Alma Mater Studiorum · Università di Bologna

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica
Sistemi ed Applicazioni Informatici

Gestione Context-Aware di Gruppi in Ambienti MANET

Tesi di Laurea in Reti di Calcolatori

Tesi di Laurea di:

Marco Marchini

Relatore:

Chiar.mo Prof. Ing. Antonio Corradi

Correlatori:

Dott. Ing. Rebecca Montanari

Dott. Ing. Dario Bottazzi

Sessione I

Anno Accademico 2004-2005

Gestione di gruppi
Applicazioni collaborative
Reti mobili ad-hoc
Tecnologie wireless
Reti di calcolatori

A mia madre,
per i sacrifici di tutti questi anni.

A mio padre,
per avermi concesso la possibilità di raggiungere questo traguardo.

Indice

Introduzione

La crescente diffusione dei dispositivi portabili che possono fruire di connettività wireless, unita all'emergenza delle tecnologie Mobile Ad hoc NETworks, aprono un nuovo scenario. Nel nuovo scenario gli utenti non necessitano solo dei tradizionali servizi Internet quali il web o la e-mail, ma richiedono anche la possibilità di beneficiare di servizi collaborativi avanzati. In particolare, i nuovi servizi devono consentire la collaborazione fra utenti vicini che condividono i medesimi interessi ed obiettivi. Esempi di servizi abilitati dalle nuove tecnologie sono costituti da applicazioni per il file sharing fra utenti mobili, dal coordinamenti di veicoli e da scenari di protezione civile.

Le caratteristiche delle MANET sollevano diversi problemi nello sviluppo di servizi collaborativi. La topologia della rete non è determinabile a priori, rendendo impossibili assunzioni sulla corrente disponibilità on-line delle diverse entità interagenti. Disconnessioni, partizioni e merge di rete sono eventi comuni che causano transitori nella collaborazione fra partner nuovi e precedentemente sconosciuti. Inoltre, un ulteriore elemento di complessità, deriva dalla richiesta degli utenti di potere collaborare tramite dispositivi eterogenei sia per natura che per prestazioni quali laptop, PDA o telefoni cellulari.

L'alto livello di dinamicità degli scenari MANET mina le assunzioni di progetto alla base delle soluzioni di supporto per servizi collaborativi attualmente disponibili. Infatti, questi sistemi tipicamente richiedono la disponibilità di server centralizzati ed assumono che la collaborazione avvenga fra utenti dotati di terminali fissi con elevate risorse computazionali connessi da canali di comunicazione affidabili e a larga banda. Le tecnologie MANET richiedono perciò di ripensare e di riprogettare il supporto a servizi di tipo collaborativo.

La tesi è organizzata come segue: il *Capitolo 1* introduce le reti wireless e le principali tecnologie di rete, il *Capitolo 2* descrive le caratteristiche delle MANET e dei protocolli di routing, il *Capitolo 3* introduce il middleware AGAPE, il *Capitolo 4* mostra la gestione delle viste in AGAPE, ed infine il *Capitolo 5* mostra il funzionamento di AGAPE in uno scenario applicativo.

Capitolo 1

Reti Wireless

I recenti sviluppi delle *tecnologie wireless* e la grande diffusione di dispositivi portatili a basso costo quali laptop, telefoni cellulari e computer palmari (PDA), hanno ampliato le possibilità degli utenti di beneficiare dei servizi di rete ovunque ed in ogni momento. Mediante questo tipo di tecnologie gli utenti possono godere di numerosi vantaggi rispetto alle soluzioni cablate, quali la maggiore flessibilità, in termini di scalabilità ed usabilità, ed una maggiore mobilità.

L'assenza di cavi per il collegamento dei dispositivi, permette agli utenti di abbattere ogni vincolo di mobilità e di beneficiare della possibilità di movimento rimanendo, con i loro dispositivi, costantemente connessi alla rete. Inoltre, grazie all'assenza di cablaggio, le tecnologie wireless risultano particolarmente adatte per l'impiego di infrastrutture di comunicazione in ambienti difficili. Il grande vantaggio di questa tecnologia è infatti la facilità d'installazione, che rende questo tipo di reti ideali per tutti quegli ambienti in cui non si intende intervenire mediante cablaggio dell'edificio o per installazioni temporanee quali fiere, congressi e conferenze.

Questo nuovo concetto di rete apre le porte a nuove problematiche e crea un'esigenza di rinnovamento a livello di infrastrutture software ed hardware per il supporto alle nuove tecnologie. Le tradizionali strutture di reti fisse, devono essere pertanto riviste alla luce delle nuove possibilità offerte dal mezzo wireless, e si assiste alla definizione di nuove reti di quali le *Body Area Network* (BAN), le *Personal Area Network* (PAN) e le *Wireless Local Area Network* (WLAN). Questa necessità di cambiamento si riflette anche nel tentativo di imporre nuovi standard tecnologici per l'utilizzo del mezzo wireless.

A partire dal 1997 infatti diverse organizzazioni e consorzi hanno promosso standard proprietari per il supporto alla nuova tecnologia. Il comitato *Institute of Electrical and Electronics Engineering* (IEEE) è risultato molto attivo in questa direzione promuovendo diversi standard tra cui la pila di protocolli IEEE 802. All'interno di quest'ultima quelli di maggior successo di mercato sono stati il protocollo 802.11, che permette la comunicazione di dispositivi a medio e corto raggio ed adottato in particolare per le reti WLAN, il protocollo 802.15 che tenta di standardizzare la tecnologia *Bluetooth* ed infine il protocollo 802.16 che fornisce accesso wireless a banda larga per utenti residenziali. La tecnologia Bluetooth infatti non è stata sviluppata direttamente da IEEE, il quale tenta di promuovere per essa un proprio standard, ma da un consorzio di cinque aziende (denominato SIG,

Special Interest Group) con l'intento di fornire un mezzo wireless per l'interconnessione di dispositivi e per la creazione di reti a cortissimo raggio (*piconet*).

Nel settore della telefonia cellulare si è assistito, e si assiste tutt'ora, all'apparizione continua di nuove tecnologie che, a partire dal *Global System Mobile* (GSM), si è giunti fino all'odierno *Universal Mobile Telecommunications System* (UMTS), passando per tecnologie ibride intermedie quali il *General Packet Radio System* (GPRS) e l'*Enhanced Data rates for GSM global Evolution* (EDGE). Tutte queste tecnologie differiscono tra loro per la banda fornita e per il *data-rate* che, nelle tecnologie di terza generazione (i cosiddetti servizi *3G*), è dell'ordine di 2Mbps.

Le tradizionali reti cablate vengono sostituite in questo nuovo ambito dalle controparti wireless, che risultano maggiormente eterogenee e si differenziano tra loro a seconda delle esigenze a cui rispondono, della copertura fornita e della banda. Inoltre, data la natura del mezzo wireless, nasce l'esigenza per queste reti, di nuovi livelli di protezione e di sicurezza per il raggiungimento e la garanzia di una comunicazione affidabile.

1.1 Tipologie di reti wireless

Una diffusa classificazione divide le reti wireless in diverse famiglie basandosi sul *range* di copertura fornita. Come indicato in Figura 1.1 infatti, le reti wireless vengono suddivise in quattro gruppi. Le reti a più corto raggio sono le BAN, che hanno come range massimo di copertura il corpo umano, seguite dalle PAN con un raggio dell'ordine della decina di metri. Le WLAN invece possiedono una range massimo di comunicazione pari a $500m$ e sono rivolte all'interconnessione di numerosi dispositivi su aree di medie dimensioni, mentre infine le *Wireless Wide Area Network* (WWAN), sono rivolte ad un tipo di comunicazione su scala geografica e necessitano necessariamente nei tratti intermedi anche di soluzioni cablate.

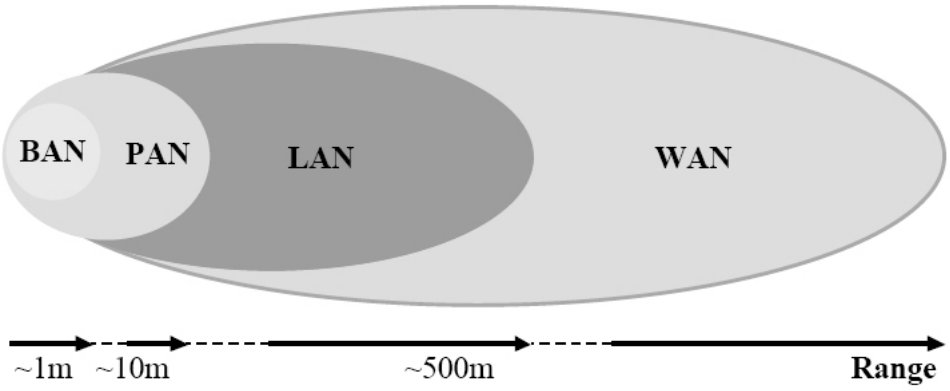

Figura 1.1: Classificazione di reti wireless.

1.1.1 Body Area Network

Le Body Area Network (BAN) nascono per rispondere all'esigenza di fornire un supporto di comunicazione necessario per dispositivi indossabili quali display, microfoni ed auricolari. Questi dispositivi infatti sono distribuiti sul corpo dell'utilizzatore e, al fine di poter interoperare, è necessaria una connettività tra di essi che viene fornita proprio dalla BAN.

Le principali caratteristiche delle BAN sono pertanto la capacità di interconnettere dispositivi eterogenei, la capacità di autoconfigurarsi a fronte della rimozione od aggiunta di dispositivi e la capacità di interconnettersi con altre BAN.

Il range di comunicazione di una BAN corrisponde pertanto al corpo umano e risulta dell'ordine di grandezza di 1/2 metri. Cablare il corpo con cavi per interconnettere i dispositivi indossabili è generalmente scomodo e la tecnologia wireless risulta la migliore soluzione per il mezzo trasmissivo di una BAN. La banda fornita in questo tipo di reti non è elevata e dell'ordine di qualche centinaio di Kbps.

Un primo prototipo di BAN fu sviluppato da T.G. Zimmerman nel 1996 [Z96]. Questo sistema prevedeva di fornire una comunicazione dati con data-rate fino a $400Kbps$ utilizzando il corpo come canale di comunicazione. In particolare Zimmerman mostrò come i dati possono essere trasferiti attraverso la pelle sfruttando una corrente molto piccola dell'ordine del miliardesimo di ampére. In via teorica il trasferimento di dati tra due persone, e quindi l'interconnessione di due BAN, poteva essere ottenuto attraverso un semplice scambio di mani.

Diversi esempi di dispositivi indossabili, e quindi di BAN, stanno facendo proprio recentemente il loro ingresso nel mercato. Questi esempi consistono di dispositivi elettronici come telefoni cellulari, player MP3, auricolari e microfoni che sono direttamente connessi senza cavi mediante le onde radio. In futuro è possibile aspettarsi che sempre più dispositivi o parti di dispositivi saranno connessi utilizzando tecnologie wireless.

1.1.2 Personal Area Network

Le Personal Area Network (PAN) rispondono all'esigenza di interconnessione di tutti i dispositivi posseduti da un utente. Si ha la necessità infatti di poter interconnettere dinamicamente più dispositivi presenti in un determinato e circoscritto ambiente. Mentre una BAN si dedica all'interconnessione dei dispositivi indossabili dall'utente, una PAN si occupa dell'interconnessione di questi con l'ambiente circostante ed è sviluppata nelle immediate vicinanze di esso.

Il range di comunicazione di una PAN è dunque leggermente superiore a quello di una BAN ed è dell'ordine della decina di metri. Il mezzo wireless che meglio soddisfa questa esigenza è senza dubbio la tecnologia radio e l'infrarosso. Tra le tecnologie del primo tipo lo standard più comunemente usato è il Bluetooth mentre tra quelle del secondo tipo è l'*Infrared Data Association* (IrDA).

