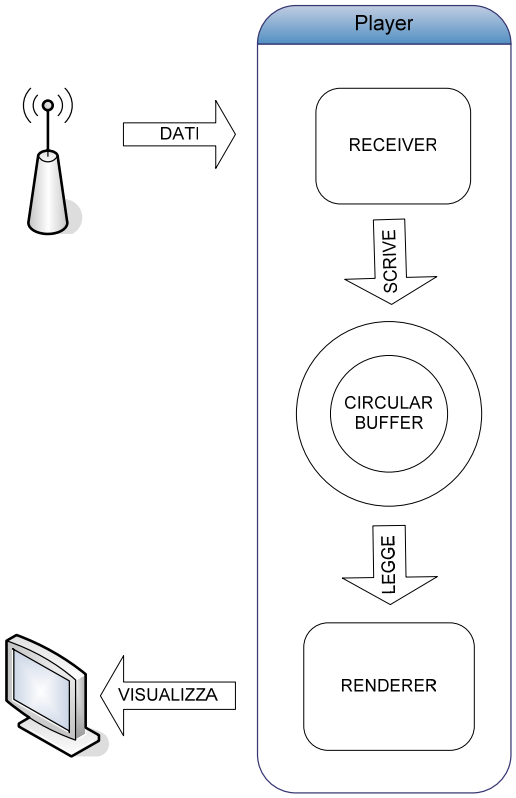

Figura 2.7: Player

Il Listener è il componente che deve gestire la comunicazione del flusso di dati multimediali. Si pone in ascolto aspettando l'arrivo dei dati, poi ricevendo l'evento del loro arrivo invoca il Receiver che provedera a riceverli. Il componente Receiver, che viene creato dal Listener all'arrivo del primo dato dal proxy, ha il compito di estrarre i dati dal flusso in arrivo e di aggiungerli nel buffer. Il buffer è

un contenitore di dati ricevuti del client. È importante che il buffer marchi ogni frame ricevuto con un timestamp mediante il quale sia possibile stabilire una successione tra frames, necessaria per chiedere il rewind sul lato proxy. Inoltre deve fornire le informazioni sul proprio stato di riempimento, necessarie per chiedere di modificare la velocità di invio dei dati da parte del proxy. Il Renderer è invece il componente incaricato di consumare i dati multimediali estraendoli dal buffer e riproducendoli all'utente.

3.3.2 Stub

Lo stub è il componente del client incaricato di recuperare tutte le informazioni sulla scheda wireless utilizzata dal client per effettuare la comunicazione con il proxy. Viene utilizzato un insieme di comandi di basso livello per interagire con la scheda wireless e recuperare tutte le informazioni necessarie. Alcune di queste informazioni sono: la probabilità dell'handoff orizzontale, la potenza del segnale, la qualità del segnale, ecc. Inoltre può effettuare una previsione dell'AP al quale si dovrebbe passare avendo un'handoff orizzontale. Questo consente al controller del client di poter migliorare la qualità di servizio gestendo opportunamente gli handoff verticali. Questo componente è implementato dal ClientStub.

3.3.3 Controller

Il controller è il componente principale del client ed è implementato dal MainClient. Ha il compito di avviare il player e lo stub e di gestire la comunicazione con il proxy inviandogli le proprie richieste. Inoltre ha il compito di controllare il contesto nel quale opera e inviare le richieste opportune al proxy che provvederà a gestirle. Per gestire la richiesta di *Acknowledgement*, la quale serve a segnalare la propria presenza al proxy, il controller crea l'istanza AckController che ha il compito di inviare l'ACK al proxy periodicamente con una certa frequenza decisa dal controller.

3.4 Simulatore

Il Simulatore ha lo scopo di permettere lo sviluppo delle applicazioni senza avere la necessità di provarle in un ambiente wireless reale e su macchine con dispositivi wireless. Il suo compito principale sarò quindi di riprodurre il più fedelmente possibile la realtà che circonda un dispositivo wireless. È stato introdotto per poter effettuare tutti gli esperimenti e può essere diviso in due parti: la prima si occupa della simulazione dei segnali che sarebbero captati, nel mondo reale, dalla scheda wireless e che proverrebbero dagli AP, la seconda invece recupera tutti i dati offerti dalla prima parte ed emula il comportamento di una scheda wireless.

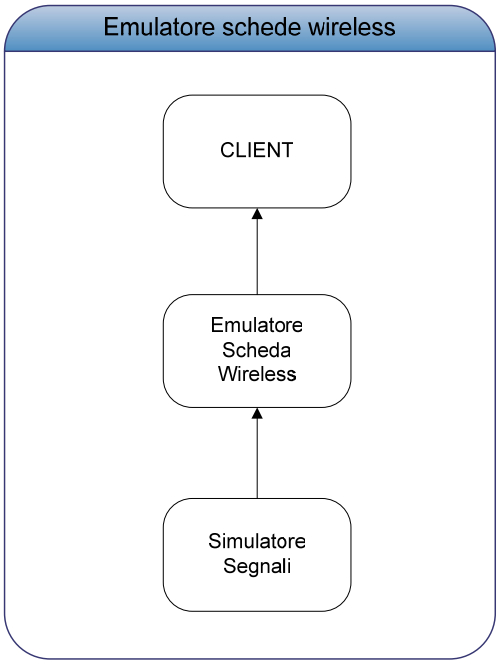

Figura 2.8: Emulatore schede wireless

Il simulatore inizialmente recupera tutti i dati relativi agli AP presenti e successivamente calcola la potenza del segnale di ciascun AP. I dati relativi ad ogni AP sono incapsulati in una struttura chiamata LocationOss che mantiene le informazioni quali il MAC e la potenza del segnale di un AP. Sono poi valutate le potenze dei segnali ricevuti da ogni client e la politica della sua scheda (Hard Proactive o Soft Proactive). Il Simulatore indica a quale AP è logicamente

collegato e poi, chi raccoglie i dati dal Simulatore, valuterà in maniera indipendente la probabilità e la gestione dell'handoff orizzontale.

3.4.1 Emulatore delle schede

Questo componente ha il compito di fornire informazioni al client e di simulare il comportamento di una scheda wireless. Deve offrire un'interfaccia che non sia diversa da quella di un contesto reale e presentare gli stessi comportamenti a livello di connessione di una scheda wireless reale. In particolare l'emulatore simula l'handoff e quindi la perdita di connessione che ne deriva. La decisione di effettuare l'handoff è presa in base ai dati forniti dal modulo sottostante ossia dal simulatore dei segnali. Per offrire i servizi principali delle schede wireless, questo modulo avrà dei metodi che permettono di ottenere informazioni quali gli AP visibili, le potenze di questi e il MAC dell'AP attuale.

3.4.2 Simulatore di segnali

Questo modulo prevede un insieme di meccanismi che simulano la visione del mondo da parte della scheda wireless e fornisce informazioni che dipendono dalla posizione della scheda all'interno del contesto. Il contesto non deve essere fisso ma variare per poter rappresentare un insieme di situazioni e sperimentare al meglio la validità delle soluzioni adottate.

3.4.3 SimulatorStub

È il componente che è a conoscenza della caratteristiche del simulatore e delle richieste dello stub del client e si interpone fra di loro. Nella figura successiva vediamo l'interazione fra questi componenti.

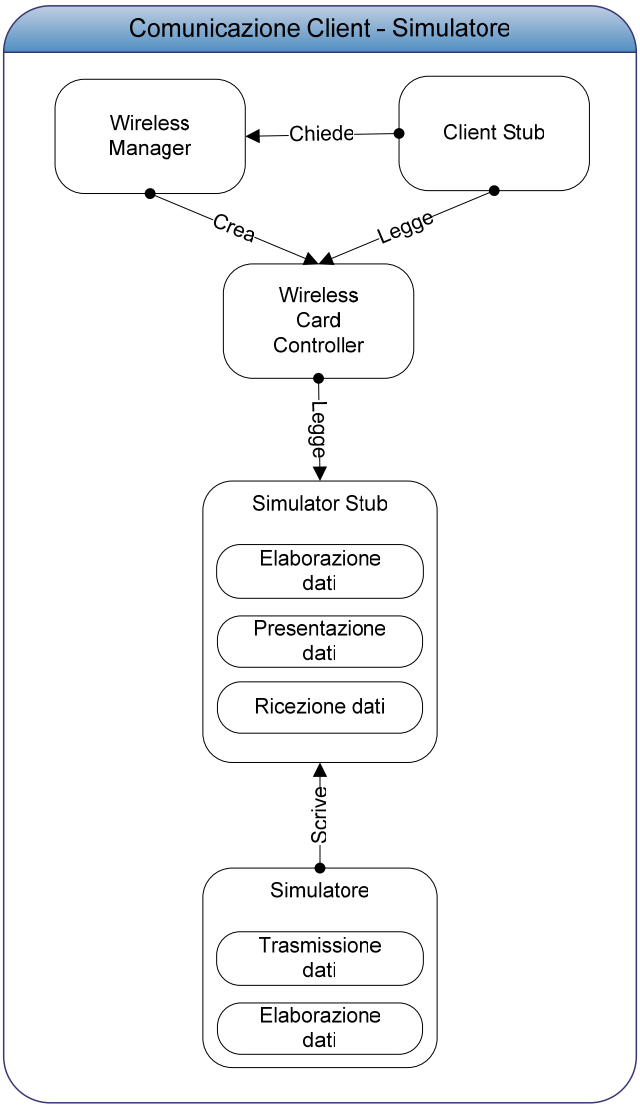

Figura 3.9: Comunicazione Client - Simulatore

Il SimulatorStub riceve informazioni dal Simulatore ed interrogazioni da parte di altre entità. Questa classe è costituita da due server, il primo si pone in ascolto e riceve i dati dal Simulatore, il secondo accetta richieste da parte di un particolare tipo di WirelessCardController.

Il WirelessCardController è introdotto per fornire tutte le informazioni standard sulla scheda al ClientStub e svincolarlo così dalla conoscenza del tipo di scheda. Fornisce informazioni sulla presenza o meno di una connessione ad internet, sul tipo di scheda, restituisce l'AP a cui si è connessi o la lista degli AP visibili con i relativi RSSI. In realtà al ClientStub verrà fornito un WirelessCardController

mediante il WirelessManager che è a conoscenza della reale situazione del sistema (se è in atto una simulazione o è una vera e propria connessione) il quale restituisce o una istanza di LogRunner o di SimWirelessController che implementano il WirelessCardController che di fatto è astratto.

Il SimWirelessController implementa il WirelessCardController. Per prima cosa ricava le informazioni sul tipo di scheda e l'AP a cui è connesso nella simulazione, dopodichè si dichiara attivo; questa seconda fase corrisponde logicamente all'individuazione di una connessione Wireless. Il Wireless Card Controller ciclicamente chiede al Simulator Stub le informazioni sugli AP visibili e sull'AP con il quale è attualmente stabilita la connessione. Queste informazioni vengono memorizzate all'interno del Controller per poter essere lette dal ClientStub. Le letture del ClientStub sul Controller sono asincrone rispetto alle letture che il Controller opera sul SimulatorStub. Cosi facendo il Controller svincola il ClientStub dalla durata della propria interazione con il SimulatorStub.

Il Simulatore, oltre alla modalità di funzionamento spiegata precedentemente, offre la possibilità di funzionare come emulatore. Questa modalità viene implementata dalla classe LogRunner che è restituita al ClientStub dal WirelessManager sostituendo il SimWirelessController. Questa classe rappresenta un Thread che è lanciato e successivamente chiede i diritti di lettura sul file di trace. È anche creato un nuovo Thread chiamato Wall che blocca le socket e le risveglia emulando, in pratica, l'handoff. Il LogRunner effettua il parsing delle righe dei file di trace cambiando i valori degli AP attuali e di quelli visibili. Ad ogni ciclo è comunicata la lettura della riga successiva per poter calcolare il tempo di inattività prima di poter operare il successivo cambiamento. Quando è incontrata una riga che segnala l'handoff, il LogRunner configura il Wall per la durata prevista ed infine lo risveglia, emulando cosi un'interruzione momentanea della connessione.

3.5 Classi del gruppo Mobilab

Nel progetto è stato utilizzato anche un package sviluppato dal gruppo Mobilab di Napoli il cui compito è quello di recuperare e fornire alcune informazioni di contesto e anche alcune API di controllo e gestione per quello che riguarda la tecnologia Bluetooth [CLM/05].

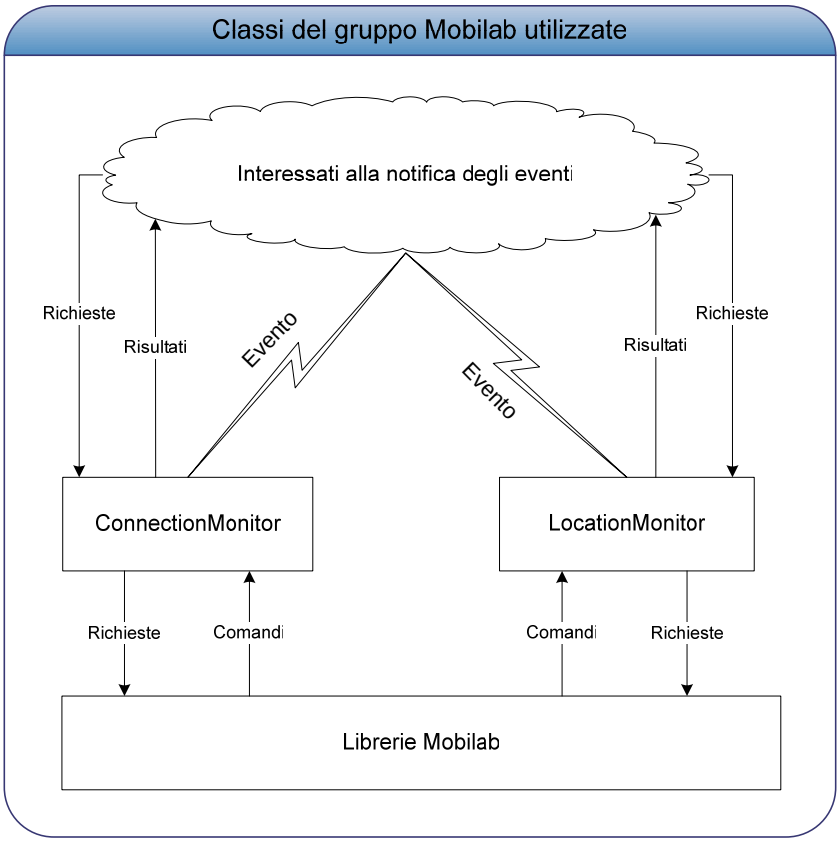

Figura 2.9: Classi del gruppo Mobilab utilizzate

Le entità ConnectionMonitor e LocationMonitor possono essere interrogate a polling oppure fornire informazioni mediante eventi che saranno intercettati e gestiti in modo opportuno. In particolare queste due entità forniscono informazioni relative alla connessione (potenza del segnale corrente, numero di connessioni create fino a questo momento, disponibilità canale, probabilità di handoff orizzontale) ed alla locazione (nome e grado della locazione).

È stato anche utilizzato il demone “conmand”, entità sempre attiva sul lato client che, una volta attivata (mediante l’utilizzo dell’entità CLM), effettua la scansione utilizzando la tecnologia Bluetooth. Questo demone si occupa del rilevamento del demone Napd sul lato server, ed in caso questo esista instaura una comunicazione con quel server.