La comunicazione permessa dallo standard Bluetooth avviene nell'ambito della banda di frequenze *Industrial Scientific and Medical* (ISM) a $2.4GHz$ e fornisce un data-rate di $1Mbps$. L'IrDA invece consente collegamenti bidirezionali punto-punto utilizzando gli

infrarossi. Questa tecnologia è leggermente più onerosa in fase di configurazione, in quanto richiede, a differenza dello standard Bluetooth, che i dispositivi siano in visibilità reciproca ed a una distanza di 1/2 metri. Il data-rate massimo teorico fornito da questo standard è di $4Mbps$ ma nelle più diffuse implementazioni seriali, note con il nome di *Serial InfraRed* (SIR), raggiungono un massimo di $155Kbps$.

Esempi di questo tipo di reti sono le reti domestiche costituite da una insieme di dispositivi elettronici, quali computer, stampante e monitor che eventualmente possono connettersi, grazie alla PAN, con dispositivi elettronici mobili quali laptop, PDA e telefono cellulare. Grazie a questa interconnessione è ad esempio possibile condividere l'accesso Internet tra il computer di casa ed il laptop, sincronizzare la rubrica di un telefono cellulare e condividere file multimediali. Se inoltre, viene utilizzato come standard di comunicazione la tecnologia Bluetooth, tutto questo può avvenire nella massima libertà di movimento.

La Figura 1.2 mostra la relazione esistente tra una rete BAN ed una PAN.

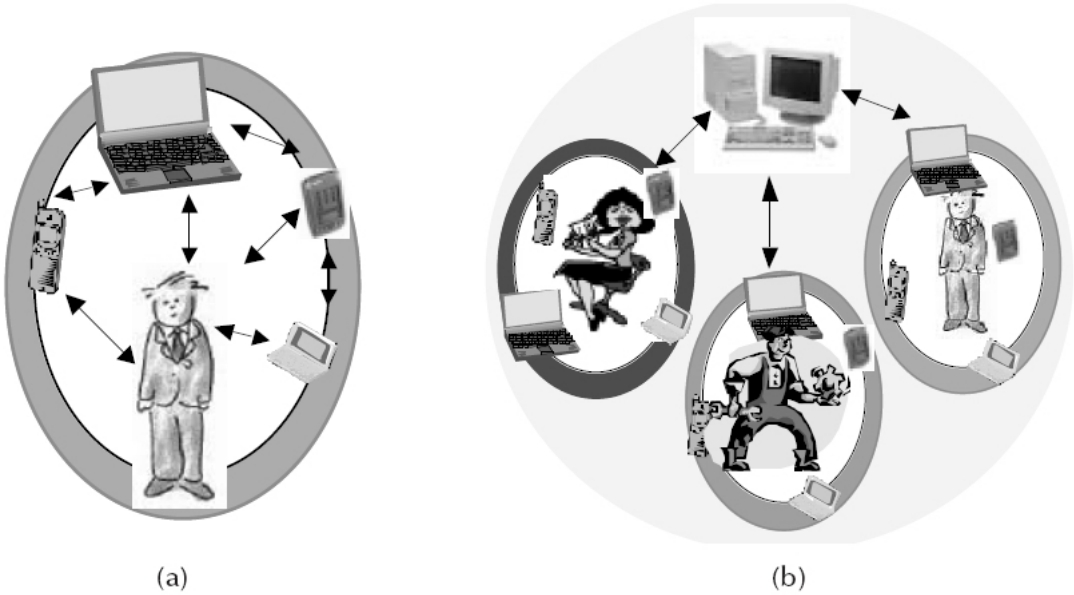

Figura 1.2: Relazione tra una rete BAN (a) ed una PAN (b).

1.1.3 Wireless Local Area Network

Le Wireless Local Area network (WLAN) nascono per rispondere all'esigenza di interconnettere più dispositivi all'interno dello stesso edificio o area residenziale, evitando di intervenire strutturalmente mediante il cablaggio degli edifici stessi. Le esigenze che portano alla definizione di questo nuovo tipo di reti sono le stesse delle LAN cablate, con l'ulteriore vantaggio di non essere più vincolati dai cavi di collegamento. Visti i numerosi vantaggi introdotti dalle WLAN è facile prevedere che diventeranno la soluzione ideale per le reti locali portando al definitivo abbandono di Ethernet.

Come le LAN cablate anche le WLAN hanno un range di comunicazione tipicamente di un singolo edificio o di un gruppo di edifici, e generalmente dell'ordine delle centinaia di metri. La standard di comunicazione maggiormente adottato per l'interconnessione dei dispositivi in questo tipo di reti è il protocollo IEEE 802.11. Questo protocollo, descritto in seguito nel paragrafo 1.3.1, permette una comunicazione a onde radio mediante la banda di frequenze ISM a $2.4GHz$ e con un data-rate variabile a seconda della versione, che va da un minimo di $1Mbps$ fino ad un massimo di $54Mbps$ della versione g. Questo standard inoltre permette la configurazione secondo due distinte implementazioni: un'approccio basato su *infrastruttura* ed un'approccio cosiddetto *ad hoc*.

Un'architettura basata su infrastruttura impone l'esistenza di un controllore centralizzato per ogni cella, spesso chiamato *Access Point*, che normalmente è connesso alla rete cablata e fornisce l'accesso internet ai dispositivi mobili. Un'architettura ad hoc invece è una rete *peer-to-peer* formata da stazioni posizionate entro il range di ogni altra che si autoconfigurano dinamicamente per formare reti temporanee. In queste reti non esistono controllori fissi ma ne viene eletto dinamicamente uno tra le stazioni partecipanti alla comunicazione.

A differenza delle LAN cablate, le WLAN dovrebbero essere progettate per fronteggiare problemi specifici dell'ambiente wireless, come la sicurezza, il consumo di potenza, la mobilità e la limitata ampiezza di banda.

1.1.4 Wireless Wide Area Network

Le Wireless Wide Area Network (WWAN) nascono dall'esigenza di interconnettere dispositivi wireless e dunque utenti su scala geografica. Queste reti sono quelle a più ampio raggio possibile e vengono utilizzate oggi sia per la telecomunicazione cellulare sia per il trasferimento dati, che grazie alle odierne tecnologie, ha raggiunto alti livelli di data-rate. Le WWAN vengono inoltre impiegate per l'interconnessione di LAN dislocate su territori diversi.

Le WWAN sono estese su vaste aree geografiche e raggiungono persino, nel caso della rete di telefonia cellulare, dimensioni di una nazione. Il raggio di trasmissione tipico è dunque dell'ordine delle decine e centinaia di chilometri, mentre la banda trasmissiva dipende strettamente dalla tecnologia di comunicazione impiegata.

Esempi di questo tipo di reti sono dunque la rete di telefonia cellulare nazionale, che rappresenta il futuro della comunicazione voce e dati di ogni giorno. Si assiste infatti ad una crescente esigenza di accedere in qualunque momento a file e dati ed al bisogno di scambiarli con altri utenti indipendentemente dalla locazione fisica in cui ci si trova. Le WWAN forniscono questo tipo di supporto e realizzano un accesso alle informazioni di tipo *anytime* ed *anywhere*.

La rete di telefonia cellulare vede, da alcuni anni, un susseguirsi continuo di standard che incrementano sempre più il data-rate di trasferimento ed il numero di servizi offerti. Negli Stati Uniti lo standard *Advanced Mobile Phone Systems* (AMPS) permette la trasmissione di voce tra cellulari Tacs. Questo standard rappresenta la prima generazione di telefonia cellulare ($1G$) ed è basato su tecnologie analogiche operanti ad una frequenza di $800MHz$.

Gli standard di seconda generazione ($2G$) sono basati su tecnologia digitale e sono stati

progettati principalmente per servizi telefonici. Anche se il data-rate è più elevato rispetto agli standard precedenti, non è possibile ancora fruire di servizi multimediali con adeguate performance. Tra gli esempi di seconda generazione quello di più successo è senza dubbio il Global System Mobile (GSM) per la comunicazione voce a livello europeo. Il data-rate per questa tecnologia non supera i $9.6Kbps$ ed opera a frequenze di $900MHz$ e $1800MHz$.

Gli standard successivi a quelli di seconda generazione rappresentano una generazione intermedia, i cosiddetti $2.5G$, in quanto non introducono sostanziali modifiche ai protocolli $2G$ se non un'aumento della banda trasmissiva al fine di usufruire anche una comunicazione di tipo dati. Esempi di questa generazione sono il General Packet Radio System (GPRS) e l'Enhanced Data rates for GSM global Evolution (EDGE), che aumentano il data-rate fino a $348Kbps$.

L'ultima generazione di standard in ordine cronologico è la terza ($3G$). Questi protocolli si propongono di aumentare notevolmente la banda offerta raggiungendo data-rate intorno ai $2Mbps$ fornendo dunque il giusto supporto per la trasmissione e lo scambio di dati. Grazie all'aumentata banda disponibile, nascono anche servizi innovativi quali le videochiamate e lo scambio di brevi filmati. Uno standard di questa famiglia è lo Universal Mobile Telecommunications system (UMTS) introdotto dalla International Telecommunications Union (ITU).

Mentre la maggior parte dei sistemi di comunicazione cellulare utilizza tecnologie switched, UMTS prevede anche il supporto per la comunicazione basate su pacchetti, più adatta alla gestione del traffico dati. Le frequenze utilizzate da UMTS sono comprese tra i $1900MHZ$ ed i $2200MHz$ ed il data-rate minimo disponibile fornito è di $144Kbps$ per le applicazioni a mobilità totale e di $384Kbps$ per applicazioni a mobilità parziale. Il data-rate massimo teorico di $2Mbps$ è raggiunto dalle applicazioni a bassa mobilità.

1.2 Modelli di comunicazione per reti wireless

Tipicamente è possibile classificare il modello di comunicazione wireless in base all'appoggio eseguito sulla rete cablata. Seguendo questa linea è possibile distinguere un *modello a cella*, in cui si utilizzano connessione cablate per l'interconnessione delle varie celle, un *modello a cella virtuale*, in cui le celle sono connesse con collegamenti wireless, ed infine un *modello ad hoc* in cui tutti i nodi sono pari e connessi con collegamenti wireless. Ogni modello rappresenta un rilassamento del modello precedente.

1.2.1 Modello a cella

In questo modello l'area di copertura è suddivisa in un certo numero di celle le quali possono anche sovrapporsi le une con le altre. Ciascuna cella è formata da una stazione base fissa, indicata in Figura 1.3 da un pallino grigio, chiamata appunto *Base Station* (BS), e da uno o più host mobili, indicati in figura con pallini bianchi e chiamati *Mobile Hosts* (MH). Le BS hanno il compito fondamentale di permettere la comunicazione tra i nodi della cella e tra i nodi appartenenti a celle diverse. Gli MH invece rappresentano le entità comunicanti e possono muoversi liberamente all'interno della stessa cella e tra celle diverse.

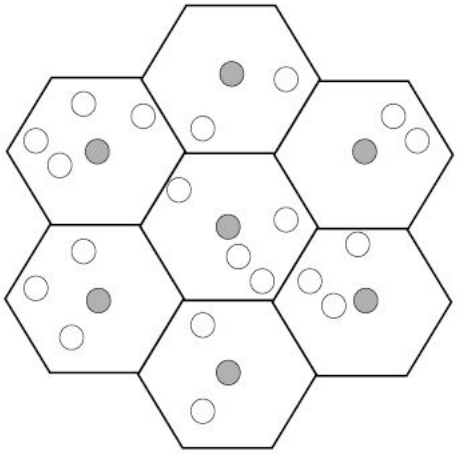

Figura 1.3: Modello a cella.

Gli MH sfruttano l'infrastruttura di comunicazione fornita dalle BS per comunicare, pertanto tale comunicazione è possibile finchè gli MH comunicanti rimangono all'interno del range di copertura delle BS presenti. Non appena un MH si sposta al di fuori la comunicazione cade.