Napd viene utilizzato sul lato proxy e rende possibile realizzare un AP Bluetooth intercettando connessione in arrivo sulla PAN e effettuare un bridge su un’unica interfaccia a livello datalink (livello delle interfaccie delle schede Bluetooth). Il demone Napd utilizza Bnep (Bluetooth network encapsulation protocol), all’atto della creazione di una connessione PAN il demone pan (distribuito con lo stack BlueZ, stack ufficiale bluetooth per Linux) crea un’interfaccia ad-hoc che prende il nome di bnepx (dove x indica la x-esima connessione). Per consentire la comunicazione mediante una sola interfaccia virtuale Napd fonde l’interfaccia virtuale bnepx e pan0. In questo modo è possibile associare a schede Bluetooth anche indirizzi IP.

3.6 Mobility Monitor

Il componente Mobility Monitor è stato sviluppato al DEIS [IMGW/05], e serve a fornire informazioni sul contesto in modo omogeneo indipendentemente dalla particolare tecnologia utilizzata. È stata pensata un’architettura in cui si ha un’entità, che è chiamata Manager, il cui compito è quello di recuperare queste informazioni e fornirle agli interessati. Questa entità può essere sia attiva e sia passiva a seconda delle esigenze e delle risorse a disposizione. L’effettivo recupero delle informazioni di interesse è effettuato da altre due entità distinte, che sono chiamate BluetoothExecutor e WiFiExecutor entrambe entità passive, il cui compito è quello di recuperare alcune informazioni per il Bluetooth e per il WiFi e di fornirle al Manager. Inoltre queste due entità si occuperanno anche dell’attivazione e disattivazione del dispositivo Bluetooth e WiFi e restituiranno al Manager due oggetti, rispettivamente BtInfo per il Bluetooth e WifiInfo per il

WiFi, che incapsulano le informazioni di interesse. È stata prevista la possibilità, da parte del BluetoothExecutor, di fornire altre informazioni mediante un ulteriore oggetto chiamato BtDevice.

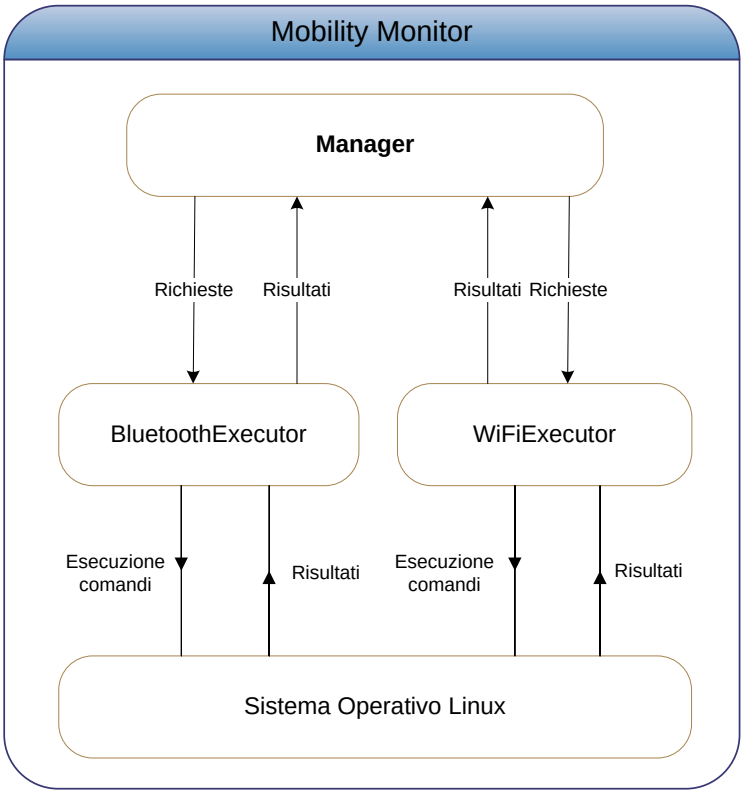

Figura 2.10: Mobility Monitor

La comunicazione tra Manager e lo strato superiore avviene mediante la notifica di eventi asincroni. Sono stati previsti alcuni tipi di eventi con lo scopo di fornire informazioni addizionali sul cambiamento dell'indirizzo IP del client, sul cambiamento della probabilità di handoff orizzontale in Bluetooth e WiFi e per sul cambiamento della locazione in Bluetooth. Per ricevere questi eventi le entità interessate alla notifica devono implementare i relativi listeners e registrarsi presso il Manager.

Capitolo 4
Analisi del Progetto

Questo capitolo ha il compito di dare una descrizione architetturale del progetto senza entrare nel dettaglio dell'implementazione fornendo però l'architettura e le funzionalità svolte dalle varie entità.

4.1 Architettura Proxy-based

Per risolvere il problema dell'handoff si è scelto di interporre una terza entità tra il server ed il client, e tale entità prende il nome di proxy. Una ragione che ha portato a questa scelta è la possibilità di avere client eterogenei con i quali c'è bisogno di una interazione dinamica. Un'altra ragione è la caratteristica del client di poter cambiare modalità di funzionamento in modo dinamico, ad esempio può richiedere e cambiare la tecnologia con cui comunicare, coordinare il flusso dei dati multimediali, ecc.

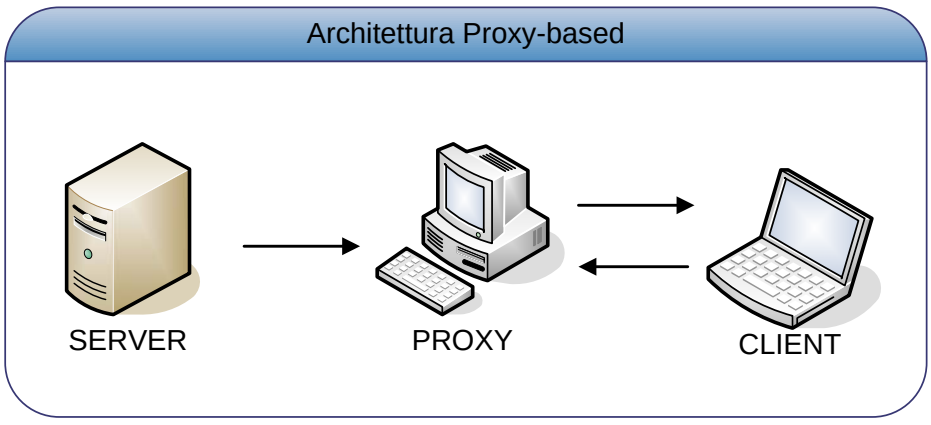

Figura 4.1: Architettura Proxy-based

Si è ritenuto di dover decentralizzare la gestione del client in modo da poter adattare dinamicamente il servizio alle variazioni del contesto. In questo modo si da' la possibilità al server di essere svincolato dalle caratteristiche del client, e

quindi viene diminuita la sua complessità e ne viene aumentata l'efficienza. Il server delega questi compiti ad uno specifico componente: il proxy.
Avendo a che fare con dei servizi multimediali necessitiamo di una coordinazione forte tra il trasmittente e il ricevente dei dati. In questa architettura di ciò si occupa solamente il proxy isolando il server da questa problematica.

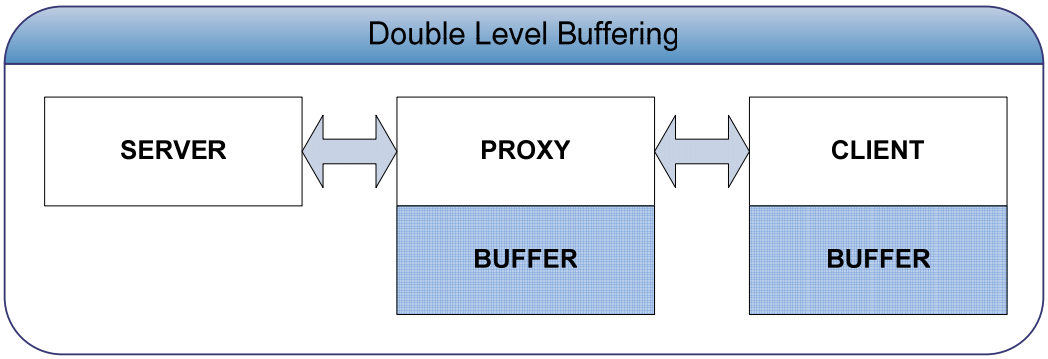

Figura 4.2: Doppio livello di buffering

In particolare, per migliorare e facilitare la coordinazione tra il proxy e il client durante il trasferimento di dati multimediali è stata adottata un'architettura a doppio livello di buffering [PB/04]. Sia il proxy che il client sono dotati di un proprio buffer. Al client serve per migliorare la rappresentazione dei dati, cioè se si verifica una temporanea perdita o mancanza di dati nuovi nel frattempo il client può proseguire la rappresentazione consumando i dati presenti nel buffer come se non ci fosse stata nessuna perdita. Invece al proxy il buffer serve per poter svincolare il server dei problemi di coordinazione del flusso dei dati dovuto a problemi di instabilità e dicontinuità di comunicazione con il client. Ad esempio, il client può richiedere al proxy di rimandare dei dati già precedentemente mandati, ma evidentemente persi per una discontinuità della connessione, e anche di cambiare frequenza di invio dei dati.
Questa tesi si concentra sull'interazione tra proxy e client non occupandosi invece della comunicazione fra server e proxy.

4.2 Gestione dell'handoff verticale

Utilizzando tecnologie wireless abbiamo la possibilità di svincolare il terminale da una postazione fissa. Potendo sfruttare questa mobilità del terminale si può verificare la perdita di connessione con un AP uscendo dalla sua area di copertura. L'evento del passaggio dalla connessione su un AP a un altro viene denominato handoff. Quando avviene un handoff orizzontale normalmente si ha una perdita di dati sul lato client. Invece nell'handoff verticale questo si può evitare usando una politica di gestione dell'handoff di tipo Proactive, cioè quando si avvia la procedura di handoff prima che il segnale con AP attuale venga perso, questo viene fatto appena si trova un'AP con un segnale migliore di quello attuale.

Nel nostro caso la gestione dell'handoff verticale è proprio di questo tipo. Questa scelta comporta sicuramente un maggiore dispendio di energia, ma ci consente di migliorare la continuità e qualità del servizio offerto.

Il client può controllare il contesto attuale nel quale opera solo quando esiste la necessità di effettuare un'handoff verticale lo esegue e si coordina con il proxy al riguardo. Così facendo è possibile evitare di perdere dati sul lato client. Questo coordinamento durante l'esecuzione dell'handoff verticale fra il client e il proxy viene definito come *rebind.*

4.2.1 Procedura di Rebind

Durante l'erogazione del servizio multimediale ci possono essere dei disturbi di comunicazione, discontinuità di servizio, ecc. Per ovviare a questi problemi il client deve poter valutare il contesto attuale nel quale opera ed eventualmente, nel caso la qualità della comunicazione non lo soddisfi, controllare le possibili alternative, nel nostro caso l'altra tecnologia wireless disponibile per la comunicazione e, se si presume un miglioramento della comunicazione eseguire la procedura di rebind. Durante i controlli sull'altra tecnologia il client può mantenere attive entrambe le tecnologie e ricevere i dati usando entrambe. Nel

caso si decida di effettuare l'handoff verticale basterà disattivare la tecnologia precedentemente usata e lasciare attiva l'altra oppure nel caso non si voglia effettuare l'handoff si disattiva l'altra tecnologia. Avendo a che fare con servizi multimediali che necessitano di una continuità di servizio capiamo l'importanza di questa procedura che ci consente di non avere perdite di dati durante l'esecuzione dell'handoff verticale.

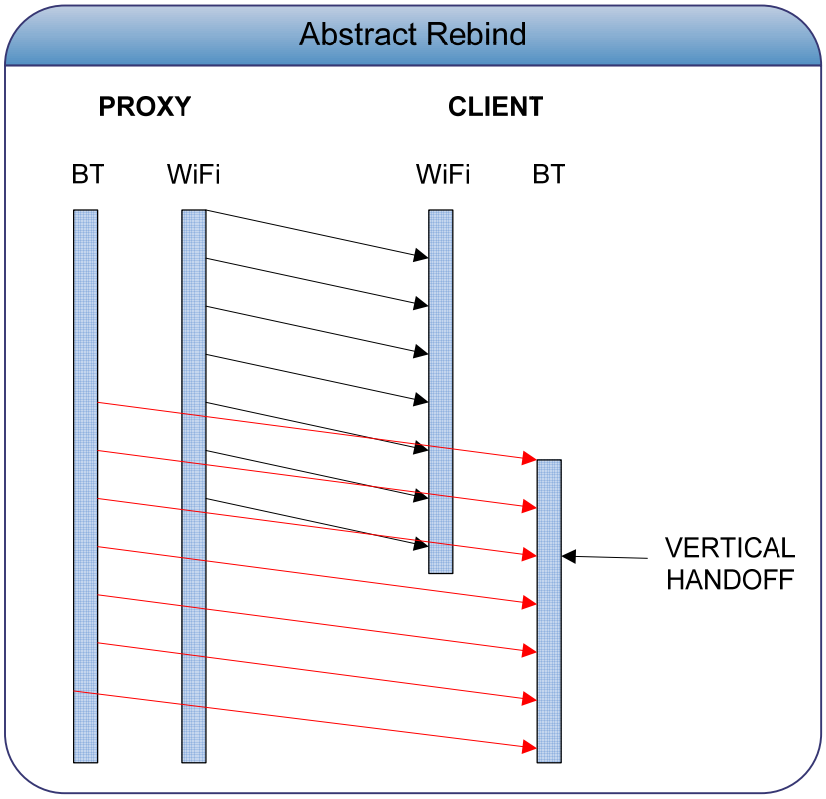

Figura 4.3: Rebind astratto

La figura mostra solo la trasmissione dei dati, mentre per il controllo del flusso dati viene utilizzato un diverso canale. Se il client vuole effettuare un'handoff verticale deve notificare questa informazione al proxy che si dovrà adattare eseguendo anch'esso l'handoff verticale.

La procedura di rebind serve per coordinare il client e il proxy durante l'esecuzione dell'handoff verticale. La funzionalità di rebinding, oltre a contenere il coordinamento delle tecnologie usate per il trasferimento, contiene anche la

logica di coordinamento del flusso dati multimediale e gestisce il doppio livello di buffering.

4.3 Proxy

Per Proxy si intende il modulo che deve porsi tra il server e il client in modo da offrire al server la possibilità di svincolarsi dalla conoscenza dello specifico client e al client la possibilità di adattarsi dinamicamente al suo funzionamento con l'intento di migliorarne le prestazioni. Diventa importante per il proxy offrire la possibilità di trasmissione su diverse tecnologie wireless per dare più flessibilità al client e migliorare la qualità del servizio offerto. Per poter offrire questa possibilità il proxy ovviamente deve essere in grado di comunicare col client utilizzando diverse infrastrutture (AP di tecnologie wireless diverse).

4.3.1 Adattamento dinamico al Client

Il proxy deve avere una struttura adattabile alle esigenze del client e non una struttura fissa per tutti. Questo perché le macchine che ospitano l'applicazione client possono essere estremamente diverse tra loro. Proprio per questa eterogeneità dei client si è deciso di avere un proxy specifico per ogni client che possa adattarsi al meglio alle sue esigenze. Questo ci consente di spezzare l'architettura del proxy su due livelli: molti proxy specifici, uno per ogni client, e un proxy generale che gestisce tali proxy specifici. Il proxy generale riceve la prima richiesta del client e poi crea un proxy specifico delegandogli la gestione delle richieste di quel client. In questo modo al cambiamento delle caratteristiche o del contesto nel quale operano i diversi client sarà necessario adattare solo il loro proxy specifico.