In questo modello un MH comunica sempre con il BS della cella a cui appartiene in quell'istante e necessita del proprio BS per poter comunicare con qualsiasi altro host. Le comunicazioni tra un MH ed il suo BS avvengono mediante mezzo wireless mentre i BS delle diverse celle sono connessi tra di loro mediante collegamenti cablati. Per poter comunicare con un'altro host un MH deve segnalare al proprio BS l'intenzione di comunicare ed il destinatario del messaggio. A questo punto il BS controlla se il destinatario della comunicazione è anch'esso presente nella sua cella e si comporta in due modi: se l'MH si trova nella stessa cella del mittente della comunicazione il BS mette in contatto direttamente i due MH mediante comunicazione wireless; se l'MH invece si trova in un'altra cella, il BS del richiedente deve cercare la cella in cui risiede il destinatario della comunicazione, quindi, una volta individuata, stabilisce un collegamento cablato con la BS di tale cella ed infine permette la comunicazione tra i due MH.

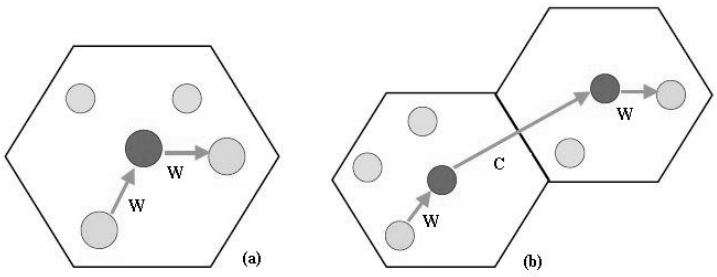

Figura 1.4: (a)Comunicazione intra-cella. (b)Comunicazione inter-cella.

1.2.2 Modello a cella virtuale

Rilassando il modello a cella al fine di permettere alle BS di muoversi si ottiene il modello a cella virtuale. Pertanto l'unica differenza dal modello precedente sta nella mobilità delle BS (che diventano ora *Mobile Base Station*, MBS), che risultano ora connesse tra loro tramite collegamenti wireless al fine di permettere il movimento.

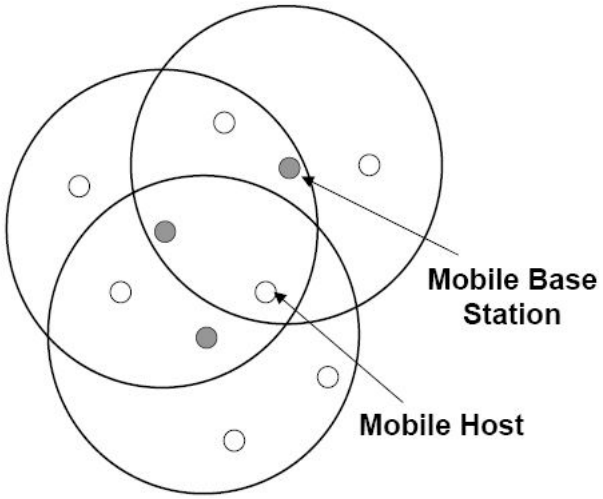

Figura 1.5: Modello a cella virtuale.

Analogamente a quanto avveniva nel modello a cella le comunicazioni tra due MH avvengono ancora attraverso i BS delle celle e sono sempre loro che coordinano e concedono le comunicazioni agli MH richiedenti. A differenza del precedente modello però il grafo dei BS evolve nel tempo e, per poter rendere effettiva la comunicazione tra diversi BS, i BS devono vedersi reciprocamente tra di loro cioè i BS per poter comunicare e permettere la comunicazione devono essere reciprocamente nel range di un'altro BS. Quando un BS vuole comunicare con un'altro BS che non rientra nel suo range di comunicazione deve attraversare altri BS direttamente o indirettamente raggiungibili fino ad arrivare a quello cercato. Per poter giungere al BS voluto deve però esistere una cammino tra il BS sorgente e quello destinazione composto da un numero qualsiasi di BS intermedi. Affinchè questa ipotesi sia sempre valida, in questo modello si assume comunemente che, per ogni coppia di BS, esiste sempre un cammino di routing composto di sole BS che le interconnette.

L'intera comunicazione tra due MH pertanto avviene tra soli BS tranne per i tratti iniziale e finale in cui avviene tra MH e BS. L'MH sorgente infatti, quando intende iniziare una comunicazione con un'altro MH, si rivolge ancora una volta al proprio BS il quale si comporta come nel modello a cella: se il destinatario della comunicazione è nella stessa cella esegue direttamente la comunicazione tra i due MH mediante collegamento wireless altrimenti inoltra la richiesta al BS della cella destinazione se presente nel suo range di comunicazione o, se non lo fosse, inoltra la richiesta al primo BS del cammino necessario per raggiungere il BS destinazione.

1.2.3 Modello ad hoc

Rilassando ulteriormente il modello precedente e facendo sfumare le differenze tra BS ed MH si ottiene il modello ad hoc.

Ora scompaiono i ruoli tra le entità del modello e tutti gli host diventano pari svolgendo le stesse funzioni. In questo modello di comunicazione tutte le entità sono mobili come nel caso precedente, ma dato che nessun host in particolare svolge più la funzione di BS, gli MH possono o comunicare direttamente con host presenti nel proprio range o indirettamente attraverso un cammino che può contenere qualsiasi nodo della rete. Ora infatti la richiesta di comunicazione può essere inoltrata a qualsiasi host della rete in quanto ogni nodo implementa le funzioni del BS. Come mostrato in Figura 1.6a infatti, tutti i nodi sono pari con un proprio range d'azione e, come si vede dalla Figura 1.6b, quando il nodo A vuole raggiungere il nodo B fuori dal suo range, sfrutta il nodo intermedio C che lo vede direttamente e può quindi istanziare la comunicazione.

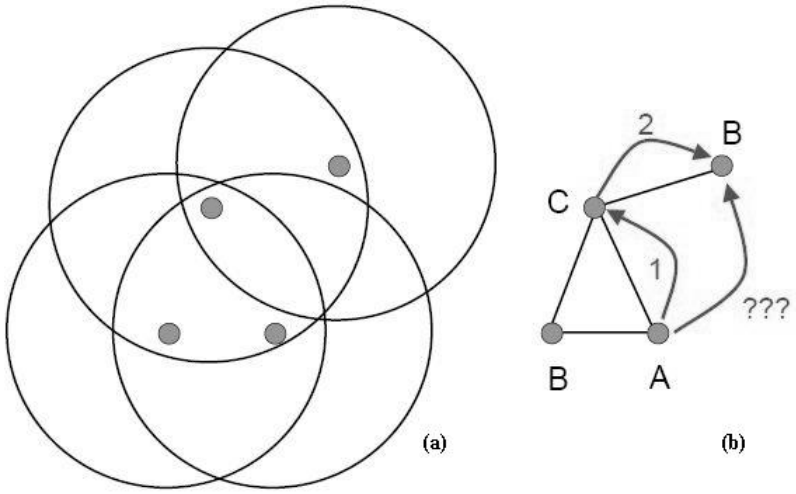

Figura 1.6: (a)Modello ad hoc. (b)Esempio di comunicazione.

In pratica ogni nodo è in grado di gestire la propria comunicazione, senza l'esigenza di appoggiarsi ad una stazione fissa ma cooperando con gli altri pari della rete. Lo svolgimento di questa funzione innalza una serie di problematiche che riguardano la cooperazione tra i vari nodi nel prendere le decisioni e porta al necessario sviluppo di nuovi protocolli di cooperazione e di comunicazione. Questo tipo di reti sono particolarmente indicate in situazioni ambientali prive di infrastrutture fisse dove la comunicazione dei nodi deve essere gestita in modo autonomo dagli host stessi. Esempi pratici di queste situazioni sono i casi di *disaster recovery* ed *emergency rescue* come le operazioni di soccorso in casi di terremoto e di disastri ambientali.

1.3 Tecnologie wireless

Come descritto in precedenza, esistono numerose e diverse tecnologie che impiegano il mezzo wireless come tecnica trasmissiva. Le più importanti dal punto di vista del mercato, oltre alle tecnologie di telefonia mobile viste nella sezione 1.1.4 adottate per le WWAN, sono senza dubbio gli standard adottati per le reti a medio e corto raggio: lo standard Bluetooth per le reti BAN e PAN e lo standard IEEE 802.11 per le reti WLAN.

1.3.1 Standard IEEE 802.11

Questo protocollo è stato il primo standard WLAN a guadagnare consensi di mercato. Originariamente fu concepito nel 1987 dall'IEEE come parte di uno standard esistente chiamato IEEE 802.4 riguardante il *token bus* e da cui prese il nome di 802.4L. Nel 1990 quest'ultimo fu separato dal progetto 802.4 e fu rinominato *IEEE 802.11 WLAN Project Committee*, il quale creò uno standard 802 indipendente con il compito di definire le specifiche di tre strati fisici e di uno strato *Medium Access Control* (MAC). Quest'ultimo si pone nel modello di riferimento OSI come un sottostrato del livello Data-link, e permette di avere un'interfaccia uniforme sullo strato fisico quando il canale di comunicazione è di tipo multiaccesso. Al di sopra dello strato MAC vi è il sottostrato *Logical Link Control* (LLC) che è il vero protocollo di strato Data-link con controllo di errore e di flusso e sul quale si appoggiano Ethernet e tutti i protocolli della famiglia 802. Un compito fondamentale di LLC è nascondere le differenze tra i vari tipi di reti 802 offrendo un'interfaccia unica verso lo strato Network.

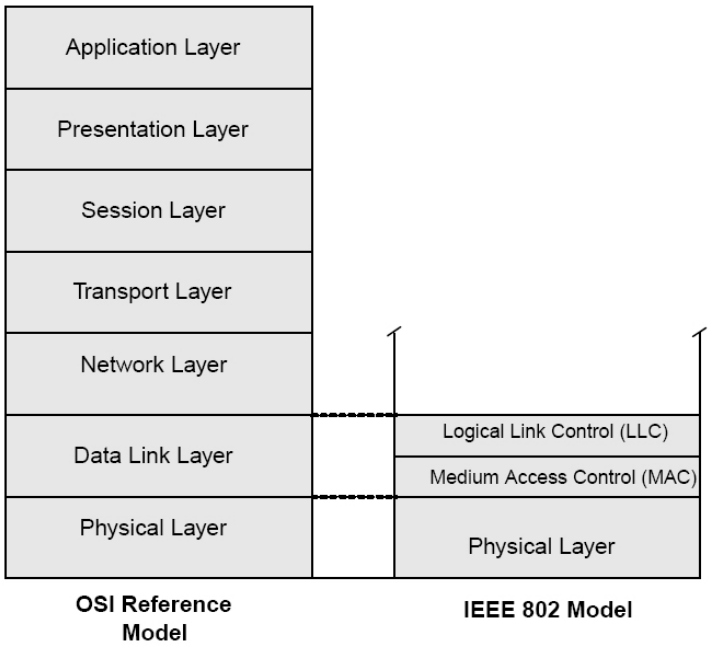

Figura 1.7: Modello di riferimento OSI ed IEEE 802.

Come mostra la Figura 1.7 il modello IEEE 802 tratta solamente gli strati inferiori Physical e Data-link, mentre non specifica nulla al di sopra di essi a differenza del modello ben strutturato di OSI.