Nel caso dell'handoff verticale, la procedura di rebind inizia sul client con il cambiamento dell'interfaccia di rete usata che corrisponde al cambiamento della tecnologia, quindi il client informa del rebind il proxy specifico. Il proxy specifico

riceve la richiesta dal client ed effettua anch'esso il proprio rebind (cambiamento dell'AP usato che corrisponde al cambiamento della tecnologia).

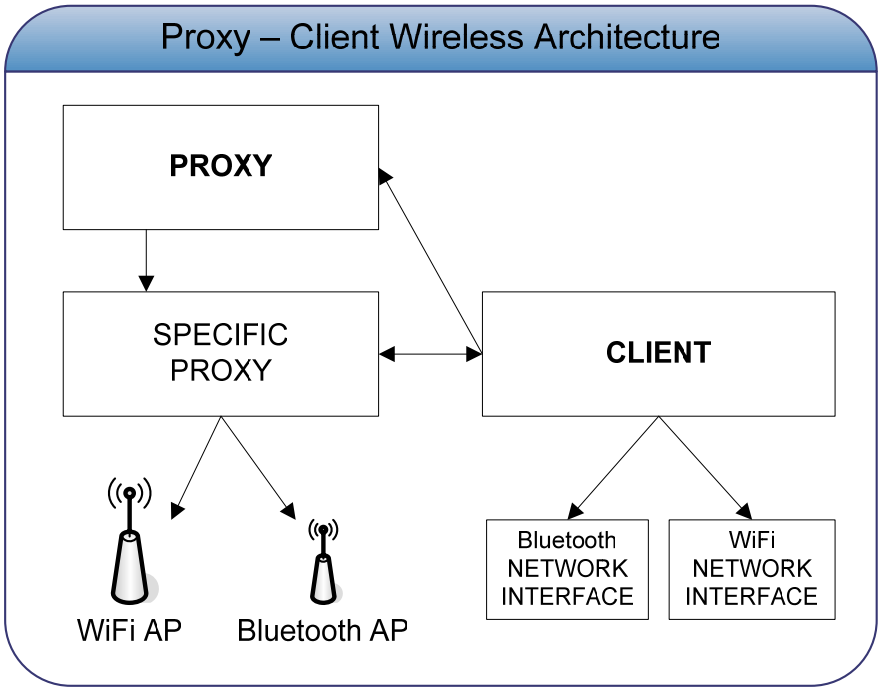

Figura 4.4: Architettura wireless proxy-client

4.4 Client

Abbiamo sottolineato come nei servizi multimediali ci sia necessita di avere continuità e qualità di connessione. Per rispondere a questi vincoli il client deve essere in grado di recuperare informazioni di contesto nel quale opera e di conseguenza poter decidere di effettuare delle modifiche sul proprio funzionamento. Una di queste situazioni è proprio quella nella quale il client decide di effettuare effettua l'handoff verticale, come spiegato meglio nella sezione seguente, e poi si coordina con il proxy che si adegua in modo opportuno.

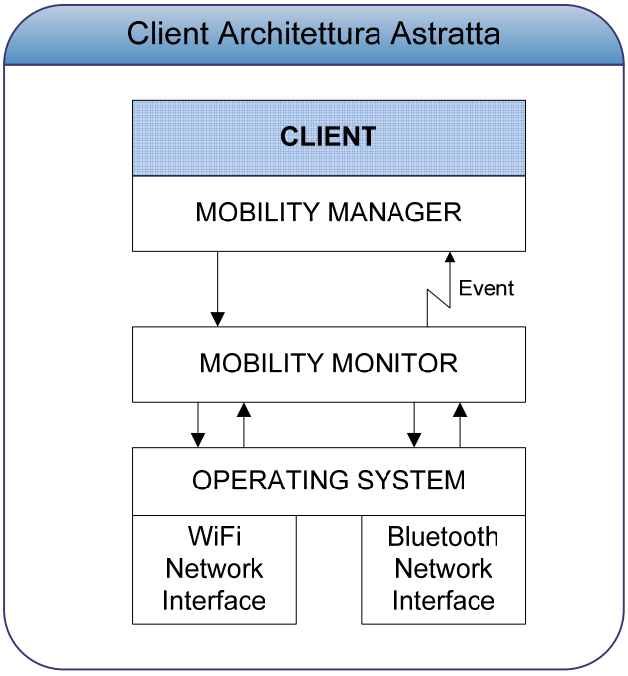

Figura 4.5: Architettura astratta del client

4.4.1 Gestione del contesto

Per poter sapere quando sia necessario effettuare un'handoff verticale il client deve essere in grado di recuperare tutte le informazioni necessarie al riguardo. Queste informazioni possono essere di varia natura, ma per l'obbiettivo di questa tesi le importanti sono quelle riguardanti le varie interfacce di rete e tecnologie disponibili. Queste possono essere indirizzi di rete o gli indirizzi di livello datalink (MAC), qualità e potenza del segnale, larghezza di banda, probabilità di un handoff orizzontale, rapporto segnale-rumore, ecc. Queste informazioni di contesto del client vengono fornite dal blocco Mobility Monitor in due modalità: la prima è essere direttamente interrogato sulle informazioni che si desiderano e la seconda e di registrarsi presso lui per ricevere eventi relativi alla modifica del contesto riguardante un tipo di informazioni. Il blocco Mobility Manager è quello che usufruisce delle informazioni fornite dal Mobility Monitor e effettua la gestione del contesto adattando il comportamento del client alle specifiche modifiche di contesto.

4.4.2 Politiche di gestione dell'handoff verticale

Dovendo decidere di modificare il proprio comportamento in dipendenza dello specifico contesto diventa necessario introdurre delle politiche di gestione. Una di queste politiche è anche la politica di gestione dell'handoff verticale. Al riguardo esistono due possibili politiche:

- *Band Amplitude*: la politica di gestione che privilegia la tecnologia WiFi, essendo essa dotata di una maggiore banda di trasmissione, rispetto a quella Bluetooth;
- *Battery Consumption*: la politica di gestione che privilegia la tecnologia Bluetooth, essendo essa dotata di un minore consumo di energia, rispetto a quella WiFi.

Ovviamente si nota subito che queste sono le politiche di gestione più semplici che possono però essere combinate e complicate creando politiche sempre più specifiche.

Capitolo 5
Progetto ed implementazione

In questo capitolo si descriveranno in dettaglio il progetto e l'implementazione mettendo in evidenza i dettagli e le scelte che sono stati effettuati.
Come introdotto brevemente nel capitolo uno, l'obbiettivo dell'infrastruttura è quello di recuperare le informazioni dei dispositivi wireless necessari e di operare una gestione dell'handoff verticale. Nella architettura preesistente era possibile solo la gestione dell'handoff orizzontale e quindi è stato necessario ampliare questa architettura dandole la possibilità di gestire anche gli handoff verticali.
La descrizione e divisa in quattro parti: la prima contiene il progetto, la seconda contiene i dettagli implementativi relativi al lato client, la terza contiene i dettagli relativi al lato proxy, e la quarta contiene alcune applicazioni di test.

5.1 Progetto

Dato che il client della struttura preesistente aveva la possibilità di gestire solo handoff orizzontali ovviamente si è reso necessario modificarlo in modo opportuno, anche aggiungendo alcune classi.
Nella figura seguente sono riportate le nuove entità e le entità modificate per aggiungere la possibilità di effettuare handoff verticali.
Abbiamo visto che nella struttura preesistente il client si divideva in 3 blocchi principali: il player ,lo stub e il controller. Principalmente nelle entità modificate le modifiche riguardano l'aggiunta dell'adattamento all'esecuzione dell'handoff verticale. Nella parte del player sono state fatte modifiche all'entità RPCClient, che implementa un listener della ricezione dei dati multimediali dal proxy, e all'entità QueableCircularBuffer, che implementa il buffer contenente i dati ricevuti dal proxy. Nella parte del controller è stata modificata l'entità del

MainClient e sono state aggiunte le entità Manager e ManagerEventListener. Il ManagerEventListener è il listener degli eventi di cambiamento del contesto scatenati dal Manager e serve a controllare se il contesto attuale necessità dell'esecuzione dell'handoff verticale ed eventualmente invocare l'esecuzione della procedura di rebind da parte del MainClient.

L'entità MainClient, che nell'architettura astratta abbiamo definito Mobility Manager, opera tutte le azioni di controllo sul client ed effettua le operazioni di coordinamento con il proxy. Per recuperare informazioni di contesto si appoggia all'entità Manager e contiene tutta la procedura di rebind del client che viene invocata dal ManagerEventListener. E stata inoltre modificata la gestione del ridimensionamento del buffer del client aggiungendo la possibilità di verificare lo stato della connessione su più tecnologie.

Viene ulteriormente modificata l'entità AckController, che ha il compito di spedire con una certa frequenza al proxy il messaggio di acknowledge (dice al proxy che il client è ancora presente), aggiungendole la possibilità di cambio della tecnologia usata per spedire i messaggi di ACK al proxy.

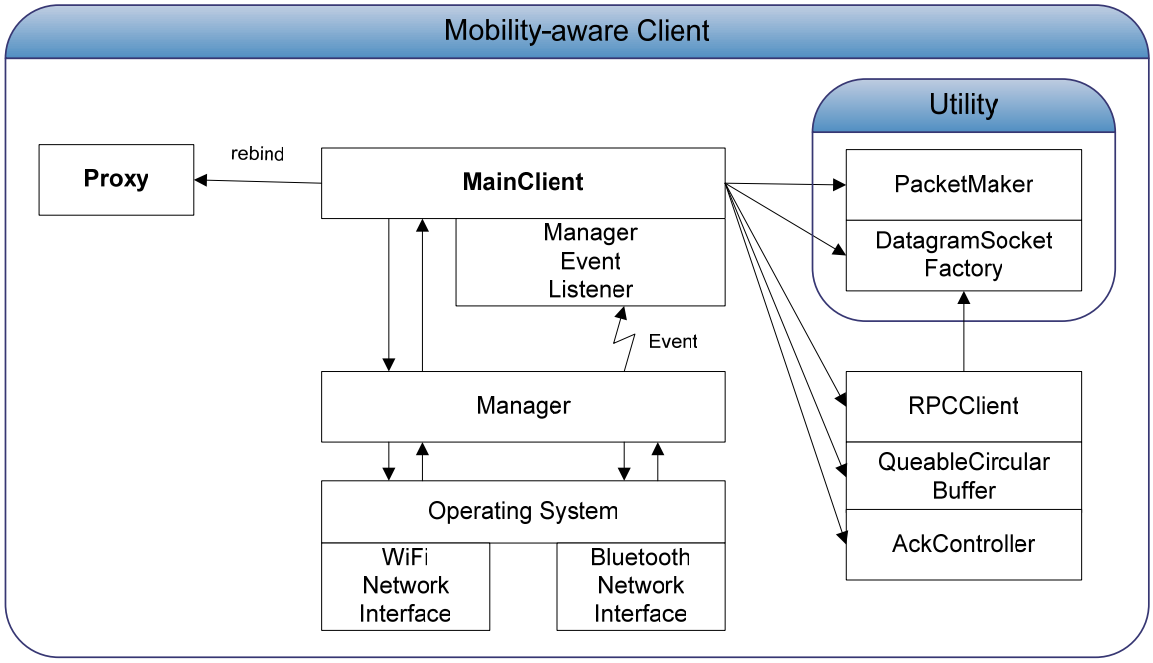

Figura 5.1: Mobility-aware client

Un blocco particolarmente importante per poter effettuare una gestione dell'handoff verticale è il Manager, che nell'architettura astratta abbiamo denominato Mobility Monitor, ed ha lo scopo di fornire le informazioni di contesto al MainClient (essendogli state richieste) o al ManagerEventListener (tramite eventi ogni qualvolta capiti un cambiamento di interesse).
In particolare, la scelta della politica di gestione dell'handoff verticale inizialmente viene fatta durante l'inizializzazione del MainClient e successivamente viene modificata dal ManagerEventListener adeguandosi dinamicamente al cambiamento di contesto.

Dato che il proxy della struttura preesistente aveva la possibilità di gestire la comunicazione solo con una tecnologia wireless ovviamente si è reso necessario modificarlo in modo opportuno, anche aggiungendo alcune classi.
Nella figura seguente sono riportate le nuove entità e le entità modificate per aggiungere la possibilità di comunicazione su più tecnologie wireless.

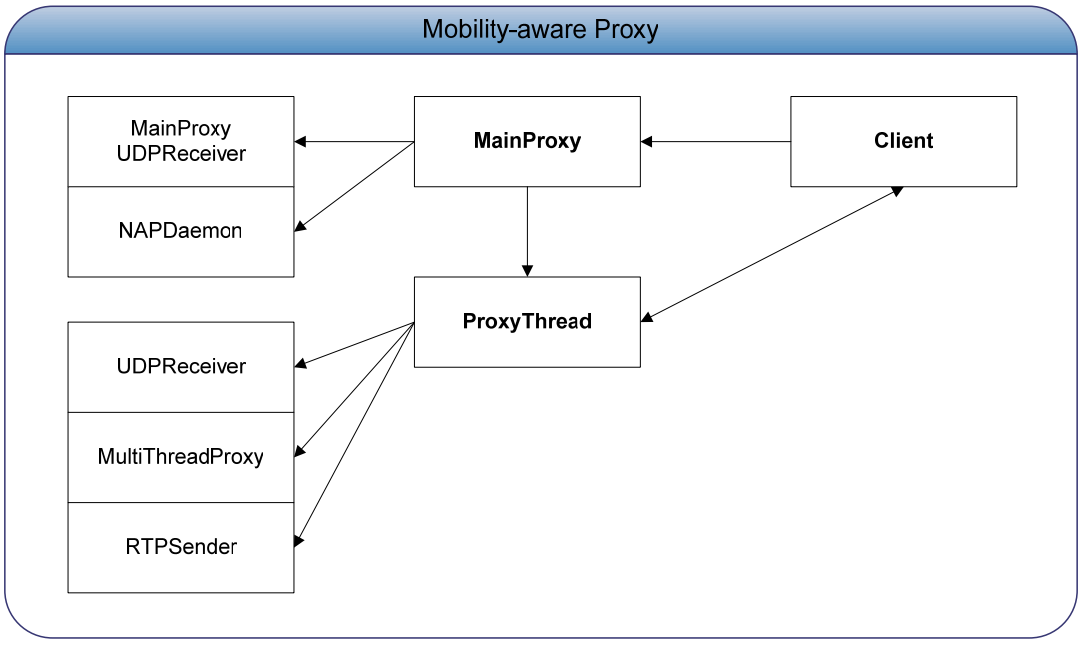

Figura 5.2: Mobility-aware proxy

Principalmente, nelle entità modificate, le modifiche riguardano l'aggiunta dell'adattamento alla esecuzione su più tecnologie. Il MainProxy è il componente che implementa il proxy generico, accetta la prima richiesta del client, poi associa un proxy specifico per lui ed infine delega la comunicazione con il client al proxy specifico stesso. Dovendo funzionare su più tecnologie il MainProxy deve essere in ascolto delle richieste del client solo su una tecnologia e quindi delega questo compito alla nuova entità creata denominata MainProxyUDPReceiver che svolge questo compito per lui passandogli le richieste arrivate. Il componente ProxyThread implementa il proxy specifico che viene associato al client dal MainProxy. Ha il compito di comunicare con il client e di adattarsi al suo funzionamento. L'UDPReceiver invece implementa l'ascoltatore di richieste del client durante l'esecuzione del servizio. Quando riceve una richiesta semplicemente la passa al client. Il componente MultiThreadProxy implementa la parte del proxy che gestisce il flusso di dati multimediali dal server al proxy, ricevendo dati dal server, effettuando modifiche su di essi e inviandoli al client. Un componente che fa parte del MultiThreadProxy è l'RTPSender. Il suo compito è inviare i dati multimediali al client. E stata inoltre aggiunto il NAPDaemon che ha il compito di attivare e disattivare le funzionalità di un AP per quel dispositivo Bluetooth.

5.2 Client

In questo paragrafo sono descritti i dettagli implementativi e le scelte fatte sul client.

5.2.1 MainClient

Il MainClient è il componente principale del Client, in pratica è quello che costruisce tutte le altre parti. Questa classe nell'architettura preesistente

supportava solo l'handoff orizzontale per la tecnologia WiFi. Quindi è stato necessario estendere le sue funzionalità per fornirgli la possibilità di effettuare la gestione dell'handoff verticale, con gestione del rispettivo contesto. Inoltre e stato necessario modificare la gestione del ridimensionamento del buffer del proxy espandendo la gestione dello stato del client a più tecnologie. Lo stato è dato dalla probabilità dell'handoff orizzontale sulla tecnologia correntemente utilizzata.
In questa implementazione l'handoff verticale è possibile tra le tecnologie WiFi e Bluetooth, ma questo non comporta limitazioni ed è facilmente espandibile anche ad altre tecnologie.

5.2.1.1 Inizializzazione

Dato che l'implementazione preesistente funzionava su una sola tecnologia i costruttori non prendevano in considerazione la molteplicità dei dispositivi sia dalla parte del Client (schede wireless) sia dalla parte del Proxy (Acess Point). Per ovviare a questo sono stati introdotti 2 nuovi costruttori che estendono quelli precedenti :

- `public MainClient(long id,InetAddress addressWiFi, InetAddress addressBT, int serverPort, String WiFidevice, String BTdevice)`
- `public` **`MainClient`**`(long id,InetAddress addressWiFi, InetAddress addressBT, int serverPort, int samples, int window,int numXpred,int clientServiceClass,String WiFidevice, String BTdevice)`

Prendiamo in considerazione quello più generale spiegando a cosa si riferiscono i parametri :

- *id*: identificativo del Client (numero univoco associato a un Client)
- *addressWiFi*: indirizzo del AP WiFi sul Proxy
- *addressBT*: indirizzo del AP Bluetooth sul Proxy

- *serverPort*: porta del Proxy
- *samples*: numero di campioni per il modello Gray
- *window*: dimensione della finestra di previsione della scheda WiFi
- *numXpred*: numero di campioni per la previsione
- *clientServiceClass*: livello di priorità del Client (gold, silver o copper)
- *WiFidevice*: nome identificativo della scheda wireless (es. "eth1")
- *BTdevice*: nome identificativo della scheda bluetooth (es. "hci1")

Come vediamo nel costruttore ci servono 2 indirizzi distinti riguardanti il Proxy, uno per tecnologia WiFi e uno per Bluetooth. Si nota anche che si presuppone che il Proxy usi su entrambi gli indirizzi la stessa porta. Questi dati associati al Proxy non sono variabili nel tempo. Quindi il Proxy cambiando tecnologia usa sempre la coppia (addressWiFi,serverPort) oppure (addressBT,serverPort) per essere raggiunto.
Altri due parametri che sono stati aggiunti sono *WiFidevice e BTdevice* che sono i nomi identificativi delle rispettive interfaccie di rete dei dispositivi wireless (es. “eth1” per la scheda WiFi, “hci1” per la scheda Bluetooth). Questi parametri ci servono principalmente per l’inizializzazione del Manager e vengono passati al metodo di inizializzazione del MainClient denominato *init*:

```
private void init(String WiFidevice, String BTdevice)
```

All’inizio del metodo vengono inizializzati il Manager e il ManagerEventListener. Segue l’accensione di entrambi i dispositivi wireless del Client e il recupero delle informazione del contesto. Il Manager oltre a fornirci le informazioni di contesto ci fornisce anche dei metodi per l’accensione e spegnimento dei dispositivi.
La scelta della politica di gestione è operata nel seguente modo: se al momento è presente una scheda WiFi attiva di tipo WiFi allora si sceglie la politica Band Amplitude (viene data priorità alla scheda WiFi), se invece non è presente la

scheda WiFi, ma è presente la scheda Bluetooth si sceglie la politica Battery Consumption (viene data priorità alla scheda Bluetooth). Ovviamente una volta effettuata la scelta della politica viene automaticamente spento il dispositivo non prioritario.

Il codice seguente si riferisce proprio a questa situazione.

```
manager.Accendi(PolicyType.AllDevices());
wifiinfo = manager.getWifiInfo();
btinfo = manager.getBtInfo();

if (wifiinfo!=null) {
manager.Accendi(PolicyType.BandAmplitude());
manager.setPolicyType(PolicyType.BandAmplitude());
ipAdd = ipAddWIFI;
currentTech = WIFI;
}
else if (btinfo!=null) {
manager.Accendi(PolicyType.BatteryConsumption());
manager.setPolicyType(PolicyType.BatteryConsumption());
if (!manager.isConmandDaemonActive())  manager.startConmandDaemon();
connectToBTdevice();
ipAdd = ipAddBT;
currentTech = BLUETOOTH;
}
else if ((wifiinfo==null) && (btinfo==null)) manager.Spegni();
```

Notiamo che l'inizializzazione della tecnologia Bluetooth ha 2 istruzioni in più. La prima serve a avviare il demone Conmand, se non è stato gia avviato in precedenza, che a sua volta avvia il CLM (Connection and Location Manager) che responsabile della gestione dell'handoff orizzontale della tecnologia Bluetooth. La seconda serve a connettere il dispositivo Bluetooth all'AP Bluetooth del proxy.

Dopo l'inizializzazione, all'avvio del MainClient viene creata una socket di comunicazione con il Proxy sull'interfaccia di rete prescelta utilizzando la utility DatagramSocketFactory e inoltre viene inizializzato lo stato del client. Poi si procede con l'inizializzazione standard presente anche nell'architettura preesistente.

5.2.1.2 Gestione dello stato

La gestione dello stato del client serve a gestire al meglio le risorse del proxy richiedendo opportunamente il ridimensionamento del buffer del proxy. Al cambiamento dello stato del client si invia un messaggio di changeState al proxy che provvederà in dipendenza al parametro indicante lo stato del client ad aggiornare la dimensione del buffer del proxy. Lo stato del client è dato dalla sua probabilità dell'handoff orizzontale sulla tecnologia correntemente usata.
Per effettuare questa gestione è stato introdotto il metodo:

```
public void changeState(int state){
        if (this.state!=state) {
                this.state = state;
                changeState();
}}
```

Come vediamo questo metodo non fa altro che confrontare lo stato corrente con quello passato come parametro e se sono diversi aggiornare lo stato e invocare il metodo *changeState()* dell'architettura preesistente che provvederà a mandare il messaggio di changeState al proxy. Questa gestione serve a migliorare la gestione delle risorse sul proxy.

5.2.1.3 Procedura di Rebind

La gestione dell'handoff verticale consiste nel recuperare informazioni di contesto e con il verificarsi delle condizioni necessarie eseguire la procedura di Rebind.
Questa è divisa in due parti principali: nella prima parte si notifica il proxy della possibilità di un handoff e si modifica il client in tutte le parti interessate per poter continuare la esecuzione su una nuova tecnologia; nella seconda parte si mandano le informazioni necessarie al proxy affinché possa eseguire lui stesso la sua procedura di Rebind.

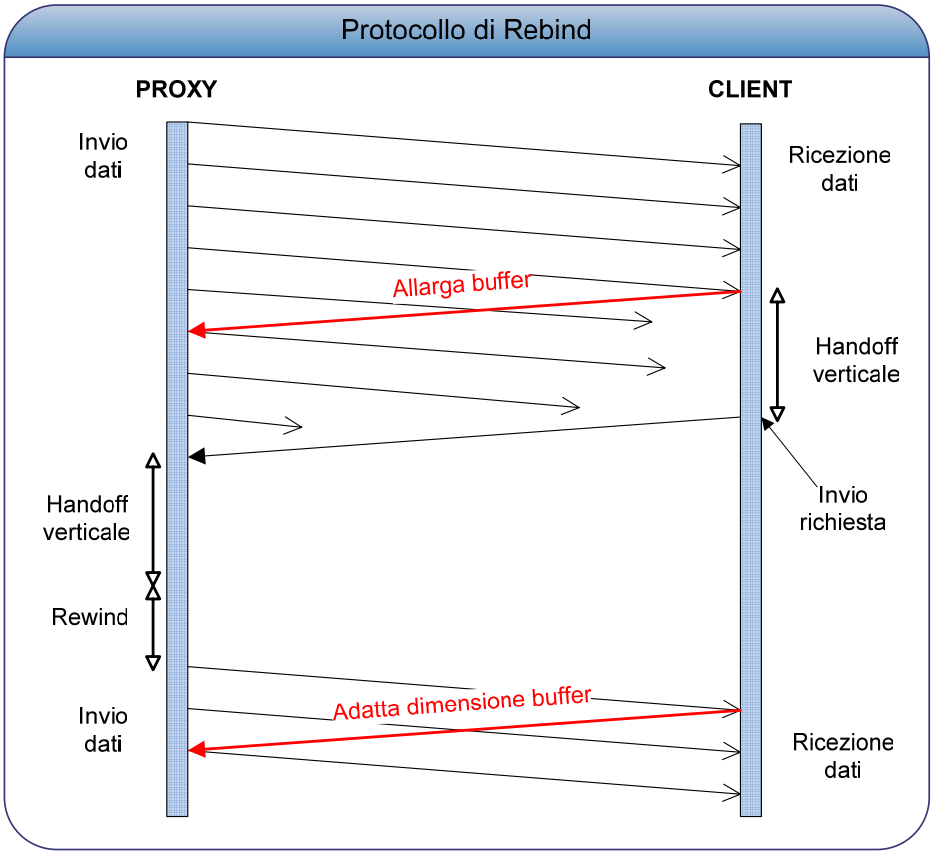

Figura 5.3: Protocollo di rebind

La procedura di Rebind viene iniziata dal ManagerEventListener il quale, al verificarsi delle condizioni per l'esecuzione dell'handoff verticale, invoca il metodo *rebind* del MainClient. Queste condizioni si possono verificare al

cambiamento della probabilità di handoff verticale, oppure al cambiamento dell'indirizzo del client (dovuto all'handoff verticale).

Una volta chiamato il metodo di rebind del MainClient come prima cosa si invia al proxy una richiesta di changeState per fargli allargare il suo buffer e successivamente si "propaga" l'invocazione del metodo di rebind a tutte le altre componenti interessate del client come si nota anche dalla figura 5.4. In tutti questi componenti vengono apportate opportune modifiche dovute al cambio dell'indirizzo IP sia dalla parte client, sia dalla parte proxy. Una volta terminate queste modifiche il client deve inviare al proxy, all'indirizzo corrispondente alla nuove tecnologia alla quale si vuole passare, tutte le informazioni necessarie per l'esecuzione della sua procedura di Rebind. Dopo l'invio di queste informazioni si aggiorna lo stato corrente del client sulla nuova tecnologia e si manda un altro messaggio di changeState al proxy. Questo messaggio serve a ridimensionare opportunamente il buffer del proxy in base allo stato client. Così facendo si opera una migliore gestione delle risorse del proxy.

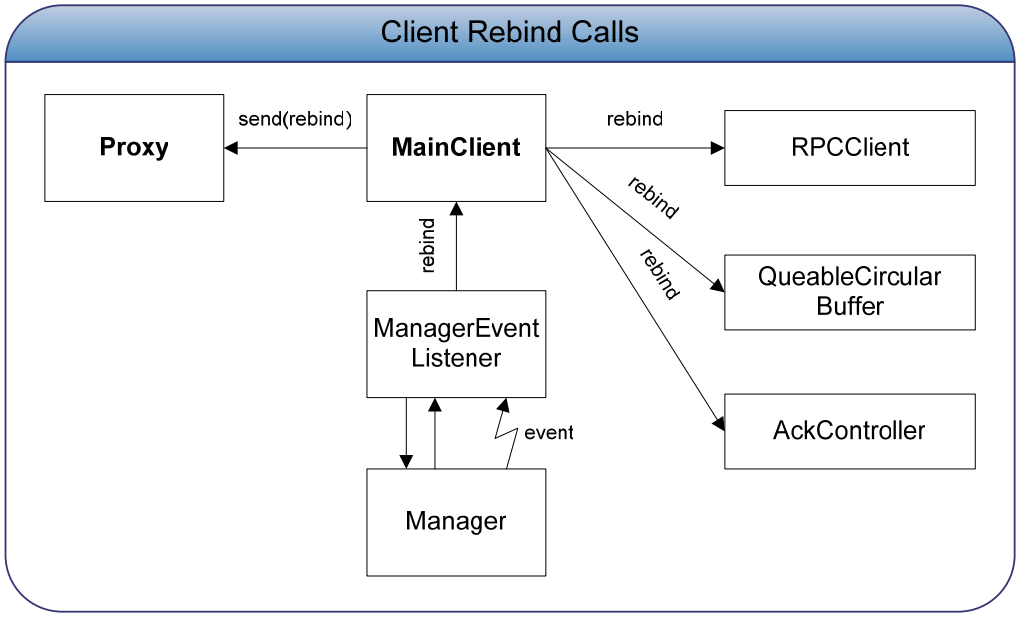

Figura 5.4: Chiamate di rebind sul client

In particolare, è necessario aggiornare il proxy con le seguenti informazioni:

- id: identificativo del client
- richiesta: numero che identifica il tipo di richiesta
- clientAddress: nuovo indirizzo IP del client
- clientPort: porta della nuova DatagramSocket del MainClient
- clientRTPport: nuova porta RTP del RPCClient
- tech: costante che identifica la tecnologia corrente
- timestamp: timestamp necessario per eseguire il rewind sul proxy

Per implementare l'intera procedura di Rebind sul client è stato predisposto il metodo rebind presente in tre versioni :

```
private void rebind(NetworkInterface ni, int tech)
```

Parametri:

- ni: istanza della classe NetworkInterface rappresentante il dispositivo della tecnologia alla quale si vuole passare
- tech: intero rappresentante la tecnologia alla quale si vuole passare

Questo è il metodo principale. Come prima cosa crea una DatagramSocket temporanea sull'interfaccia di rete passata come parametro utilizzando la utility DatagramSocketFactory. Poi aggiorna l'indirizzo del proxy e invia la richiesta di changestatus per fargli allargare il proprio buffer, ciò consentirà di gestire un rewind di dimensioni maggiori. Poi aggiorna le altre variabili e invoca il metodo rebind dei componenti RPCClient, QueableCircularBuffer e AckController. Alla fine chiude la vecchia DatagramSocket e la sostituisce con quella temporanea creata in precedenza.

```
tmpSock = DatagramSocketFactory.createDatagramSocket(ni);


currentTech = tech;
currentNetworkInterface = ni;
if (currentTech==WIFI) ipAdd = ipAddWIFI;
```

```
else if (currentTech==BLUETOOTH) ipAdd = ipAddBT;

InetAddress clientAddress = tmpSock.getLocalAddress();
inRTPport=PortManager.getRTPport();
int clientport = tmpSock.getLocalPort();

rpc.rebind(clientAddress,clientport,ipAdd,senderRTPport);
long timestamp=appoggio.getLastTimeStamp();
appoggio.rebind(ni,ipAdd);
ack.rebind(ni,ipAdd);
sockUDP.close();
sockUDP = tmpSock;
```

`public void rebind(String device, int tech)`

Parametri:

- device: nome identificativo dell'interfaccia di rete associata alla tecnologia alla quale si vuole passare (es. "eth1", "hci1")
- tech: intero rappresentante la tecnologia alla quale si vuole passare

Questo metodo viene invocato quando il ManagerEventListener decide di effettuare l'handoff verticale. Esso recupera l'istanza di NetworkInterface associata al nome passato come argomento e il numero rappresentante la tecnologia alla quale si vuole passare e successivamente richiama il metodo principale.