Lo scopo di 802.11 era favorire la compatibilità tra i prodotti industriali WLAN di diversi produttori, e proprio questa raggiunta compatibilità, portò all'approvazione dello standard il 27 Giugno 1997. In questa prima versione furono adottate tre tecniche di trasmissione supportate nello strato fisico: un metodo a infrarossi e due metodi che utilizzano un sistema radio a bassa potenza basato su tecniche di modulazione *Frequency Hopping Spread Spectrum* (FHSS) e *Direct Sequence Spread Spectrum* (DSSS) operanti nello spettro della banda ISM a $2.4GHz$ e con data-rate rispettivamente fino a 1 e $2Mbps$. Da allora sono stati istituiti due standard IEEE aggiuntivi per estendere la banda di trasmissione delle WLAN, in particolare agendo sulle specifiche dello strato fisico. Questi due protocolli vennero approvati nel 1999 e chiamati IEEE 802.11a e IEEE 802.11b. Entrambe le specifiche condividono lo stesso sottostrato MAC dell'802.11 standard, mentre le differenze sono evidenti nelle specifiche del nuovo strato fisico: il primo utilizza un multiplexing a divisione di frequenza ortogonale (OFDM) nella banda UNI a $5GHz$ ottenendo una velocità di trasferimento fino a $54Mbps$, il secondo invece utilizza una modulazione *High Rate Direct Sequence Spread Spectrum* (HR-DSSS) nella banda ISM a $2.4GHz$ ottenendo una velocità di trasferimento fino a $11Mbps$. Una versione evoluta di quest'ultimo è chiamata 802.11g ed è stata approvata da IEEE nel Novembre 2001. Questo standard utilizza il metodo di modulazione OFDM di 802.11a ma opera nella banda ristretta ISM a $2.4GHz$ come per 802.11b, consentendo un data-rate teorico fino a $54Mbps$.

1.3.1.1 Sottostrato MAC di 802.11

Il livello MAC fornisce un meccanismo di accesso base che supporta diverse funzionalità come l'assegnamento unico del canale, configurazione del collegamento, autenticazione, sincronizzazione del canale ed il *roaming*. Quest'ultimo è una caratteristica unica delle WLAN 802.11 e permette agli utenti mobili di muoversi tra le aree di copertura di più punti di accesso chiamati *Basic Service Areas* (BSA). Tra tutte queste caratteristiche di MAC, quella di maggiore importanza che richiede uno sguardo più approfondito è sicuramente l'assegnamento del canale.

Con Ethernet una stazione che voleva trasmettere si limitava ad aspettare che il mezzo condiviso di trasmissione diventasse silenzioso e, una volta tale, iniziava a trasmettere. Se vi era una collisione tra due stazioni che trasmettevano contemporaneamente la situazione era recuperata mediante il protocollo *Carrier Sensing Multiple Access/Collision Detect* (CSMA/CD) che è in grado di rilevare se sul mezzo stanno trasmettendo due host e risolvere il problema. Tutto questo con il wireless non regge più. In questo ambiente non esiste più un canale fisico su cui effettuare il *sensing* e quindi non è più possibile rilevare le eventuali collisioni e reagire opportunamente. C'è inoltre il cosiddetto problema della *stazione nascosta*, una stazione fuori dal campo di un'altra che sta trasmettendo crede erroneamente che il canale è libero, e quello della *stazione esposta*, una stazione vuole trasmettere dati ad un'altra ma sente il canale occupato da una terza stazione presente nel suo range. A causa di tutti questi problemi 802.11 non utilizza CSMA/CD e adotta

due modalità operative. La prima è chiamata *Distributed Coordination Function* (DCF) che, similmente a Ethernet, non utilizza nessun tipo di controllo centrale; l'altra, chiamata *Point Coordination Function* (PCF), usa la stazione base per controllare tutta l'attività nella cella. Tutte le implementazioni devono supportare obbligatoriamente DCF mentre è opzionale PCF.

Quando adotta DCF, 802.11 utilizza un protocollo chiamato *Collision Avoidance* (CSMA/CA). In questo caso non è possibile rilevare le collisioni in aria e per questo si previene da tali situazioni di errore riducendo il numero delle possibili occorrenze di collisione. Una stazione che vuole trasmettere utilizza CSMA per ascoltare o sentire se è presente un'altra stazione trasmittente prima di tentare di inviare i propri pacchetti, dopodichè, se è stata rilevata un'altra trasmissione il terminale ritarda l'invio fino alla liberazione del mezzo, altrimenti invia immediatamente. Lo schema CA inoltre fornisce un ritardo alla ritrasmissione con *back-off* casuale che aiuta ad evitare collisioni tra più trasmettitori simultanei.

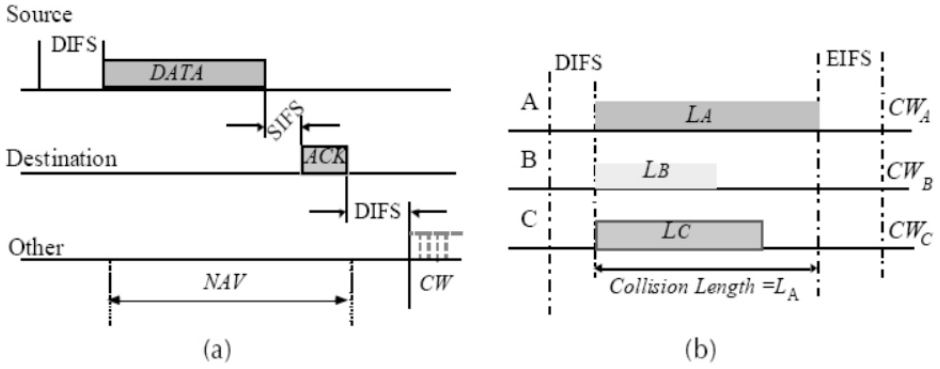

Figura 1.8: Protocollo CSMA/CA: (a)Trasmissione con successo; (b)Collisione.

In Figura 1.8 viene mostrato il funzionamento del protocollo CSMA/CA nei due casi con e senza collisione. Prima dell'inizio di una trasmissione si verifica, mediante *sensing* del canale, che nessun'altra stazione stia trasmettendo in quell'istante e, se il canale risulta libero per un determinato intervallo temporale chiamato *Distributed InterFrame Space* (DIFS), la stazione si impossessa di esso e prosegue con la trasmissione vera e propria. Nel pacchetto inviato dal mittente c'è un campo che porta l'indicazione della lunghezza totale dei dati che vuole inviare in modo tale che le altre stazioni possano memorizzare tale informazione e conoscere quindi il periodo di tempo per cui il canale sarà occupato. Quest'informazione è chiamata *Network Allocation Vector* (NAV). Il NAV, come si vede da Figura 1.8a, comprende anche un intervallo di tempo chiamato *Short InterFrame Spacing* (SIFS) e un messaggio di controllo di *acknowledgement* (ACK) che conferma la ricezione del pacchetto. Il ricevente infatti prima di inviare la conferma della ricezione attende un periodo di tempo SIFS, minore di DIFS, che serve per concordare i turni tra le parti coinvolte nella conversazione. Gli ACK non vengono trasmessi se si verificano delle collisioni o il pacchetto è andato perduto o corrotto.

Per diminuire la probabilità di collisione, 802.11 adotta inoltre un meccanismo che

garantisce la diffusione dell'istante di trasmissione. Una stazione che intende inviare un pacchetto e che trova il canale occupato, rimanda tale operazione fino a che il canale diventa libero e cioè attende per un periodo di tempo pari al NAV. Al termine di tale intervallo il canale è stato rilasciato e la stazione che tenta di trasmettere esegue sensing del canale per un periodo pari a DIFS. Al termine di questo periodo la stazione inizia un conteggio, chiamato *backoff timer*, e sceglie l'istante di inizio trasmissione selezionando un intervallo casuale (*backoff interval*). L'algoritmo di *backoff esponenziale binario* opera nel seguente modo: se una stazione sente il canale occupato rinvia la propria trasmissione al termine di quella in corso; riesegue il sensing del canale e inizializza un contatore scegliendo il backoff interval in modo casuale tra una serie di *slot time* chiamata *Contention Window* (CW) che dipende dal numero delle trasmissioni fallite; quando il canale è inattivo tale contatore viene decrementato mentre in presenza di trasmissioni viene stoppato: al termine di tale intervallo la stazione ritenta di accedere al canale e se quest'ultimo risulta libero per un periodo di tempo superiore al DIFS inizia a trasmettere. Nel caso contrario la stazione sceglierà un nuovo intervallo di backoff ed eseguirà nuovamente i passi precedenti. Il nuovo intervallo scelto sarà doppio rispetto il precedente, per cui si parte con un'intervallo pari a CW_{min} al primo tentativo fino al valore massimo CW_{max} dopo un determinato numero di collisioni.

Nel caso in cui due o più stazioni inizino a trasmettere nello stesso istante si possono verificare delle collisioni sul canale. Come indicato in Figura 1.8b, le stazioni rinviano il tentativo di trasmissione al termine dell'intervallo di collisione dove vengono riattivate dopo un'intervallo di tempo chiamato *Extended InterFrame Specing* (EIFS) se il canale è risultato libero in tale periodo. Al termine di tale periodo inizia il protocollo a contesa appena visto.

1.3.1.2 RTS/CTS

Un problema che si può verificare in una rete wireless è quello della cosiddetta stazione nascosta (*hidden station*).Questa situazione si può verificare quando una stazione è in grado di ricevere pacchetti provenienti da due stazioni distinte, ma queste ultime non possono ricevere i loro segnali reciproci perchè fuori della portata d'azione dei reciproci dispositivi. In questo caso il rischio di collisione diventa molto elevato dato che una delle due stazioni può trasmettere anche se lo sta già facendo l'altra, in quanto non può rilevare mediante sensing del canale la trasmissione in corso.

Per risolvere, o comunque ridurre il problema, è stato aggiunto un meccanismo conosciuto con il nome *Request to send/Clear to send* (RTS/CTS). Una stazione che intende trasmette un pacchetto (A), aspetta che il canale sia libero per un periodo DIFS come avviene nell'accesso base, e nel caso lo sia, invia un frame di piccole dimensioni chiamato RTS per indicare la propria intenzione a trasmettere a quel determinato destinatario. Quando la stazione destinataria (B) riceve l'RTS, risponde dopo un intervallo di tempo SIFS, con un frame chiamato CTS per dare il via libera all'inizio della trasmissione. Non appena la stazione mittente riceve il CTS, avvia la trasmissione dei pacchetti. Le trame RTS e CTS includono un campo *duration*, che specifica l'intervallo di tempo necessario per trasmettere completamente il frame dati e la relativa conferma di ricezione. Questa

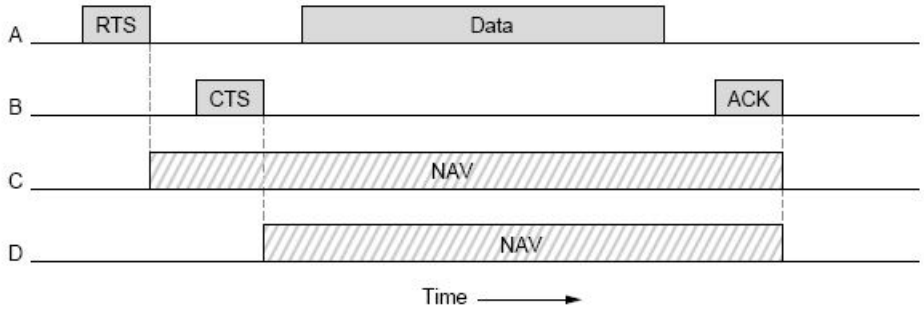

Figura 1.9: Controllo del canale virtuale mediante CSMA/CA.

informazione è utilizzata da tutti i nodi della rete (C, vicina al mittente, e D vicina al destinatario) che sono in grado di rilevare la stazione trasmittente o ricevente al fine di aggiornare il proprio NAV.

1.3.1.3 Sottostrato LLC: IEEE 802.2

Questo protocollo è il vero e proprio protocollo di strato Data-link con controllo di flusso e di errore. Esso si posiziona sopra Ethernet e sopra gli altri protocolli 802 ed IEEE lo ha standardizzato con l'acronimo di 802.2. Un'altra importante funzione di questo sottostrato è quella di nascondere le differenze tra i vari tipi di reti 802 offrendo un'interfaccia e un formato unici verso lo strato Network. Il formato, l'interfaccia e il protocollo si basano strettamente sul protocollo *High-level Data Link Control* (HDLC).