`public void rebind (InetAddress clientAddress)`

Parametro:

- clientAddress: nuovo indirizzo IP del client

Questo metodo viene invocato quando si verifica un cambiamento dell'IP del Client. Esso recupera l'istanza di NetworkInterface associata all'indirizzo passato

come argomento e il numero rappresentante la tecnologia alla quale si vuole passare e successivamente richiama il metodo principale.

Questa gestione dell'handoff verticale cerca di ridurre al minimo il costo dell'handoff cercando di applicare una sovrapposizione temporale delle tecnologie per quanto possibile. Questo vuole significare che il client mantiene il contatto con il proxy ricevendo le informazioni usando la "vecchia" tecnologia il più a lungo possibile. Questo è lo scopo ad esempio dell'utilizzo della DatagramSocket temporanea durante il rebind. Come si può notare questa gestione è di tipo Hard Proactive. Si è scelto una gestione di questo tipo per minimizzare il consumo della batteria sul client.

Nel protocollo di Rebind che segue questa implementazione, come si nota anche dalla seguente figura, i dati multimediali mandati dal proxy e ricevuti dal client vengono sempre inviati usando solo una tecnologia, anche se sul client è possibile effettuare controlli anche avendo attive entrambe le tecnologie. Ovviamente questo rende possibile la perdita di alcuni dati durante l'esecuzione dell'handoff verticale.

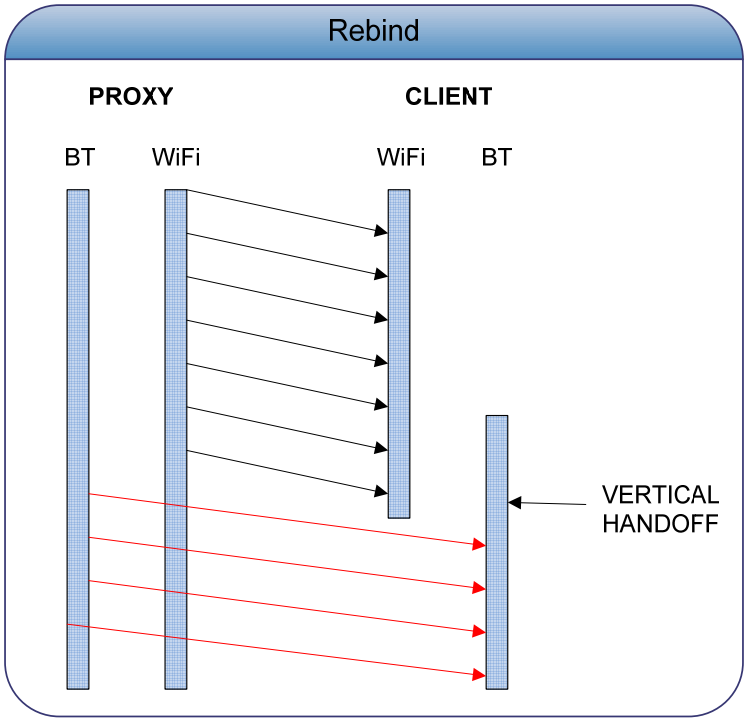

Figura 5.5: Rebind

5.2.2 ManagerEventListener

Il ManagerEventListener è il componente che si interpone fra il MainClient e il Manager ed e quello che incapsula le politiche di gestione dell'handoff verticale del client. Inoltre effettua la gestione del cambiamento dello stato del client. Viene inizializzato dal MainClient durante la sua inizializzazione. Come prima cosa si deve registrare presso il Manager per ricevere gli eventi sul cambiamento del contesto. Gli eventi possibili sono: il cambiamento dell'indirizzo del client, il cambiamento della probabilità dell'handoff su WiFi (Bluetooth) e il cambiamento della locazione.

Per registrare il ManagerEventListener presso il Manager per poter ricevere gli eventi il MainClient deve invocare il metodo *addListeners* passando come argomento l'istanza di Manager e per togliersi dalla lista dei registrati bisogna invocare il metodo *removeListeners*.

```
public void addListeners(Manager manager){
if (manager!=null){
    manager.addClientAddressListener(this);
    manager.addLocationListener(this);
    manager.addHandoffProbListener(this);
    manager.addHandoffProbBTListener(this);
    }}

public void removeListeners(Manager manager){
if (manager!=null){
    manager.removeClientAddressListener(this);
    manager.removeLocationListener(this);
    manager.removeHandoffProbListener(this);
    manager.removeHandoffProbBTListener(this);
    }}
```

5.2.2.1 Client Address Change

Questo metodo gestisce l'evento che si scatena quando il client cambia indirizzo IP. Questa situazione può verificarsi ad es. durante un'handoff orizzontale. Come prima cosa il metodo recupera le informazioni necessarie dall'evento, in questo caso il nuovo indirizzo IP. L'evento contiene un array contenente gli indirizzi IP che sono cambiati. Poi si richiama il metodo rebind del MainClient passando l'indirizzo cambiato (nel nostro caso il primo del array).

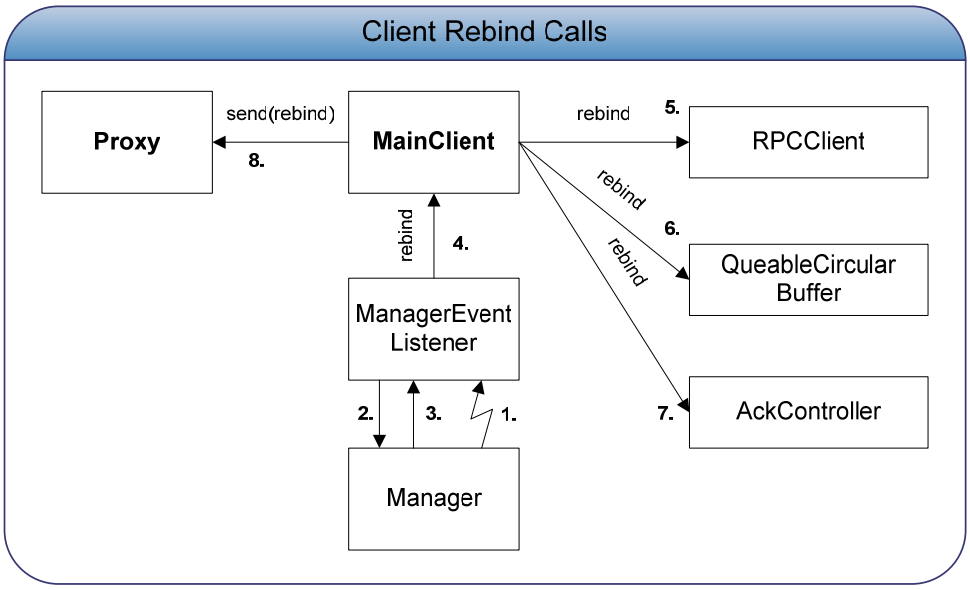

Figura 5.6: Sequenza delle chiamate di rebind sul client

5.2.2.2 WiFi Handoff Probability Change

Questo metodo gestisce l'evento che si scatena quando cambia la probabilità dell'handoff orizzontale del client sulla tecnologia WiFi. Questo evento si verifica solo quando è in utilizzo la tecnologia WiFi.

All'inizio del metodo si recuperano le informazioni necessarie dall'evento, in questo caso l'istanza del Manager che ha scatenato l'evento e la probabilità dell'handoff orizzontale (passo 1 in Figura 5.6). Poi si invoca il metodo *changeState()* del MainClient passando come parametro la probabilità dell'handoff orizzontale. Tale metodo provvederà ad aggiornare lo stato del client. Dopo, in

dipendenza dalle politiche di gestione e dalle condizioni necessarie, è possibile effettuare l'handoff verticale da WiFi a Bluetooth (passo 4 in Figura 5.6). Per astrarre dalle specifiche implementazioni delle schede wireless la qualità del segnale è stata divisa in fasce di valore simbolico alto, medio e basso.

Se la politica di gestione è Band Amplitude, la tecnologia WiFi è prioritaria, allora le condizioni per l'handoff verticale sono: la probabilità dell'handoff orizzontale è alta, la qualità del segnale WiFi deve essere a livello simbolico basso e la qualità del segnale Bluetooth deve essere almeno a livello simbolico medio.

Invece se la macropolitica di gestione è Battery Consumption, la tecnologia Bluetooth è prioritaria, le condizioni per l'handoff verticale sono: la qualità del segnale Bluetooth deve essere almeno a livello simbolico medio.

Queste informazioni sulla probabilità dell'handoff orizzontale o sulle qualità del segnale si ottengono interrogando il Manager (passi 2 e 3 in Figura 5.6).

Un aspetto positivo da sottolineare è che durante i controlli effettuati dalle politiche di gestione entrambe le schede sono attive. Così facendo, se non si verifica la condizione necessaria per effettuare l'handoff verticale, si spegne la scheda dell'altra tecnologia, e la comunicazione con il proxy non viene interrotta. Invece, se si verifica la condizione necessaria per l'handoff, allora la scheda che veniva usata prima dell'handoff verrà spenta solo dopo l'esecuzione del metodo rebind del MainClient in modo da realizzare un hard handoff. Così facendo, ritardiamo il più possibile l'interruzione della comunicazione con il proxy.

5.2.2.3 Bluetooth Handoff Probability Change

Questo metodo gestisce l'evento che si scatena quando cambia la probabilità dell'handoff orizzontale del client sulla tecnologia Bluetooth. Inoltre questo evento di dovrebbe verificare solo quando è in utilizzo la tecnologia Bluetooth.

All'inizio del metodo si recuperano le informazioni necessarie dall'evento, in questo caso l'istanza del Manager che ha scatenato l'evento e la probabilità dell'handoff orizzontale (passo 1 in Figura 5.6). Poi si invoca il metodo

changeState() del MainClient passando come parametro la probabilità dell'handoff orizzontale. Tale metodo provvederà ad aggiornare lo stato del client. Dopo, in dipendenza dalle politiche di gestione e dalle condizioni necessarie, è possibile effettuare l'handoff verticale da Bluetooth a WiFi (passo 4 in Figura 5.6).
Se la politica di gestione è Band Amplitude, la tecnologia WiFi è prioritaria, allora le condizioni per l'handoff verticale sono: la qualità del segnale WiFi deve essere almeno a livello simbolico medio.
Invece se la politica di gestione è Battery Consumption, la tecnologia Bluetooth è prioritarià, le condizioni per l'handoff verticale sono: la probabilità dell'handoff orizzontale è alta, la qualità del segnale Bluetooth deve essere a livello simbolico basso e la qualità del segnale WiFi deve essere almeno a livello simbolico medio
Queste informazioni sulla probabilità dell'handoff orizzontale o sulle qualità del segnale si ottengono interrogando il Manager (passi 2 e 3 in Figura 5.6).
Per quanto riguarda l'attivazione contemporanea delle schede di entrambe le tecnologie durante il controllo delle condizioni necessarie per l'handoff verticale valgono le stesse considerazione fatte per il metodo analogo sopra riportato.

5.2.2.4 Location Change

Questo metodo gestisce l'evento che si scatena quando il client cambia locazione ovvero cambia AP sul quale è connesso. Questo renderebbe il client location-aware, cioè consapevole della locazione nella quale opera ed esso potrebbe modificare il proprio comportamento in base al cambiamento della locazione.
Per la nostra applicazione questo non da un grosso contributo dato che il proxy ha due AP statici e il client usa solo quelli. Comunque, per un possibile sviluppo futuro si potrebbe implementare questo metodo per rendere il client location-aware.

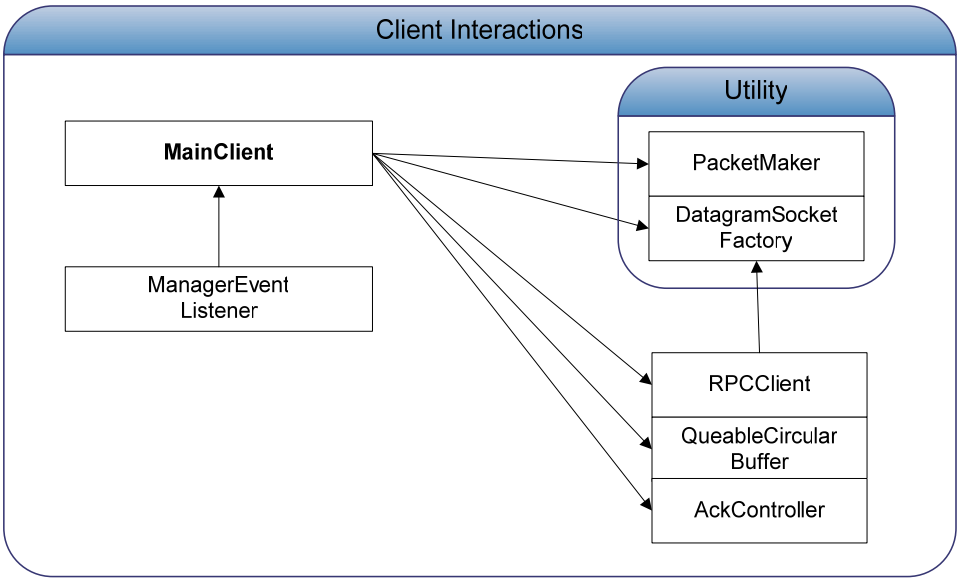

Figura 5.7: Interazioni nel client

5.2.3 RPCClient

Questo componente è un listener di eventi che descrivono l'arrivo dei dati multimediali tramite protocollo RTP dal proxy ed adegua la ricezione dei dati a seconda di questi eventi. Per effettuare questo deve conoscere gli indirizzi e le porte usate dai client e proxy per trasferire questi dati. Nell'implementazione preesistente c'era solo la possibilità di funzionare entro una tecnologia.

A questo componente sono state fatte due modifiche a finche possa funzionare con più di una tecnologia. La prima modifica è stata aggiungere un nuovo costruttore estendendo quello esistente :

```
public RPCclient(String senderAddress, int senderPort, int controlPort, InetAddress localAddress, int localPort, QueableCircularBuffer appoggio)
```

Questo costruttore aggiunge a quello esistente il parametro localAddress che rappresenta l'indirizzo IP correntemente usato dal client e l'inizializzazione del RTPManager iniziale dovrà avvenire con questo indirizzo e non come prima con InetAddress.getLocalHost().