LLC fornisce tre modalità di servizio: *datagram inaffidabile*, *datagram con acknowledge* e *servizio affidabile orientato alla connessione*. L'intestazione di LLC che viene aggiunta al pacchetto passato dallo strato di rete contiene tre campi: l'AP di destinazione, quello sorgente che ha generato il frame ed un campo di controllo che contiene i numeri di sequenza e di acknowledge. Questi campi sono generalmente utilizzati quando viene richiesta una connessione affidabile sul livello Data-link.

1.3.1.4 Modalità operative

Lo standard IEEE 802.11 permette la configurazione di reti WLAN secondo due modelli di architetture: un modello basato su *infrastruttura fissa* di comunicazione (Figura 1.10a) ed uno basato sul *modello ad hoc*, cioè privo di tale infrastruttura (Figura 1.10b).

Nel primo caso si ha un *Basic Service Set* (BSS), ovvero una rete wireless composta da almeno un AP e da uno o più nodi mobili. Tutte le comunicazioni provenienti da tali nodi e dirette ad essi passano attraverso l'AP che gestisce quella cella e si occupa quindi della loro coordinazione. L'AP della cella è l'unica stazione collegata, mediante cavo, alla rete esterna alla LAN e fornisce quindi a tutti i nodi connessi il collegamento ad Internet e quindi la possibilità di uscire dalla rete locale.

Nel secondo caso invece 802.11 definisce una rete wireless ad hoc che viene chiamata *Independent Basic Service Set* (IBSS). Un IBSS rende possibile la comunicazione diretta tra

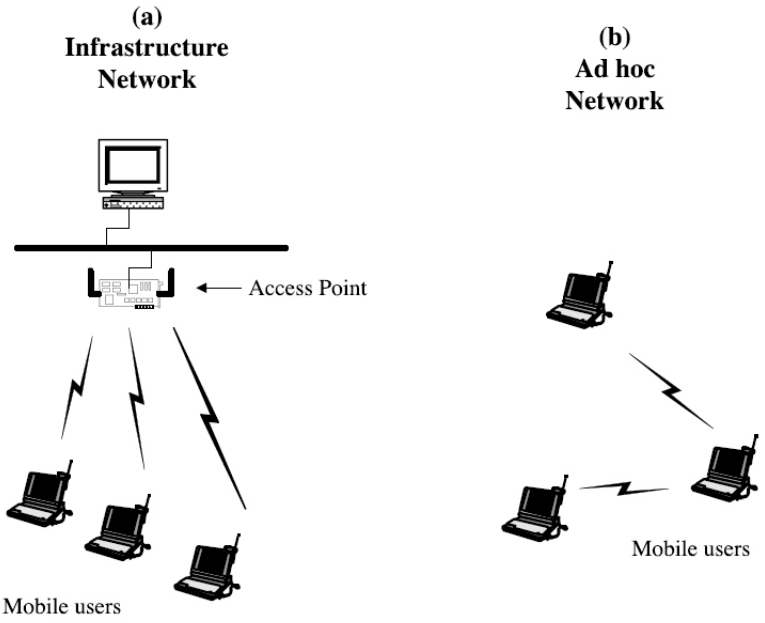

Figura 1.10: (a)Rete wireless infrastructure-based; (b)Rete wireless ad hoc.

due nodi mobili in visibilità tra loro, senza l'appoggio ad un'infrastruttura che permetta tale comunicazione: grazie alla flessibilità di CSMA/CA, la sincronizzazione è infatti sufficiente a garantire una corretta ricezione dei dati ed un buon livello di coordinamento tra i pari comunicanti. 802.11 per raggiungere questa sincronizzazione, si serve principalmente di due funzioni:

- **Acquisizione della sincronizzazione**: Questo metodo serve ad un nodo quando vuole entrare a far parte di una IBSS già esistente e precedentemente rilevata. Il nodo che intende accedere ad una IBSS scansiona il mezzo wireless alla ricerca di particolari frame di controllo su diverse frequenze radio: se il nodo rileva una IBSS allora richiede la sincronizzazione altrimenti può procedere alla creazione di una nuova IBSS.

- **Mantenimento della sincronizzazione**: Data la mancanza di un nodo centrale che fornisca un livello di coordinamento mediante un clock comune a tutti i nodi, nelle IBSS si utilizza un algoritmo distribuito di mantenimento della sincronizzazione. Quest'algoritmo viene eseguito da tutti i nodi della rete e si basa sull'invio di piccoli frame chiamati *beacon frames* ad una frequenza nominale, decisa dal nodo creatore dell'IBSS, con lo scopo di fornire una sorta di segnale di clock per il sincronismo.

La possibilità di permettere la configurazione di entrambe le modalità operative è stato uno dei motivi che ha decretato il successo di mercato dell'IEEE 802.11.

1.3.2 Bluetooth

La tecnologia Bluetooth fornisce una nuova prospettiva nei collegamenti a corto e cortissimo raggio. Sebbene non fornisca molte delle funzioni offerte dalle specifiche di IEEE 802.11, Bluetooth fornisce una funzionalità di *service discovery* che permette la creazione dinamica di reti ad hoc e soprattutto rappresenta una soluzione *cable replacement* per interconnettere i dispositivi degli utenti. La tecnologia Bluetooth offre la libertà di interconnettere senza fili un computer ad un telefono cellulare, un computer ad un PDA, oppure un PDA ad un telefono cellulare e tra molti altri tipi di dispositivi elettronici. Quando viene usato per interconnettere un computer con un telefono cellulare, quest'ultimo può effettivamente operare come *gateway* per inviare e ricevere informazioni su una LAN o WAN o perfino connettere il dispositivo ad Internet. Da tutti questi esempi si evince come la tecnologia Bluetooth porti un nuovo livello di interconnettività alla dimensione personale.

Bluetooth possiede inoltre diverse caratteristiche chiave che permettono il suo impiego anche nella sfera delle reti PAN, in particolare in sostituzione dell'ormai obsoleto IrDA. Quest'ultimo richiedeva infatti all'utente una fastidiosa fase di configurazione e vincolava i dispositivi ad essere in visibilità reciproca per il trasferimento dei dati, cosa che in caso di disallineamento, portava a problemi di connettività e a frustranti ritrasmissioni. Bluetooth supera questi problemi offrendo un'interfaccia in aria universale, a basso costo, user-friendly ed è in grado di sostituire la grande varietà di cavi proprietari che gli utenti devono possedere per poter interconnettere dispositivi eterogenei.

Nel 1994 Ericsson iniziò ad interessarsi alla connessione senza cavo dei suoi telefoni cellulari con altri dispositivi. Insieme ad altre quattro aziende leader del settore (IBM, Nokia, Intel e Toshiba), Ericsson formò uno *Special Interest Group* (SIG) con l'intenzione di sviluppare uno standard wireless che avrebbe dovuto permettere il collegamento tra dispositivi di calcolo, di comunicazione e accessori, mediante un sistema radio a basso costo, a bassa potenza e ridotta portata. Il progetto del consorzio venne chiamato Bluetooth[1]. Sebbene l'idea originale fosse solo quella di liberarsi dei cavi di collegamento tra i dispositivi, il suo ambito ben presto si espanse fino ad invadere il settore delle LAN wireless, rendendolo più utile ma contrastando l'emergente 802.11.

Nel Luglio 1998 SIG Bluetooth pubblicò la specifica della versione 1.0 e subito dopo, il gruppo di standard IEEE 802.15, adottò tale documento come punto di partenza per un proprio lavoro, con l'intento di standardizzarlo. Anche se la specifica prodotta dal SIG riguardava tutto lo *stack* di strati, il comitato IEEE 802.15 lavorò solo sullo strato fisico e Data-link non contemplando il resto della pila di protocolli. Nel 2002 IEEE ha approvato il primo standard PAN, chiamato 802.15.1, mentre SIG Bluetooth è ancora impegnato nello sviluppo di migliorie che ha portato all'attuale specifica 1.1.

1.3.2.1 La pila di protocolli Bluetooth

Al fine di favorire l'interconnessione di dispositivi di una grande varietà di produttori, le specifiche Bluetooth non solo definiscono un sistema radio, ma introducono anche una pila

[1]Tale nome venne assegnato in onore al re vichingo Harald Blaatand II, in inglese Bluetooth, il quale regnò la Danimarca dal 940 al 981 e fu impegnato nell'unificazione della sua terra con la Scandinavia. Similmente, la tecnologia Bluetooth mira ad unire dispositivi di *computing* personali eterogenei.

di protocolli a strati che permette alle applicazioni di scoprire altri dispositivi Bluetooth nell'area, scoprire quali servizi ci offrono e capire come utilizzare tali servizi. Nell'intento di capire questo modello di comunicazione è utile tracciare un confronto tra la pila di protocolli Bluetooth e il modello standard OSI, anche se in verità gli strati non combaciano esattamente.

La Figura 1.11 mostra strato per strato il modello OSI e Bluetooth con le rispettive gerarchie di comunicazione. A differenza del modello OSI che ne definisce sette, il protocollo Bluetooth presenta otto strati. Il primo di questi è chiamato strato *Radio* ed è responsabile dell'interfacciamento elettrico sul mezzo di comunicazione, della codifica/decodifica, della modulazione/demodulazione e della trasmissione dati. La banda di frequenze usate è quella delle ISM senza licenza che operano a bassa potenza a $2.4GHz$ su modulazione di tipo *Frequency Shift Keying* (FSK) e con portata massima fino a 10 metri. Tale banda viene divisa in 79 canali da $1MHz$ con 1 bit per Hz, che raggiunge una velocità di $1Mbps$, ma la maggior parte della quale viene consumata per il *checksum*. Per assegnare i canali in modo imparziale si utilizza la tecnica FHSS con 1600 cambi al secondo e un tempo di rotazione di $625msec$. E' da notare come entrambi gli standard 802.11 e Bluetooth operino nella stessa banda delle ISM e sugli stessi 79 canali, potendo interferire pertanto elettricamente l'uno con l'altro. Dato che 802.11 e 802.15 sono entrambi standard IEEE, il comitato sta cercando di risolvere il problema, ma non è facile trovare una soluzione perchè entrambi i sistemi utilizzano la banda ISM per la stessa ragione, e cioè, perchè non richiede alcuna licenza. Solo una soluzione basata sul mercato porrà fine alla disputa, imponendo alla parte più debole politicamente ed economicamente di modificare il suo standard ed eliminare l'interferenza.

Lo strato *Baseband* assomiglia molto ad un sottostrato MAC in quanto trasforma il flusso di bit grezzi e definisce alcuni formati chiave. Questo strato si sovrappone con lo strato *Link Control* al fine di coprire tutte le funzionalità dello strato Data-link di OSI. Assieme, questi due livelli, controllano i collegamenti fisici sullo strato Radio assemblando pacchetti, controllando la frequenza di *hopping* ed eseguendo il controllo di errore e di flusso.

Al di sopra di questi due livelli un confronto diretto tra i due modelli di riferimento diventa meno chiaro. Lo strato di Rete OSI è responsabile del trasferimento dati attraverso la rete, indipendentemente dalla topologia di rete e dal tipo di mezzi di comunicazione utilizzati dalle varie reti. Questo ruolo nel protocollo Bluetooth viene assunto dallo strato *Link Manager* (LM), il quale controlla e configura i collegamenti con gli altri dispositivi Bluetooth. L'LM è responsabile della connessione dei nodi slave ad una piconet e della creazione degli indirizzi dei suoi membri attivi (vedi oltre). In aggiunta l'LM serve per stabilire un collegamento dati asincrono connectionless a bassa potenza, chiamato *Asynchronous ConnectionLess* (ACL), ed uno voce a bassa potenza sincrono connection-oriented chiamato *Synchronous Connection-Oriented* (SCO). Lo strato Trasporto OSI è responsabile dell'efficenza e del *multiplexing* dei dati attraverso la rete. Questa funzione nella pila Bluetooth è svolta in parte dallo strato LM e in parte dall'*Host Controller Interface* (HCI), il quale permette la comunicazione tra un *host* separato ed un modulo Bluetooth.

Lo strato Sessione OSI è responsabile della manutenzione del servizio e del controllo di flusso dei dati. In accordo a questa funzione, lo strato *Logical Link Control and Adaptation*

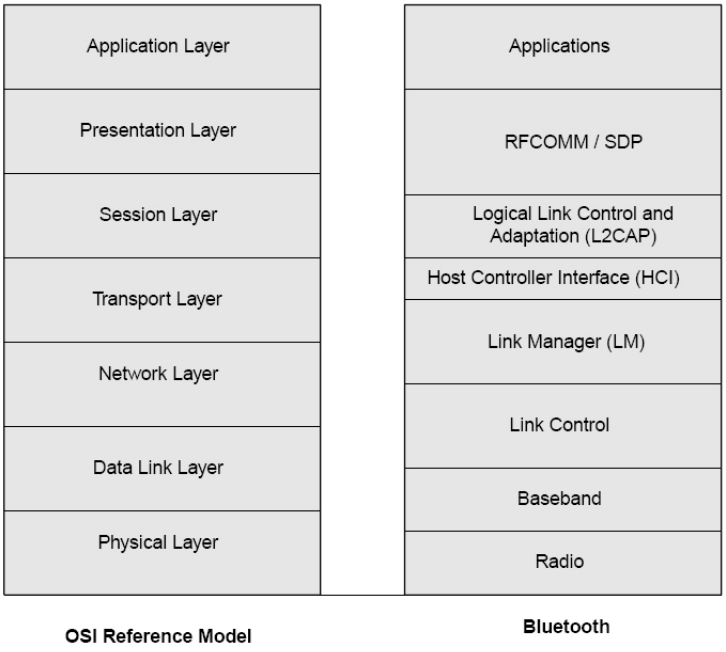

Figura 1.11: Modello OSI e Bluetooth a confronto.

(L2CAP) multiplexa i dati provenienti dagli strati alti e li converte in pacchetti di diversa dimensione. Lo strato Presentazione fornisce una rappresentazione comune dei dati per gli strati inferiori, allo stesso modo lo strato *Service Discovery Protocol* (RFCOMM/SDP) fornisce un modo comune di presentazione emulando una comunicazione seriale simile a RS232, dal momento che Bluetooth lavora principalmente su collegamenti punto-punto. La porzione SDP di RFCOMM permette ai dispositivi Bluetooth di scoprire quali servizi supporta l'altro dispositivo.

Infine lo strato Applicazione gestisce la comunicazione tra gli host connessi.

1.3.2.2 Architettura Bluetooth

Le reti Bluetooth possono operare in due modalità distinte: come *Master* o come *Slave*. Il master è responsabile del settaggio della frequenza di hopping con la quale si sintonizzerà uno slave che intende connettersi a tale rete. In questo modo gli slave si sincronizzano con il master in tempo e frequenza applicando la sua sequenza di hopping. Bluetooth specifica una portata radio di circa 10 metri e supporta fino a sette dispositivi in una piconet. All'interno di una piconet un master ed uno slave comunicano attraverso una comunicazione *full-duplex* punto-punto. Bluetooth può supportare tre canali voce full-duplex contemporaneamente all'interno di una piconet ad un data-rate di $721Kbps$, mentre trasmette ad una potenza di $800mA$.

Ogni dispositivo Bluetooth possiede un'indirizzo univoco ed un proprio segnale di clock. Lo strato Baseband contiene un'algoritmo che è in grado di calcolare la frequenza di hopping partendo dall'indirizzo e dal clock del master. Gli slave pertanto possono usare tale

algoritmo per sincronizzarsi con il dispositivo master, e dunque, essere sincronizzati a loro volta tra loro stessi.

Il traffico dati viene controllato dal dispositivo master. Esso concede agli slave il permesso di trasmettere allocando intervalli temporali per il traffico voce o dati utilizzando una tecnica a divisione di tempo, la *Time Division Multiplexing* (TDM). Il master inoltre controlla la disponibilità totale di banda e come essa dovrà essere suddivisa tra tutti gli slave presenti. Bluetooth permette inoltre di avere configurazioni multi-hop che sono ottenute mediante il concetto di *Scatternet*, nel quale diversi master di differenti piconet possono stabilire collegamenti con ogni altro (Vedi Figura 1.12).

Come abbiamo quindi accennato, le reti wireless Bluetooth possono essere implementate secondo due topologie: le *Piconet* e le *Scatternet*. Una piconet è una collezione di dispositivi slave che comunicano insieme con un dispositivo master. Questi possono essere configurati o secondo una comunicazione punto-punto dove esistono solo un master ed uno slave in tutta la rete oppure secondo una configurazione punto-multipunto dove un master è connesso a più slave appartenenti alla stessa rete. Nella Figura 1.12 ognuna delle due piconet mostrano un'architettura punto-multipunto in cui il master diventa il gestore della rete ed agisce come controllore centralizzato della comunicazione. La piconet di destra inoltre mostra la presenza di sette slave nella rete che è anche il numero massimo consentito. I dispositivi appartenenti alla rete possono anche essere non attivi e considerati in uno stato *Parked* come mostrato in figura dai dispositivi grigi. Dato che tutti i dispositivi slave di una piconet si collegano con il master, la comunicazione diretta tra dispositivi è concessa solamente tra master e slave e non direttamente tra slave, in modo tale che una comunicazione tra di loro passi attraverso il master.

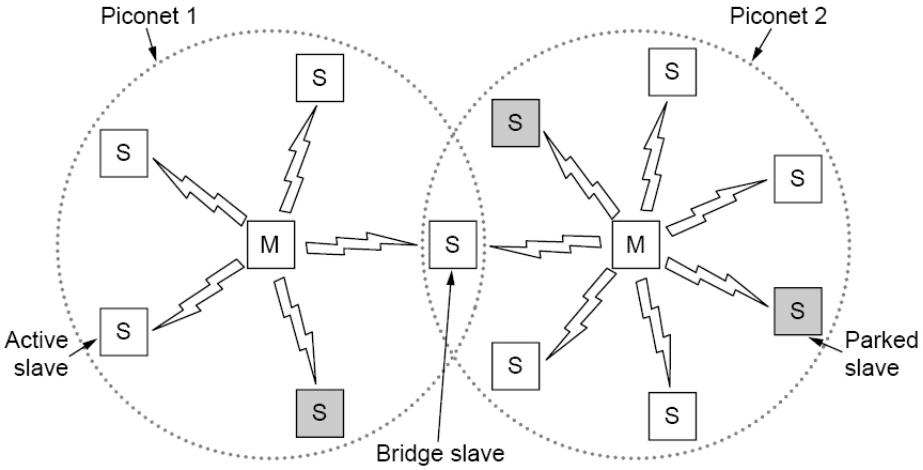

Figura 1.12: Due piconet che formano una scatternet.

La sovrapposizione di due o più piconet forma una scatternet. In Figura 1.12 esiste un dispositivo condiviso da entrambe le piconet che funge da ponte per la creazione della scatternet e attraverso il quale passano tutti i pacchetti scambiati tra nodi di differenti piconet. Può anche succedere che sia un dispositivo master di una piconet ad essere incluso

in un'altra piconet ed in questo caso, all'interno della nuova rete, tale dispositivo assumerà il ruolo di slave dato che nessun master può servire più di una piconet. Quando un dispositivo slave desidera comunicare con uno slave di un'altra piconet, i master di ogni piconet devono essere coinvolti nella consegna dei pacchetti attraverso le reti. L'unico problema sollevato da questa configurazione a multi-hop è un degrado di performance a causa di un potenziale segnale di interferenza tra piconet adiacenti.

Capitolo 2

Mobile Ad Hoc Networks

Una *Mobile Ad hoc NETworks* (MANET) è un sistema autonomo di host mobili in cui la connettività è basata sulla reciproca prossimità dei nodi, i quali collaborano al fine di fornirsi vicendevolmente il routing. Come conseguenza di questo fatto, le reti MANET non richiedono la disponibilità di infrastrutture fisse e preconfigurate in quanto i dispositivi degli utenti possono liberamente interagire tra loro. I nodi di una MANET infatti formano una topologia distribuita velocemente riconfigurabile ed altamente dinamica.

Dati i rapidi cambiamenti di topologia, le informazioni possedute dai nodi riguardo lo stato dei collegamenti, diventano presto obsolete, e di conseguenza è necessario un continuo scambio di dati tra i nodi, al fine di mantenere aggiornate le informazioni di rete.

Le reti ad hoc usano un'architettura di comunicazione di tipo peer-to-peer che può essere, a seconda della locazione dei diversi nodi che formano la topologia, sia a *single-hop* sia a *multi-hop*. In questo modo ogni nodo può trovarsi nella situazione di ricevere un pacchetto, di trasmetterlo oppure di inoltrarlo (*forward*) verso un'altra destinazione. Nel caso di comunicazione single-hop è possibile raggiungere un determinato nodo direttamente mediante un singolo passo (vedi Figura 2.1 dove il nodo A raggiunge direttamente il nodo B), mentre nel caso di comunicazione multi-hop il nodo destinazione non può essere raggiunto direttamente dal nodo mittente, ma solo attraverso altri nodi intermedi, e dunque mediante più passi (vedi Figura 2.1 dove il nodo A a causa della distanza non raggiunge direttamente il nodo G ma indirettamente attraverso i nodi C e D).

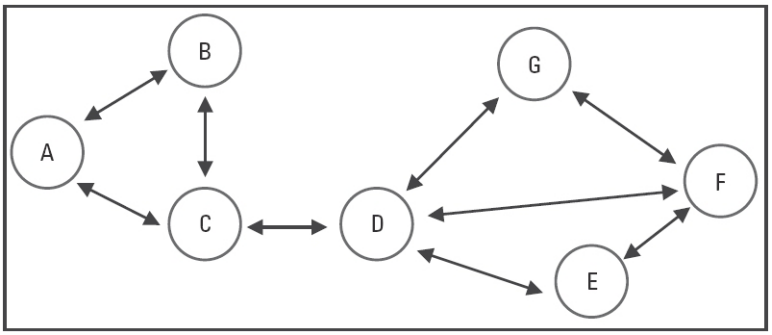

Figura 2.1: Esempio di rete wireless ad hoc.

Le reti mobili ad hoc hanno diverse caratteristiche peculiari che le distinguono dalle reti cablate:

- **Topologia dinamica**: A causa del possibile movimento dei nodi rapido ed imprevedibile ed alle condizioni di propagazione che mutano velocemente, le informazioni di rete, come lo stato dei link, diventano presto obsoleti. Questo porta a frequenti riconfigurazioni della rete ed a frequenti scambi di informazioni di controllo sul mezzo wireless.

- **Link asimmetrici**: In un ambiente wireless la comunicazione tra due nodi potrebbe non avvenire equamente in entrambe le direzioni. In altre parole se un nodo n si trova entro il range di comunicazione di un nodo m, non è detto che valga anche il contrario in quanto ogni nodo ha un proprio range di comunicazione. In Figura 2.1 sono stati indicati tutti link bidirezionali ma in realtà non è detto che sia così.

- **Comunicazione multi-hop**: In una rete ad hoc ogni nodo può agire come trasmettitore, come ricevitore oppure come stazione di instradamento dei pacchetti. In questo modo i pacchetti da un nodo trasmettitore (sorgente) possono essere ricevuti da un nodo ricevitore (destinazione) attraverso diversi nodi intermedi. Tuttavia il successo delle operazioni di una rete ad hoc sarà compromesso se un nodo intermedio, partecipante alla comunicazione tra i nodi pari, si muove ed esce improvvisamente dal range di comunicazione oppure si spegne proprio durante il trasferimento dei messaggi. La situazione è anche peggiore se in tale caso non esistono altri cammini verso il nodo destinazione. Nella situazione di rete di Figura 2.1 se il nodo D si muove ed esce dal range di comunicazione di C si verifica una partizione della rete in due parti distinte ($\{A, B, C\}; \{E, F, G\}$) e l'eventuale comunicazione tra C ed F verrebbe interrotta.