La seconda modifica che è stata fatta è aggiungere il metodo rebind :

`public void rebind(InetAddress clientAddress, int clientport, InetAddress serverAddress,int serverport)`

Parametri:

- clientAddress: indirizzo IP del client
- clientport: numero di porta del client
- serverAddress: indirizzo IP del proxy
- serverport: numero di porta del proxy

Questo metodo viene richiamato dal MainClient quando si effettua un'handoff verticale.

```
if (!clientAddress.getHostAddress().equals(localAddr.getDataHostAddress())){
    localAddr= new SessionAddress(clientAddress, clientport ,1);
    mgr = RTPManager.newInstance();   //rtpManager
    mgr.addReceiveStreamListener(this);
    mgr.initialize(localAddr);
    }

if      (!destAddr.getDataHostAddress().equals(serverAddress.getHostAddress()))
destAddr = new SessionAddress(serverAddress,serverport);

mgr.addTarget(destAddr);
```

Dal codice sopra riportato vediamo il funzionamento del rebind nel RPCClient. Se l'indirizzo del client passato come parametro è diverso dal precedente si crea un nuovo indirizzo di sessione locale e una nuova istanza del RTPManager e lo si inizializza. Poi se l'indirizzo del proxy passato come parametro è diverso dal precedente si crea un nuovo indirizzo di sessione per la destinazione. Alla fine si aggiunge al manager l'indirizzo di sessione per la destinazione.

5.2.4 QueableCircularBuffer

Questo componente viene usato come buffer tra la ricezione dei dati multimediali e la loro riproduzione. Nell'implementazione preesistente non veniva considerata la possibilità di cambiare indirizzo sia da parte del client sia da parte del server. Per aggiungere questa funzionalità è stato necessario operare una modifica, cioè introdurre il metodo rebind.

`public void rebind(NetworkInterface ni, InetAddress senderAddress)`

Parametri:

- ni: istanza della classe NetworkInterface rappresentante l'interfaccia di rete alla quale si vuole passare
- senderAddress: indirizzo IP del server

Questo metodo oltre a fungere da metodo di rebind viene utilizzato dal MainClient anche durante l'inizializzazione dell'istanza di QueableCircularBuffer per creare una DatagramSocket per la stessa istanza.

```
if (sock!=null) sock.close();
sock = DatagramSocketFactory.createDatagramSocket(ni);


if (!senderAddress.equals(this.SenderAddress)) setSenderAddress(senderAddress);
```

Come vediamo dal codice il comportamento è abbastanza semplice. Se una DatagramSocket esiste gia la chiude e ne crea un'altra sull'interfaccia di rete specificata come parametro. Poi se l'indirizzo IP del Proxy è differente di quello attuale lo aggiorna.

5.2.5 AckController

L'AckController è il componente del client che ha il compito di inviare, con una certa frequenza, segnali di Acknowledge al proxy, cioè fa sapere al proxy che il client è raggiungibile. Nell'implementazione preesistente non veniva considerata la possibilità di cambiare indirizzo sia da parte del Client sia da parte del Server. Per aggiungere questa funzionalità è stato necessario operare una modifica, cioè introdurre il metodo rebind.

`public void rebind(NetworkInterface ni, InetAddress senderAddress)`

Parametri:

- ni: istanza della classe NetworkInterface rappresentante l'interfaccia di rete alla quale si vuole passare
- senderAddress: indirizzo IP del server

```
if (sock!=null) sock.close();
sock = DatagramSocketFactory.createDatagramSocket(ni);


if (!senderAddress.equals(this.serverIp)) this.serverIp=senderAddress;
```

Come vediamo dal codice il comportamento è abbastanza semplice. Se una DatagramSocket esiste gia la chiude e ne crea un'altra sull'interfaccia di rete specificata come parametro. Poi se l'indirizzo IP del proxy è differente di quello attuale lo aggiorna.

5.2.6 DatagramSocketFactory

La DatagramSocketFactory è una ulitity ulitizzata da alcuni componenti del client. Come lo stesso nome fa supporre si tratta di una Factory di DatagramSocket. È un

componente statico e quindi ha solo metodi statici al suo interno. Contiene un metodo sovraccarico che come risultato restituisce una DatagramSocket nuova.

```
public static DatagramSocket createDatagramSocket(
NetworkInterface ni)
```

Parametro:

- ni: istanza di NetworkInterface rappresentante l'interfaccia di rete

Questo metodo restituisce una nuova DatagramSocket creata sulla specifica interfaccia di rete passata come parametro. Come indirizzo IP si prende il primo indirizzo associato a quella interfaccia di rete.

```
public static DatagramSocket createDatagramSocket(
String device, InetAddress addr)
```

Parametri:

- device: nome identificativo dell'interfaccia di rete
- addr: indirizzo IP sul quale si vuole creare una DatagramSocket

Questo metodo restituisce una nuova DatagramSocket creata sulla specifica interfaccia di rete specificata dal nome passato come parametro con indirizzo IP passato come parametro. Viene effettuato anche il controllo che l'indirizzo IP passato come parametro sia effettivamente un indirizzo associato a quella interfaccia di rete.

5.2.7 NetworkInterface

La classe NetworkInterface fa parte del package standard *java.net*. Fornisce dei metodi statici utili come ricavare l'istanza dell'interfaccia di rete passando come parametro il nome identificativo dell'interfaccia oppure passando come parametro l'indirizzo IP. Ogni istanza rappresenta un'interfaccia di rete specifica. Noi usiamo il metodo che ci restituisce una enumerazione di tutti gli indirizzi IP associati a quel interfaccia. Ma esistono un'altra serie di informazioni sull'interfaccia che questa istanza ci può fornire.

Questa classe viene usata durante la procedura di Rebind ed i metodi usati sono qui sopra citati.

5.2.8 PacketMaker

Il componente PacketMaker è una utility gia esistente nell'implementazione preesistente. Questa classe ci fornisce i metodi per creare i DatagramPacket diversi a seconda della richiesta che il client deve fare al proxy.

Ovviamente volendo comunicare al Proxy che deve eseguire la propria procedura di Rebind quando si vuole effettuare un'handoff verticale si rende necessario poter costruire un pacchetto per quella richiesta.

Si è aggiunto il metodo che crea il pacchetto necessario:

```
public static DatagramPacket rebind(long id, int request, InetAddress address, int port, String clientAddress, int clientPort,int clientRTPPort,int tech, long timestamp)
```

Parametri:

- id: identificativo del client
- request: numero che identifica il tipo di richiesta
- address: indirizzo del proxy sul quale mandare il pacchetto
- port: porta del proxy sulla quale mandare il pacchetto
- clientAddress: nuovo indirizzo IP del client
- clientPort: porta della nuova DatagramSocket del MainClient
- clientRTPport: nuova porta RTP del RPCClient
- tech: numero che identifica la tecnologia corrente
- timestamp: timestamp necessario per eseguire il rewind sul proxy

Si è inoltre dovuta effettuare una ulteriore modifica nel metodo *setProfileRequest* al quale è stato aggiunto il parametro *tech* che rappresenta la tecnologia adottata. Questo metodo viene usato dal client all'inizio della comunicazione per informare

il proxy sul proprio profilo. Mancando prima un informazione riguardante la tecnologia è chiara la necessità di questa modifica.

5.3 Proxy

In questo paragrafo sono descritti i dettagli implementativi e le scelte fatte sul proxy.

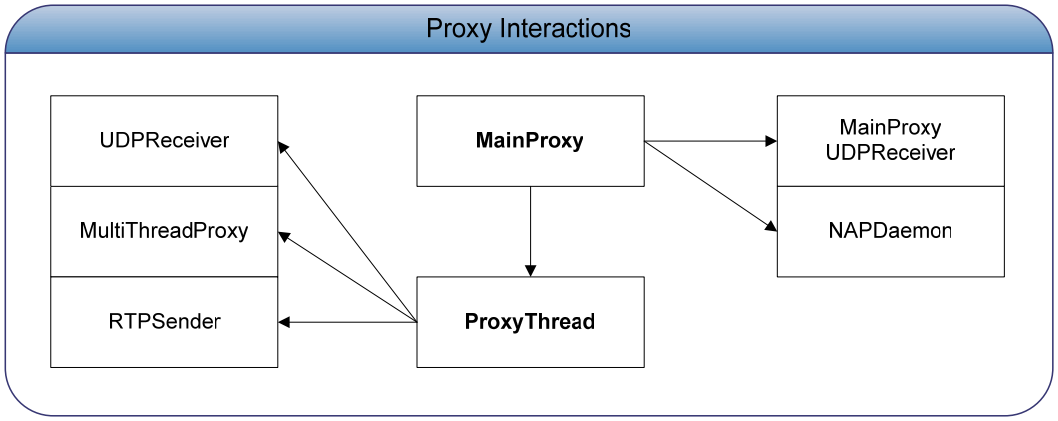

Figura 5.8: Interazioni nel proxy

5.3.1 MainProxy

Il componente MainProxy implementa il proxy generico dell'architettura astratta. Il suo compito è quello di ricevuta la prima richiesta da parte del client associare al client stesso un proxy specifico per tutta la sua esecuzione. Questa classe nell'architettura preesistente supportava solo la comunicazione su una tecnologia (WiFi). Per aggiungere la funzionalità di poter ricevere richieste su due tecnologie diverse si e dovuto svincolare il MainProxy dall'ascolto vero e proprio delle richieste dato che l'ascolto è possibile solo su una tecnologia, questo dovuto al metodo bloccante *receive* della socket. Si è ritenuto necessario creare una nuova entità, il MainProxyUDPReceiver, che abbia il compito di stare in ascolto delle richieste dei client su una tecnologia. Quindi per il MainProxy tutto si risolve creando due istanze di MainProxyUDPReceiver, ognuna in ascolto su una

tecnologia diversa. All'arrivo della richiesta del client il MainProxyUDPReceiver passa la richiesta stessa al MainProxy che la gestisce associando il proxy specifico al client.

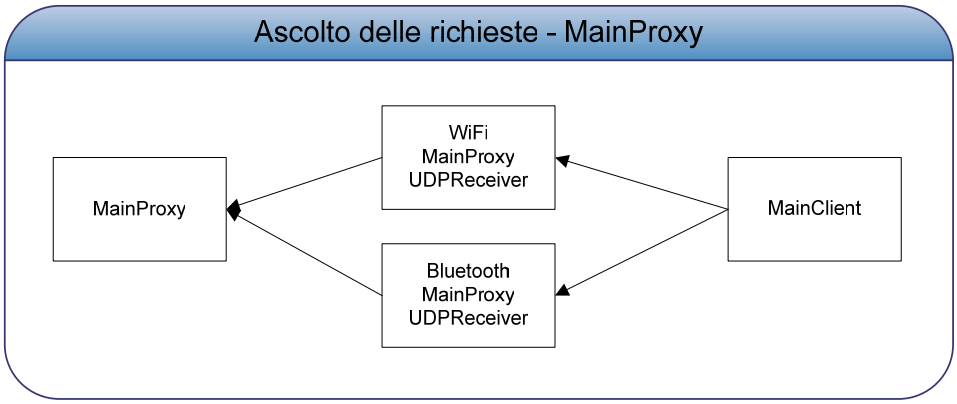

Figura 5.9: MainProxy – ascolto delle richieste

5.3.1.1 Inizializzazione

Oltre alla modifica sull'ascolto delle richieste dei client viene modificata anche l'inizializzazione stessa. Viene modificato il costruttore aggiungendogli il nome del dispositivo Bluetooth con il quale si vuole ricevere richieste dal client.

```
public MainProxy(int port, String btDeviceName)
```

Parametri:

- port: numero di porta sulla quale si vuole creare la socket
- btDeviceName: nome del dispositivo Bluetooth (es. hci0)

A questo costruttore infine è stata aggiunta l'invocazione del metodo *initBluetooth()* passando come parametro il nome del dispositivo Bluetooth. Questo metodo invoca il metodo statico *start()* del NAPDaemon e serve ad attivare le funzionalità di un AP per quel dispositivo Bluetooth e viene effettuato solo se si opera sotto il sistema operativo Linux. Il NAPDaemon offre funzionalità per creare nuove connessioni di livello datalink per Bluetooth durante gli handoff orizzontali.

Per spegnere i processi attivati dal metodo initBluetooth() viene aggiunto il metodo closeBluetooth(), con parametro il nome del dispositivo Bluetooth, che viene invocato alla chiusura del MainProxy. Questo metodo invoca il metodo *stop()* del NAPDaemon.

5.3.1.2 Ascolto delle richieste

Per poter effettuare l'ascolto delle richieste all'avvio del MainClient bisogna creare le due socket sulle quali si eseguirà l'ascolto per una specifica tecnologia e creare le due istanze di MainProxyUDPReceiver ognuna associata ad una differente socket.

È stato aggiunto un metodo che viene richiamato dal MainProxyUDPReceiver al momento della ricezione della richiesta da parte del client.

```
public void PacketRecived(DatagramPacket packet)
```

Parametro:

- packet: pacchetto ricevuto contenete la richiesta del client

Questo metodo non fa altro che associare al client un proxy specifico e a passare il pacchetto ricevuto al proxy specifico che provvederà a gestirlo.

5.3.2 ProxyThread

Il componente ProxyThread implementa il proxy specifico dell'architettura astratta. Questo componente viene creato per uno specifico client dal MainProxy all'arrivo della prima richiesta del client. Il suo compito è gestire tutte le richieste del client e di adattarsi al suo funzionamento per migliorare l'erogazione del servizio. Nell'architettura preesistente il ProxyThread comunicava usando solo una tecnologia wireless (WiFi) ed è quindi stato necessario apportare le modifiche per aggiungere la possibilità di lavorare su più tecnologie.

Per aspettare l'arrivo delle richieste veniva usata un'istanza di UDPReceiver. Questo componente sta in ascolto su una socket aspettando le richieste del client e

al loro arrivo le passa al ProxyThread che provvederà a gestirle. Per aggiungere la possibilità di comunicare con due tecnologie il ProxyThread utilizzerà due istanze di UDPReceiver, ognuna per una tecnologia diversa.