- **Operazioni decentralizzate**: Le reti ad hoc sono architetture di rete che possono essere rapidamente configurate e che non necessitano di infrastrutture preesistenti e di controllori centralizzati. Mentre in un rete wireless cellulare esistono diverse entità centralizzate come il BS, l'MSC e l'HLR, in una rete ad hoc queste non esistono e tale mancanza richiede algoritmi distribuiti più sofisticati per svolgere le medesime funzioni.

- **Connessioni a banda limitata**: Le connessioni wireless hanno e continueranno ad avere significativamente meno capacità delle loro controparti cablate. Per di più, lo throughput realizzato dalle comunicazioni wireless è spesso molto minore del massimo teorico permesso dal mezzo radio a causa degli effetti di accessi multipli, rumore e condizioni di interferenza. Un effetto di questa relativamente bassa capacità delle connessioni è che la congestione nella comunicazione è un effetto tipico piuttosto che un'eccezione. Così le applicazioni distribuite eccederanno sempre più la capacità della rete e questa tendenza continuerà a crescere con l'aumento delle applicazioni multimediali e collaborative.

- **Operazioni ad energia limitata**: I nodi mobili di una rete ad hoc dipendono strettamente dalla capacità delle loro batterie. Per questi nodi infatti la linea guida

di progetto più importante è senza dubbio il basso consumo di energia. Un modo per ottenere questo è ottimizzare la potenza di trasmissione e limitare il numero di trasmissioni di ogni nodo.

Alla luce di queste nuove caratteristiche di rete, nasce l'esigenza di nuovi protocolli in quanto cadono le assunzioni dei tradizionali protocolli per reti cablate. Tutte le caratteristiche appena elencate rappresentano dei parametri di performance per il progetto di queste nuove soluzioni.

2.1 Architettura delle reti MANET

Formalmente è possibile modellare una MANET mediante un *grafo di connessione* $G =< V, A >$, come quello indicato in Figura 2.2, dove gli elementi dell'insieme V sono i nodi della rete (i vertici di G) mentre quelli dell'insieme $A \subseteq V \times V$ sono le connessioni tra i vari nodi (gli archi di G). Se in A esiste l'arco (v_i, v_j) allora la stazione v_i è in grado di comunicare con la stazione v_j ma non è vero il contrario in quanto un elemento di A indica un *link* unidirezionale. Si noti che in un contesto mobile e dinamico come quello delle MANET la topologia del grafo, e quindi l'insieme A dei link, varierà nel tempo.

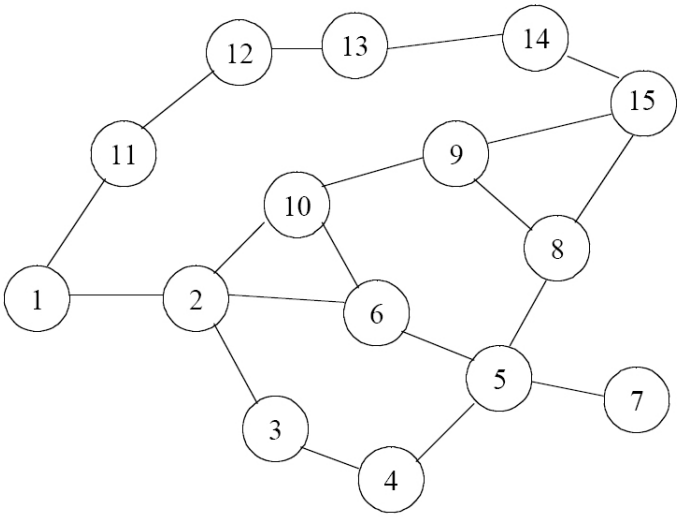

Figura 2.2: Grafo di connessione di una MANET.

E' bene sottolineare che benchè a rigore i link siano da considerare unidirezionali, a causa dell'eterogeneità dei dispositivi, d'ora in avanti questi saranno approssimati sempre come link bidirezionali. Questa assunzione è ragionevole nonostante la difficoltà nel controllo del mezzo trasmissivo, da parte degli apparati di rete, quando i nodi sono sufficientemente vicini.

La struttura di una rete ad hoc può essere classificata in architettura *gerarchica* ed architettura *flat*.

In un'architettura gerarchica i nodi della rete sono dinamicamente partizionati in gruppi chiamati *cluster* in modo tale che la topologia finale della rete è ottenuta dall'aggregazione di nodi in cluster, di cluster in super-cluster e così via. L'appartenenza dei nodi in ciascun cluster cambia nel tempo in risposta alla mobilità dei nodi ed è determinata dal criterio di appartenenza specificato nell'algoritmo di clustering. In ciascun cluster un nodo è scelto per svolgere la funzione di *cluster-head*: l'instradamento del traffico tra due nodi localizzati in due cluster diversi è normalmente svolto dai cluster-head del cluster mittente e destinazione. Al fine di eleggere un nodo del cluster come cluster-head viene utilizzato un algoritmo di selezione dinamica, ma a causa dei continui cambi di appartenenza dei nodi al cluster e della conseguente rielezione del nodo cluster-head le performance del protocollo di routing ne risentono notevolmente. Al fine di ridurre questo problema è stato proposto un approccio completamente distribuito, vedi [LRG97], per la formazione dei cluster e per la comunicazione al loro interno, il quale elimina la necessaria presenza di un nodo cluster-head. Anche questa soluzione presenta debolezze in quanto nel caso di scenari altamente dinamici la riconfigurazione dei cluster e l'assegnamento dei nodi ai cluster richiedono un forte overhead di comunicazione.

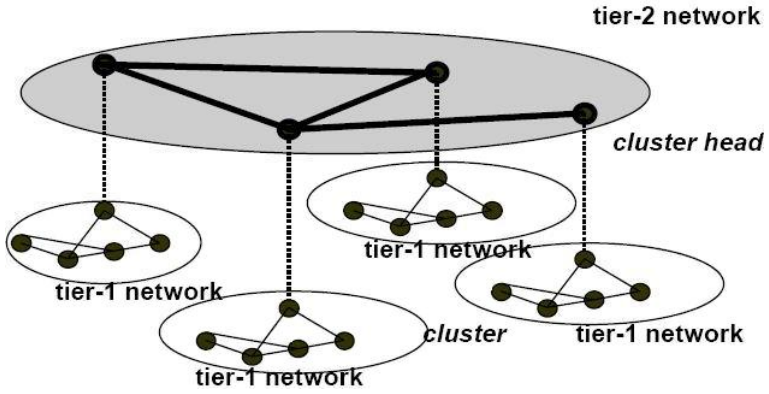

Figura 2.3: Architettura gerarchica a due livelli.

In Figura 2.3 viene mostrata un'architettura gerarchica su due livelli: sul livello 1 sono presenti i vari cluster che compongono la MANET, dove, all'interno di ognuno, viene eletto un cluster-head che comunica con i corrispondenti cluster-head dei diversi insiemi. L'insieme di questi cluster-head forma un super-cluster al livello 2 che ha come nodi membri questo insieme. Se esistessero altri cluster al livello 2 sarebbe necessario eleggere all'interno di ognuno un cluster-head che permetterebbe la formazione di un terzo livello di gerarchia e così via. Come si può notare dal livello 2 la comunicazione inter-cluster avviene per mezzo dei nodi cluster-head.

In un'architettura flat invece, non ci sono cluster ed i nodi vicini possono comunicare direttamente. Si può pertanto dedurre immediatamente che il routing in questo tipo di architettura è migliore del gerarchico (i nodi possono comunicare direttamente e non attraverso il cluster-head) e la rete tende a bilanciare meglio il carico tra cammini multipli in modo da ridurre il collo di bottiglia che si ha tra i nodi cluster nell'approccio gerarchico. Inoltre, scomparendo la presenza di un nodo con funzione di cluster-head che permette

la comunicazione, questo tipo di architettura assicura la persistenza e il funzionamento anche a seguito di guasti o cadute di alcuni nodi, cosa che non può avvenire nell'approccio gerarchico nel quale, a causa della caduta di un nodo critico, cessa la comunicazione. Tuttavia vi è un punto a favore dell'approccio gerarchico: i messaggi devono essere propagati all'interno del solo cluster in modo tale che il numero totale di messaggi propagati risulta relativamente basso. Al contrario nell'approccio flat i messaggi devono essere propagati globalmente per l'intera rete diminuendo così la scalabilità quando il numero dei nodi diventa eccessivo.

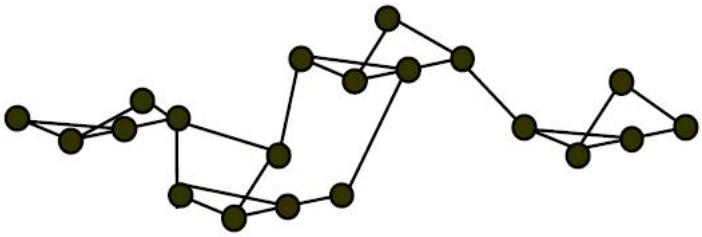

Figura 2.4: Architettura *flat*.

Il principale vantaggio dell'architettura gerarchica è senz'altro la semplicità del processo di gestione della mobilità. Infatti, al fine di limitare l'impatto che un aggiornamento delle informazioni della rete dei singoli nodi può avere, si è scelto di mantenere un informazione completa dei percorsi di routing relativi solamente ai singoli cluster. La comunicazione tra cluster diversi avviene mantenendo all'interno dei cluster-head una tabella contenente le informazioni di tutti i nodi facenti parte del cluster. I cluster-head possono agire quindi come un database che contiene la locazione dei nodi all'interno del suo cluster. In questo modo per determinare dove si trova un determinato nodo, vengono fatte delle query in modo broadcast a tutti i leader dei gruppi che si possono raggiungere. Il cluster all'interno del quale risiede il nodo cercato risponde inviando al mittente della query un messaggio di risposta. Nell'approccio flat è necessario invece implementare un appropriato algoritmo di routing per la gestione della mobilità che incrementa però l'overhead di comunicazione a causa della propagazione di messaggi di controllo attraverso la rete.

2.2 Protocolli di routing

Fornire una comunicazione punto-punto affidabile e ad alta velocità in una rete ad hoc è una sfida significativa, a causa della grande dinamicità della topologia di rete, della mobilità (movimenti casuali) dei nodi, del controllo decentralizzato, delle limitazioni sulla banda e sulla potenza, delle connessioni multi-hop e delle caratteristiche non uniformi della propagazione dei segnali nei canali wireless. Per tutti questi motivi, i tradizionali protocolli di routing progettati per reti cablate, risultano inefficienti ed inappropriati in ambito ad hoc, a causa della loro incapacità di adattarsi velocemente (aggiornare le tabelle di routing) ai cambiamenti dinamici della topologia, ed a causa delle basse performance di rete causate dall'eccessivo overhead prodotto dagli aggiornamenti periodici di routing.