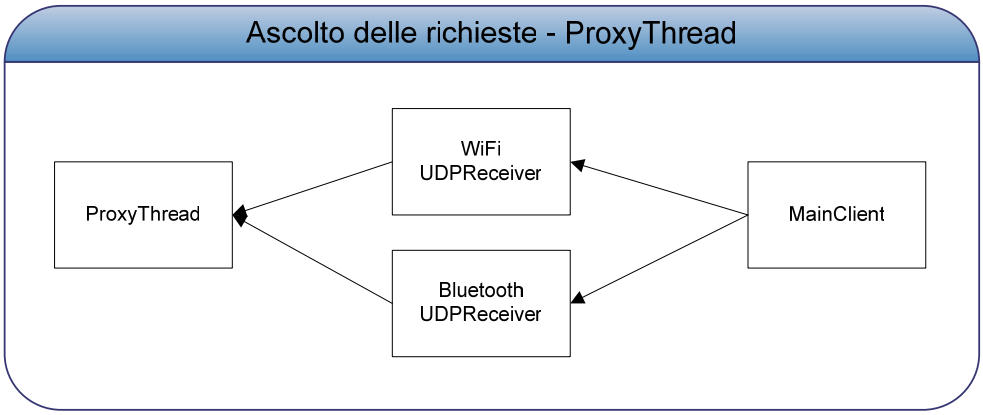

Figura 5.10: ProxyThread – ascolto delle richieste

Inoltre sono state apportate delle modifiche ai metodi del ProxyThread: *setClientProfile()*, relativo al settaggio del profilo del cliente, e *streamRequest()*, relativo alla richiesta dell'invio di dati multimediali. Nel metodo *setClientProfile()* vengono create le due socket delle rispettive tecnologie e viene scelta fra queste due quella da usare correntemente per comunicare in base alla tecnologia utilizzata dal client. Nel metodo *streamRequest()* vengono create e avviate le due istanze di UDPReceiver che si pongono in attesa delle richieste del client.

```
receiverWiFi = new UDPReceiver(this,sockUDPWiFi);
receiverBT = new UDPReceiver(this,sockUDPBT);
receiverWiFi.start();
receiverBT.start();
```

5.3.2.1 Procedura di Rebind

La procedura di rebind è la procedura che si utilizza quando si vuole eseguire un'handoff verticale e consiste nel aspettare la richiesta di rebind da parte del client è adattare il proprio funzionamento con la propria procedura di rebind. Il

Proxy si adatta spostando tutta la propria comunicazione con il client sulla nuova tecnologia richiesta dal client stesso. Nella figura 5.11 vediamo come il ProxyThread, ricevendo la richiesta di rebind da parte del client (passo 1), esegue la propria procedura di rebind che si propaga a tutti i componenti interessati (passi 2 e 3).

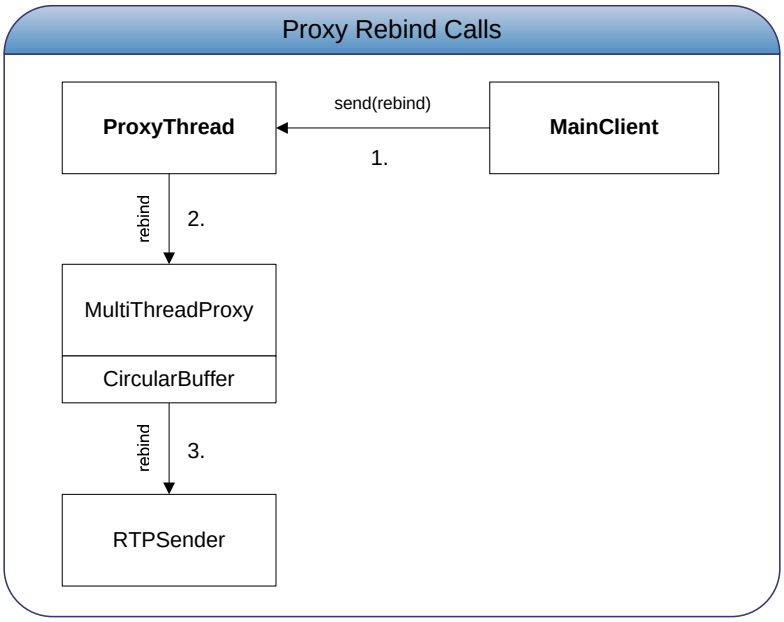

Figura 5.11: Chiamate di rebind sul proxy

In particolare, le informazioni necessarie al proxy per effettuare la propria procedura di rebind sono:

- clientAddress: nuovo indirizzo IP del client
- clientPort: porta della nuova DatagramSocket del MainClient
- clientRTPport: nuova porta RTP del RPCClient
- tech: numero che identifica la tecnologia usata dal client
- timestamp: timestamp necessario per eseguire il rewind sul proxy

Per implementare la procedura di rebind è stato predisposto il seguente metodo rebind che estrae dalla richiesta le informazioni necessario e poi esegue l'aggiornamento di tutti i componenti interessati.

```
if (tech!=currentTech) {
        if (tech==WIFI) sockUDP=sockUDPWiFi;
        else if (tech==BLUETOOTH) sockUDP=sockUDPBT;

proxy.rebindProxy(sockUDP.getLocalAddress(),sockUDP.getLocalPort(),
clientAddress,clientRTPport);
        currentTech = tech;
}
else proxy.rebindClient(clientAddress,clientRTPport);
// effettua anche il rewind con il timestamp passato dal client
rewind();
```

Come vediamo dal codice sopra riportato come prima cosa si aggiorna la socket usata dal proxy in base alla tecnologia usata dal client. Poi si verifica se è cambiata la tecnologia usata dal client, ovvero se si vuole eseguire un handoff verticale, e se è questo il caso allora si invoca il metodo *rebindProxy()* del MultiThreadProxy. Se invece la tecnologia non è cambiata, caso nel quale è cambiato solo l'indirizzo IP del client ma non si intende cambiare tecnologia, allora si invoca il metodo *rebindClient()* del MultiThreadProxy.

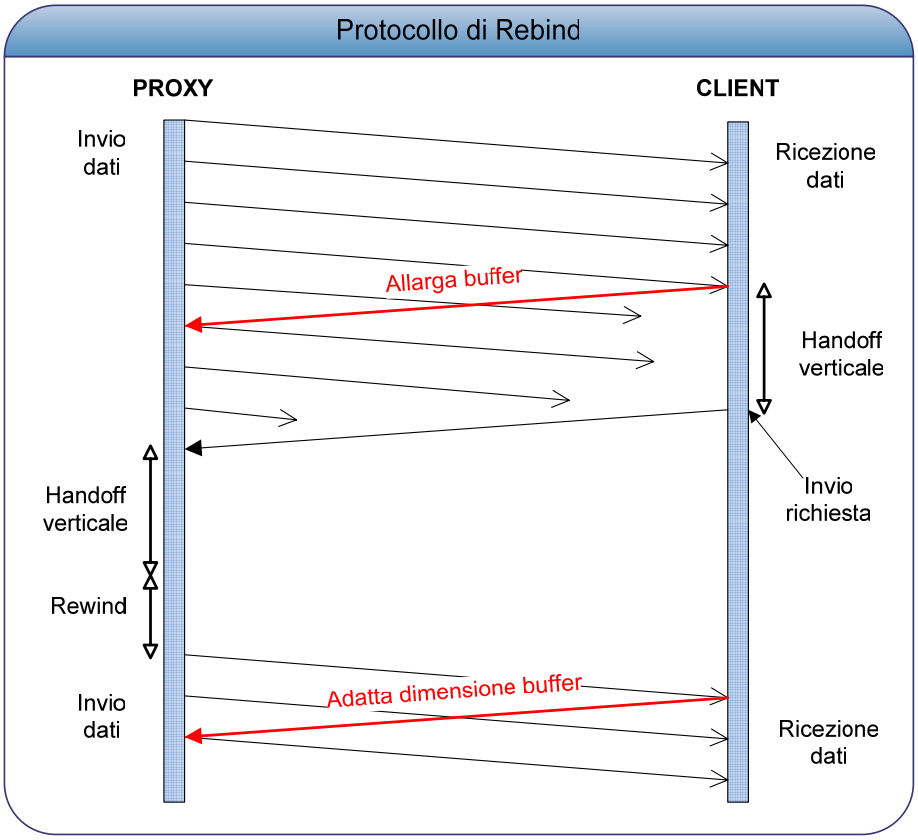

Figura 5.12: Protocollo di rebind

Il metodo *rebindClient()* aggiorno nei componenti interessati solo le informazioni relative a indirizzi e porte del client mentre il metodo *rebindProxy()* aggiorna sia le informazioni relative al client sia le informazioni relative al proxy, dato che si vuole cambiare tecnologia di comunicazione. Alle fine del metodo viene eseguito il rewind del client ovvero il proxy deve cominciare a mandare le informazioni usando la nuova tecnologia dall'ultimo dato ricevuto dal client.

Questo viene fatto perché durante l'esecuzione dell'handoff verticale si può verificare una perdita di dati da parte del client e quindi è necessario ricominciare la trasmissione dei dati multimediali dall'ultimo dato ricevuto dal client. Per "riavvolgere" il flusso di dati multimediali e quindi ripartire dal dato corretto viene usato il timestamp presente nella richiesta di rebind. Questo timestamp indica proprio l'ultimo dato ricevuto dal client. Questa procedura di "riavvolgimento" del flusso di dati multimediali nel buffer del proxy viene detta procedura di rewind.

Prima di questa procedura di rebind il client manda un messaggio che fa allargare il buffer del proxy, e alla fine della procedura rimanda un altro messaggio di adattamento della dimensione del buffer. Questo serve al proxy per poter fornire un rewind più esteso dei dati.

5.3.3 MultiThreadProxy

Il componente MultiThreadProxy si occupa dalla trasmissione del flusso di dati multimediali dal server al client. Nell'architettura preesistente non era previsto il cambiamento degli indirizzi e delle porte sia del client sia del proxy dovendo funzionare solo su una tecnologia. Quindi si è dovuta aggiungere la possibilità di cambiare indirizzi e porte dovendo funzionare in un contesto a più tecnologie. Per implementare questa funzionalità sono stati aggiunti due metodi:

```
public void rebindClient(String clientAddress, int clientRTPPort)
```

Parametri:

- clientAddress: nuovo indirizzo IP del client
- clientRTPPort: nuova porta RTP del client

Questo metodo viene invocato quando non viene cambiata la tecnologia ma cambia l'indirizzo del client. Come vediamo anche dal codice sottostante, non fa altro che aggiornare i valori identificati da questi parametri e ad invocare lo stesso metodo sull'istanza di RTPSender (transmitter) attiva passando come parametri i nuovi indirizzo e porta e i vecchi indirizzo e porta del client.

```
transmitter.rebindClient(InetAddress.getByName(clientIP),this.clientPort,InetAddress.getByName(clientAddress),clientRTPPort);

this.clientIP = clientAddress;
this.clientPort = clientRTPPort;
```

```
public void rebindProxy(InetAddress proxyAddress, int
proxyPort, String clientAddress, int clientRTPPort)
```

Parametri:

- proxyAddress: nuovo indirizzo IP del proxy
- proxyPort: nuova porta del proxy
- clientAddress: nuovo indirizzo IP del client
- clientRTPPort: nuova porta RTP del client

Questo metodo viene invocato quando si effettua un'handoff verticale e quindi cambiano sia gli indirizzo e porta del proxy che quelli del client. Come vediamo anche dal codice sottostante, non fa altro che aggiornare i valori identificati da i parametri relativi al client e ad invocare lo stesso metodo sull'istanza di RTPSender (transmitter) attiva passando come parametri i nuovi indirizzo e porta del proxy e quelli del client.

```
transmitter.rebindProxy(proxyAddress,proxyPort,
InetAddress.getByName(clientAddress), clientRTPPort);


this.clientIP = clientAddress;
this.clientPort = clientRTPPort;
```

5.3.4 RTPSender

Il componente RTPSender fa parte del componente MultiThreadProxy e svolge il compito di inviare al client il flusso di dati multimediali ricevuti dal server. Nell'architettura preesistente non era previsto il cambiamento degli indirizzi e delle porte sia del client sia del proxy dovendo funzionare solo su una tecnologia. Quindi si è dovuta aggiungere la possibilità di cambiare indirizzi e porte dovendo funzionare in un contesto a più tecnologie. Per implementare questa funzionalità sono stati aggiunti due metodi:

`public void rebindClient(InetAddress oldClientAddress, int oldClientPort, InetAddress newClientAddress, int newClientPort)`

Parametri:

- oldClientAddress: vecchio indirizzo IP del client
- oldClientPort: vecchia porta RTP del client
- newClientAddress: nuovo indirizzo IP del client
- newClientPort: nuova porta RTP del client

Questo metodo viene invocato quando non viene cambiata la tecnologia ma cambia l'indirizzo del client. Come vediamo anche dal codice sottostante, non fa altro che rimuovere la vecchia destinazione dell'invio di dati multimediali dal RTPManager invocando il metodo *removeTarget()* e ad aggiungere una nuova destinazione invocando il metodo *addDestination()*.

```
removeTarget(oldClientAddress,oldClientPort);
addDestination(newClientAddress,newClientPort);
```

`public void rebindProxy(InetAddress proxyAddress, int proxyPort, InetAddress clientAddress, int clientPort)`

Parametri:

- proxyAddress: nuovo indirizzo IP del proxy
- proxyPort: nuova porta del proxy
- clientAddress: nuovo indirizzo IP del client
- clientPort: nuova porta RTP del client

Questo metodo viene invocato quando si effettua un'handoff verticale e quindi cambiano sia gli indirizzo e porta del proxy che quelli del client. Come vediamo anche dal codice sottostante, come prima cosa crea una nuova istanza di RTPManager e aggiunge se stesso come suo listener. Poi inizializza l'RTPManager con il nuovo indirizzo di sessione associato ai nuovi indirizzo e porta del proxy passati come parameti. Infine aggiunge una destinazione del invio

dei dati multimediali associata ai indirizzo e porta del client passato come parametri.

```
rtpManager=RTPManager.newInstance();
rtpManager.addSendStreamListener(this);
SessionAddress localAddr=new SessionAddress(proxyAddress,proxyPort);
rtpManager.initialize(localAddr);

SessionAddress destAddr = new SessionAddress(clientAddress, clientPort);
rtpManager.addTarget(destAddr);
```

5.3.5 MainProxyUDPReceiver

Il componente MainProxyUDPReceiver è stato introdotto per aggiungere la funzionalità al MainProxy di poter ricevere richieste da parte del client su diverse tecnologie wireless. Il compito di questo componente è stare in ascolto su una tecnologia delle richieste dei client e al momento del loro arrivo di invocare il metodo *PacketRecived()* del MainProxy passandogli come parametro il pacchetto contenente la richiesta arrivata. Ovviamente per poter rimanere in ascolto su più tecnologie il MainProxy dovrà avere più istanze di MainProxyUDPReceiver attive, ognuna per una tecnologia.

5.3.6 UDPReceiver

Il componente UDPReceiver serve a aggiungere la funzionalità al MainProxy di poter ricevere richieste da parte del client su diverse tecnologie wireless. Il compito di questo componente è stare in ascolto su una tecnologia delle richieste dei client e al momento del loro arrivo di invocare il metodo *putDatagram()* del ProxyThread passandogli come parametro il pacchetto contenente la richiesta arrivata. Ovviamente per poter rimanere in ascolto su più tecnologie il

ProxyThread dovrà avere più istanze di UDPReceiver attive, ognuna per una tecnologia.

5.3.7 NAPDaemon

Il componente NAPDaemon è un componente di basso livello ed è composto da due metodi statici:

`public static void start(String btDeviceName)`

Questo metodo effettua vari comandi che servono ad attivare le funzionalità di un AP per quel dispositivo Bluetooth.

```
//attiva il demone DHCP che associa dinamicamente indirizzi IP ai dispositivi BT alla connessione
process = Runtime.getRuntime().exec("dhcpd -d -f pan0");

//spegne il dispositivo BT se era precedentemente acceso
//questo per essere sicuri di poter modificare la modalità
process = Runtime.getRuntime().exec("hciconfig "+btDeviceName+" down");

//cambia la modalità del dispositivo BT in MASTER (deve essere spenta)
process = Runtime.getRuntime().exec("hciconfig -a "+btDeviceName+" lm MASTER");

//accende il dispositivo BT
process = Runtime.getRuntime().exec("hciconfig "+btDeviceName+" up");

//accende il demone Napd che rende possibile fare un AP BT
process = Runtime.getRuntime().exec("napd start");
```

L'interfaccia "pan0" è l'interfaccia di rete associata al dispositivo Bluetooth rendendo possibile una comunicazione a livello di rete, cioè con indirizzi IP invece di avere una comunicazione, come normale per Bluetooth, a livello di datalink, cioè con indirizzi MAC. Questa associazione fra il dispositivo Bluetooth è l'interfaccia "pan0".

`public static void stop(String btDeviceName)`

Questo metodo serve a spegnere i processi attivati dal metodo *start()*. Viene invocato con parametro rappresentante il nome del dispositivo Bluetooth.

```
//spegne il demone Napd
process = Runtime.getRuntime().exec("napd stop");

//spegne la scheda BT
process = Runtime.getRuntime().exec("hciconfig "+btDeviceName+" down");
```

Questi metodi vengono effettuati solo se si opera sotto il sistema operativo Linux.

5.4 Test

Per verificare il funzionamento di alcuni metodi usati nell'implementazione e i tempi di esecuzione di tali metodi sono stati predisposti dei componenti di test di queste funzionalità.

Lo strumento utilizzato per lo scopo è stato:

- Pc Dell, 3GHz, 512MB RAM, Linux Gentoo dotato di un AP Wi-Fi Cisco Aironet 1100 e di un dongle Mopogo BT classe 1 versione 1.1

5.4.1 VHODatagramSocketTester

Il componente VHODatagramSocketTester ha il compito di testare la creazione delle DatagramSocket su diverse interfaccie di rete simulando un cambiamento di indirizzo del client durante l'handoff verticale.

Ai avvale di un unico costruttore:

```
public VHODatagramSocketTester(String device1, String device2)
```

Parametri:

- device1: nome identivicativo della prima interfaccia di rete (es. eth1)
- device2: nome identivicativo della seconda interfaccia di rete (es. hci0)

Durante l'esecuzione il VHODatagramSocketTester utilizza la utility DatagramSocketFactory per creare le nuove DatagramSocket sull'interfaccia di rete specifica ed utilizza il proprio metodo *printList()* per vedere lo stato di tutte le interfaccie di rete e le loro connessioni ad ogni passo. Per implementare il metodo *printList()* vengono utilizzati i metodi *getNetworkInterfaces()* e *getInetAddresses()* del componente NetworkInterface che fa parte del package standard di java *java.net*. Usando questo componente si nota come il client cambia indirizzo e porte creando nuove DatagramSocket su diverse interfaccie di rete.

Sono stati effettuati dei test sul tempo (in milisecondi) di esecuzione della creazione della socket sulle diverse scheda di rete utilizzate. Questo test presuppone che sia la scheda WiFi sia quella Bluetooth siano attive e connesse. Viene eseguita come prima cosa la creazione della socket su WiFi. Successivamente si cambiava socket creandone una sulla scheda Bluetooth e poi si eseguiva la stessa procedura permutando l'ordine delle schede.

Durante le 12 prove effettuate la creazione della prima socket richiedeva dai 2 ai 3 millisecondi mentre tutte le altre creazione richiedevano sul milisecondo.

5.4.2 ProxyBTinitTester

Il componente ProxyBTinitTester ha il compito di testare l'inizializzazione e lo spegnimento della tecnologia Bluetooth sul proxy, quindi deve verificare il funzionamento dei metodi *initBluetooth()* e *closeBluetooth()* del MainProxy. Oltre a contenere i suddetti metodi, contiene anche il metodo *printStatus()* che ha il compito di visualizzare lo stato dell'interfaccia di rete desiderata il cui noi viene passato come argomento, come vediamo dal codice sottostante.

```
process = Runtime.getRuntime().exec("ifconfig "+DeviceName);


BufferedReader processOutput=new BufferedReader( new InputStreamReader(
process.getInputStream()));
//lettura del'output del processo e scrittura su standard output
String riga = processOutput.readLine();
while(riga != null)
{
        System.out.println(riga);
        riga= processOutput.readLine();
}
```

L'esecuzione del tester si basa sull'avvio della tecnologia Bluetooth e successiva visualizzazione dello stato dell'interfaccia di rete seguita dallo spegnimento della tecnologia Bluetooth e successiva visualizzazione dell'interfaccia di rete, come si nota dal codice sottostante.

```
initBluetooth("hci0");
printStatus("pan0");
closeBluetooth("hci0");
printStatus("pan0");
```

Sono stati effettuati alcuni primi test sul tempo (in milisecondi) di esecuzione del ProxyBTinitTester, dai quali si sono ricavati i grafici riportati nelle figure seguenti:

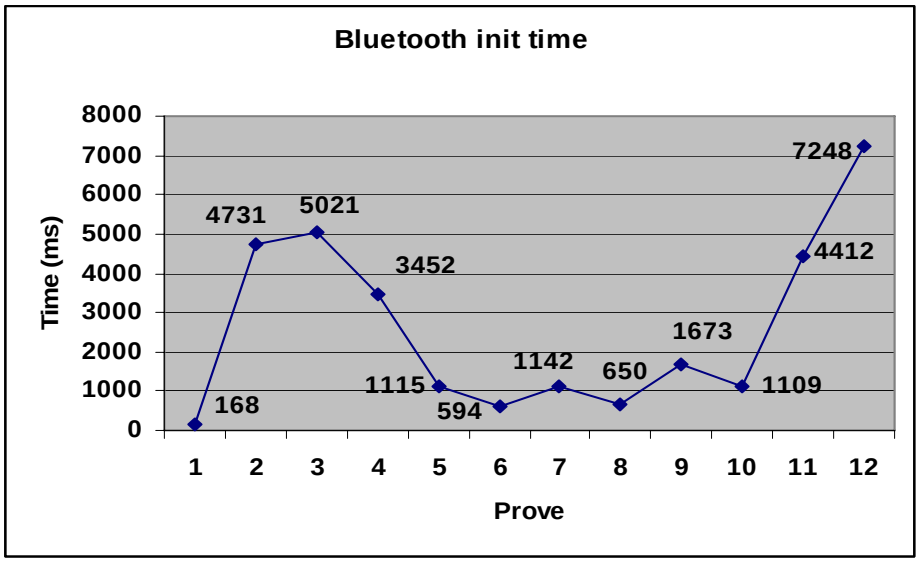

Figura 5.13: Tempo attivazione AP Bluetooth

Questa figura indica il tempo di esecuzione del metodo *initBluetooth()*, cioè l'inizializzazione del demone napd e l'attivazione della scheda Bluetooth. Vediamo che i valori cambiano nelle varie prove. Questo è anche dovuto dalla frequenza con la quale eseguiamo il test, se la frequenza è alta allora anche il valore temporale sarà più elevato, in dipendenza delle politiche di allocazione delle risorse del sistema operativo. La media in 12 prove eseguite è stata di 2.6 secondi.

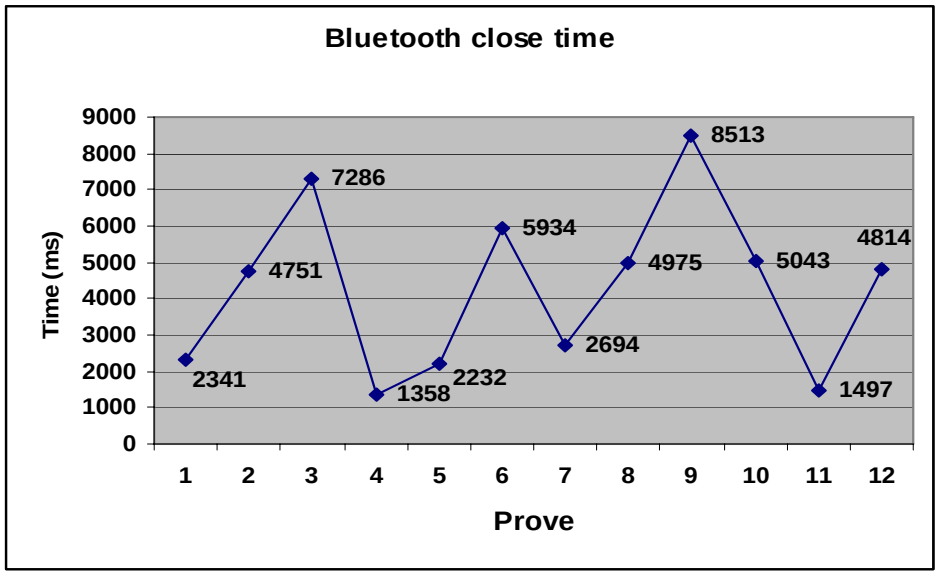

Figura 5.14: Tempo disattivazione AP Bluetooth

Questa figura indica il tempo di esecuzione del metodo *closeBluetooth()*, cioè la chiusura del demone napd e la disattivazione della scheda Bluetooth. Vediamo che i valori cambiano nelle varie prove. La media in 12 prove eseguite è stata di 4.3 secondi. Come si evince dai risultati sperimentali il tempo di chiusura è di norma superiore al tempo di inizializzazione.

Questi risultati ci danno alcune indicazioni, oltre che per i tempi d'attivazione del proxy, anche per i tempi d'attivazione del client. Ci mostrano il tempo richiesto dal proxy per mettersi in ascolto sulla tecnologia bluetooth, ma ci aspettiamo i tempi d'attivazione e disattivazione simili, e anzi migliori, sul client. Questo è dovuto al fatto che i comandi per l'attivazione e disattivazione dei dispositivi bluetooth sono uguali sia nel client che nel proxy, inoltre nel client non vi e' necessità di attivare/disattivare il napd, che è in esecuzione solamente sul proxy.

Naturalmente, questo tester può essere utilizzato solo sul sistema operativo Linux dato che tutti i metodi usati dal componente utilizzano comandi specifici si questo sistema operativo.

Conclusioni

Nel corso di questa tesi è stata di sviluppata un'infrastruttura in grado di operare su più tecnologie wireless gestendo in modo consapevole l'handoff verticale. Per fornire questa gestione si deve poter recuperare informazioni sulle condizioni delle tecnologie wireless, ad esempio WiFi e Bluetooth, disponibili su un certo client. La gestione dovrebbe portare a migliorare la qualità del servizio e a diminuire le discontinuità dovute alla mobilità del client.

È stato sviluppata sul lato client l'entità che ha il compito di essere notificata sui cambiamenti delle condizioni della tecnologia wireless correntemente usata e di operare la gestione dell'handoff verticale. Questa gestione comprende l'interrogazione di queste informazioni e la successiva verifica delle condizioni necessarie all'esecuzione dell'handoff verticale. Se queste condizioni vengono soddisfatte, viene iniziata la procedura di rebind, cioè la procedura che determina il passaggio dal funzionamento su una tecnologia wireless all'altra. Sono stati poi modificati i componenti sul lato client interessati per fornirgli la funzionalità di operare su più tecnologie, quindi consentendogli di cambiare i indirizzi e porte usati nella comunicazione.

Sul lato proxy è stato necessario aggiungere due funzionalità principali. La prima è la possibilità di ascolto delle richieste da parte del client su diverse tecnologie. Questa è stata fornita abilitando il proxy a gestire più istanze di ricevitori delle richieste che operano in modo concorrente, ognuna in ascolto su una tecnologia. La secondo è la possibilità di effettuare la propria procedura di rebind. Sono stati poi modificati tutti i componenti sul lato proxy interessati per fornirgli la funzionalità di operare su più tecnologie, quindi consentendogli di cambiare i propri indirizzi e porte usati nella comunicazione.

Per fornire il coordinamento dell'esecuzione della procedura di rebind tra client e proxy è stato creato un protocollo di interazione. Il client esegue la sua procedura

di rebind, successivamente invia la richiesta di rebind al proxy con la quale lo invita ad eseguire la propria procedura di rebind. Il protocollo proposto gestisce l'handoff verticale con una modalità di tipo hard proactive. Proactive indica che si effettuano i controlli sulla necessita di eseguire l'handoff quando la comunicazione è in corso prima della disconnessione vera e propria, attraverso l'utilizzo di un predittore. Questo ci consente, nel caso non si verifichino le condizioni, di continuare senza interruzioni la comunicazione. Nel caso si verifichino le condizioni necessarie all'esecuzione dell'handoff verticale si è deciso di disattivare la tecnologia precedentemente usata prima di attivare l'uso della nuova tecnologia, come indicato dalla modalità hard.

Ci sono diverse possibilità di sviluppi futuri. Sicuramente si può ulteriormente migliorare la qualità di servizio offerta modificando la gestione dell'handoff verticale in modo da avere una gestione di tipo Soft Proactive. Questo sarebbe possibile consentendo di avere più tecnologie simultaneamente connesse durante l'esecuzione della procedura di rebind. Così facendo non si avrebbero perdite di dati e si agevolerebbe ulteriormente la continuità di servizio. Questa soluzione introduce però un problema della complessità della gestione dell buffer. Ricevendo i dati su più tecnologie simultaneamente si crea la necessità di coordinare l'ordine di immissione dei dati nel buffer evitando copie uguali di dati. Inoltre, si introduce una maggiore complessità sul lato proxy, dovendo esso essere in grado di mandare contemporaneamente i dati su più tecnologie.
Ulteriormente, al momento le tecnologie supportate per la comunicazione sono WiFi e Bluetooth; in futuro aggiungere la possibilità di funzionamento anche su altre tecnologie wireless, ad esempio 3G.

Bibliografia

[VHM/05] Mika Ylianttila, “VERTICAL HANDOFF AND MOBILITY — SYSTEM ARCHITECTURE AND TRANSITION ANALYSIS”, ISBN 951-42-7692-2, 2005

[BPA/00] Sailesh Rathi, “Bluetooth Protocol Architecture”, Dedicated Systems Magazine, 2000

[WT/02] Mustafa Ergen, “IEEE 802.11 Tutorial”, DEECS University of California Berkeley, 2002

[CBW/02] Ken Noblitt, “A Comparison of Bluetooth and IEEE 802.11”, 2002

[SCT/03] H. Wang, A. R. Prasad, “Security Context Transfer in Vertical Handover”, PIMRC, 2003

[WSNM/04] Joshua Bardwell, “The Truth About 802.11 Signal And Noise Metrics”, D100201, Connect802 Corporation, 2004

[CLM/05] M. Cinque, F. Cornevilli, D. Cotroneo, A. Migliaccio, C. Pirro, S. Russo, “CLM and NCSOCKS: Enabling Nomadic Applications Development”, TR-WEBMINDS-53, 2005

[BVW/00] Jyrki Oraskari, “Bluetooth versus WLAN IEEE 802.11x”, 37266J, DCSE Helsinki University of Technology, 2000

[WIKI] Wikipedia
http://www.wikipedia.org

[J2se] Java Documentation online:
http://java.sun.com/j2se/1.5.0/docs/api/index.html

[IMGW/05] Miniello Giuseppe, “Infrastruttura per monitoraggio e gestione di canali wireless eterogenei”, 2005

[CSAW/03] Carlo Giannelli, “Continuità di servizio in ambiente wireless”, 2003

[PB/04] Pierangeli Diego, “Politiche di bufferizzazione in infrastrutture per l’erogazione di servizi multimediali a dispositivi wireless”, 2004

[AV/03] Paolo Angioletti, “Analisi e valutazione di un middleware per dispositivi mobili”, 2003

[HCHWN/05] Paolo Bellavista, Marcello Cinque, Domenico Cotroneo, Luca Foschini, “Integrated Support for Handoff Management and Context Awareness in Heterogeneous Wireless Networks”, 2005