L'utilizzo di questi protocolli risulta pertanto inadatto per le reti ad hoc ed il loro eventuale impiego solleverebbe tutta una serie di problemi:

- La trasmissione tra due host in una rete wireless non necessariamente lavora in modo simmetrico in entrambe le direzioni. In questo modo alcuni cammini determinati da protocolli convenzionali potrebbero non funzionare in alcuni scenari.
- Le reti cablate sono tipicamente configurate per avere solo un piccolo numero di router che connettono ciascuna coppia di reti. In uno scenario wireless ad hoc ciascun nodo è un router ed esistono troppi cammini tra due qualsiasi nodi della rete. Questi cammini ridondanti incrementano non necessariamente la dimensione degli aggiornamenti di routing che devono essere inviati nella rete ed incrementano l'overhead della computazione e della comunicazione.
- L'invio periodico degli aggiornamenti di routing in uno scenario wireless ad hoc rappresenta un consumo eccessivo di banda. Gli aggiornamenti di routing da nodi mobili fuori dal range di trasmissione di ogni altro non interferirà con nessun altro nodo, ma dove molti host mobili sono entro il range di trasmissione di ciascun altro, i loro aggiornamenti di routing consumeranno la banda di ciascun altro nodo presente nella rete.
- In una rete ad hoc altamente dinamica, la frequenza dei cambiamenti di topologia potrebbe essere più alta della frequenza delle richieste di cammini. In questo scenario gli aggiornamenti periodici potrebbero essere un problema in quanto molti di essi non saranno mai usati e quindi inutilmente inviati.
- Molti dei nodi mobili in una rete ad hoc saranno operativi per un determinato periodo di tempo, in quanto limitati dalla capacità della batteria, e l'invio periodico degli aggiornamenti di routing rappresenta un grosso problema al fine del risparmio di energia.

Data la natura delle reti ad hoc, un protocollo di routing dovrebbe essere distribuito, al fine di incrementare l'efficienza, ed i link tra i nodi dovrebbero essere considerati come unidirezionali. Ciascun nodo inoltre dovrebbe essere in grado di prendere le decisioni di routing mediante la cooperazione con gli altri nodi della rete. Infine, date le limitate risorse dei dispositivi mobili, un protocollo di routing dovrebbe essere efficiente dal punto di vista del consumo di energia e non dovrebbe propagare aggiornamenti di routing quando non necessario.

Riassumendo, i principali ostacoli che possono compromettere il corretto funzionamento dei protocolli di routing tradizionali nelle MANET sono:

- La mobilità dei nodi che comporta variazioni al grafo delle connessioni.
- La tecnologia wireless, caratterizzata da banda e portata limitate e da frequenti collisioni ed errori.
- I dispositivi eterogenei e dalle limitate risorse hardware.

Tutte queste caratteristiche rendono le MANET un ambiente in cui l'implementazione di un efficiente servizio di routing può risultare particolarmente difficile. Nasce pertanto l'esigenza di nuovi protocolli in grado di soddisfare i requisiti delle reti ad hoc, ed al contempo, nascono nuove sfide per il loro progetto a causa dei vincoli esistenti sulle risorse (quali banda trasmissiva, frequenza CPU, durata batteria, ecc...), ed alla richiesta reattività del protocollo ai cambiamenti topologici.

2.2.1 Classificazione dei protocolli di routing

Il problema del routing in una rete ha due componenti: il *route discovery* ed il *route maintenance*. Al fine di comunicare con una destinazione, una sorgente deve necessariamente scoprire un cammino adatto per l'invio dei pacchetti. Tuttavia, dato che lo stato dei collegamenti usati nel cammino può cambiare, potrebbe essere necessario dover cambiare il cammino scelto, risultando necessaria una nuova fase di route discovery.

E' possibile classificare i protocolli di routing per reti ad hoc mediante diversi criteri. Un primo criterio è il tipo di informazioni scambiate dal protocollo di routing per propagare la topologia del grafo di connessione:

- **Link State**: Nei protocolli *Link State* (LS) ogni nodo mantiene una copia dell'intera topologia di rete, ricostruita tramite la ricezione di pacchetti *Link State Packet* (LSP) trasmessi da ogni nodo a tutti gli altri per diffondere informazioni sui nodi vicini. Quando un nodo deve instradare un pacchetto calcola il cammino minimo verso la destinazione tramite l'algoritmo di Djikstra.

- **Distance Vector**: Nei protocolli *Distance Vector* (DV) ogni nodo mantiene un *vettore delle distanze* formato dalla tripla (d, n, m), dove d è l'indirizzo del nodo destinazione, n è l'indirizzo del nodo vicino che sarà utilizzato per raggiungere d ed m è la distanza di d attraverso n. Ogni nodo periodicamente trasmette il proprio vettore delle distanze ai vicini. Quando un nodo riceve un vettore delle distanze aggiorna il proprio vettore utilizzando l'algoritmo di Bellman-Ford e quando deve instradare un pacchetto consulta il proprio vettore delle distanze.

- **Position Based**: Nei protocolli *Position Based* (PB) i nodi si scambiano informazioni di tempo e posizione, ottenute ad esempio tramite il *Global Positioning System* (GPS). Quando un nodo deve instradare un pacchetto lo inoltra al nodo più vicino alla destinazione.

Un secondo criterio prevede di classificare i protocolli di routing in base al momento in cui avviene l'elaborazione dei cammini:

- **Proattivi**: Nei protocolli *proattivi* il calcolo dei cammini avviene prima che siano utilizzati. Non è detto che questi siano poi effettivamente utilizzati.

- **Reattivi**: Nei protocolli *reattivi* il calcolo di un cammino viene invece ritardato fino all'istante in cui un nodo sorgente, dovendo trasmettere un pacchetto, effettua una richiesta al servizio di routing: per questa ragione i protocolli reattivi sono anche detti *on-demand*.

- **Ibridi**: Nei protocolli *ibridi* il calcolo avviene in parte in maniera proattiva ed in parte in modo reattivo: per i nodi vicini è possibile pianificare in anticipo i cammini, per i nodi lontani invece si utilizzano le funzionalità reattive; questa strategia è corretta se le applicazioni client del servizio di routing rispettano il *principio di località*, cioè preferiscono per motivi di efficienza la comunicazione locale, con nodi vicini, a quella remota, con nodi distanti.

La ricerca nel settore dei protocolli reattivi e ibridi è stata in particolar modo attiva negli ultimi anni, a causa dei problemi di mobilità dei nodi delle MANET.

Un'altro criterio di classificazione consente di distinguere i protocolli in base alla distribuzione su più livelli delle funzionalità di routing:

- **Struttura flat**: Nei protocolli a *struttura flat* tutti i nodi svolgono le stesse funzioni.
- **Struttura gerarchica**: Nei protocolli a *struttura gerarchica* o *cluster-based* la struttura del servizio di routing si articola su più livelli. Il livello più basso è costituito dai nodi stessi, mentre i livelli superiori sono costituiti da *cluster*, cioè aggregati di nodi o di cluster dei livelli inferiori.

Tipicamente i protocolli della seconda categoria sono più complessi, poichè richiedono che i nodi o i cluster si organizzino in cluster di livello superiore, ma sono più facilmente scalabili.

Un ulteriore criterio di classificazione consente di distinguere i protocolli di routing in base a quando e come il calcolo dei cammini viene eseguito:

- **Decentralizzati**: Nei protocolli *decentralizzati* ogni nodo mantiene l'intera topologia della rete e prende autonomamente le decisioni di routing. I protocolli LS sono esempi di protocolli decentralizzati.
- **Distribuiti**: Nei protocolli *distribuiti* invece, ogni nodo mantiene solo una vista locale e parziale della topologia della rete. Il calcolo di un cammino interessa più nodi. I protocolli DV sono esempi di protocolli distribuiti.

Infine, un ultimo criterio consente di classificare i protocolli di routing in base al modo in cui i pacchetti vengono inoltrati:

- **Source-driven**: Nei protocolli *source-driven* i pacchetti vengono inoltrati verso il nodo destinazione in base alle informazioni di instradamento incluse nel pacchetto stesso dal nodo sorgente: tale tecnica è detta *source-routing*.
- **Table-driven**: Nei protocolli *table-driven* i pacchetti vengono inoltrati da ogni nodo in base a informazioni di routing locali: tale tecnica è detta *routing table-driven* o *hop-by-hop*.

Nelle sezioni successive saranno esaminati sia protocolli unicast che multicast. L'organizzazione seguita sarà quella suggerita dall'applicazione del secondo criterio di classificazione, cioè il momento in cui avviene il calcolo dei cammini.

2.2.2 Routing unicast: protocolli proattivi

In questa sezione verranno analizzati i principali algoritmi di routing unicast di tipo proattivo, il cui compito è quello di instradare i pacchetti da un singolo nodo sorgente ad un singolo nodo destinazione. Il vantaggio di questi protocolli è che il cammino verso la destinazione è immediatamente disponibile quando un'applicazione vuole inviare un pacchetto, cosa molto utile ad esempio per applicazioni interattive.

I principali meccanismi adottati da questi protocolli sono i seguenti:

- Accumulare informazioni sulla topologia della rete su ciascun nodo, in modo da evitare loop ed aumentare la velocità di convergenza.
- Variare dinamicamente la dimensione degli aggiornamenti di cammino.

Al fine di comprendere meglio il funzionamento di tali algoritmi, è utile riformulare il problema del routing nei termini utilizzati al primo paragrafo. Una MANET può essere infatti modellata con un grafo delle connessioni $G =< V, E >$ dove i vertici di V corrispondono ad host mobili e gli archi di E corrispondono a connessione wireless. Un collegamento wireless tra due nodi i e j è stabilito quando la distanza fisica $PD(i, j)$ è minore o uguale al raggio di trasmissione R (tipicamente intorno ai 250m). A causa del movimento casuale dei nodi, si creano e si cancellano continuamente connessioni apportando modifiche continue al grafo G della MANET. Un cambiamento all'insieme E del grafo viene chiamato cambiamento topologico ed è dovuto esclusivamente al movimento dei nodi.

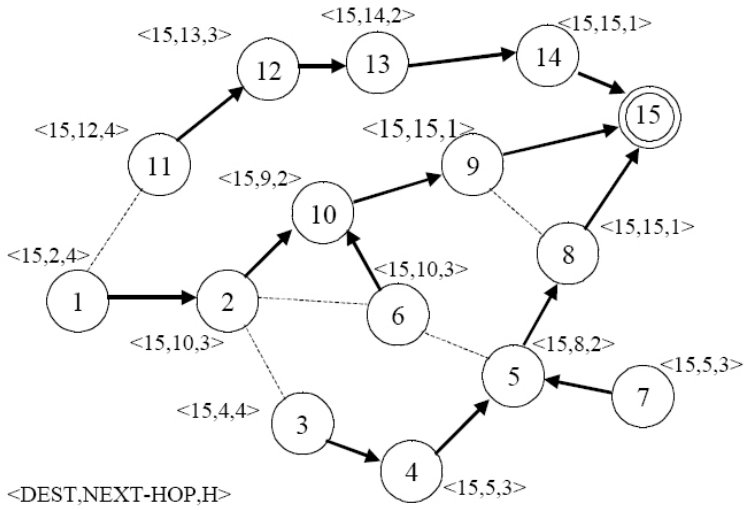

Figura 2.5: Minimum Spanning Tree del grafo di Figura 2.2.

Nel seguito si indicherà con $P(S, D)$ il cammino dalla sorgente S alla destinazione D e con $\mid P(S, D) \mid$ la lunghezza di tale cammino, intesa come numero di hop che i pacchetti devono percorrere prima di raggiungere la destinazione. Il compito del protocollo di routing unicast può essere diviso in due fasi:

1. La computazione del cammino $P(S, D)$.

2. L'inoltro dei pacchetti lungo il cammino.

Un esempio di routing unicast è quello mostrato in Figura 2.5 in cui il nodo destinazione è il numero 15 e nel quale vengono indicate ad ogni nodo le entry delle routing table relative alla destinazione scelta. La terna indica rispettivamente il nodo destinazione, il successivo nodo per raggiungere tale destinazione ed il numero di hop di distanza del nodo dalla destinazione. Il grafo ottenuto indica il *Minimum Spanning Tree* (MST), ovvero l'albero di copertura minimo, per la destinazione 15.

2.2.2.1 DSDV: Destination Sequenced Distance Vector

Il protocollo *Destination Sequenced Distance Vector* (DSDV) è stato uno dei primi tentativi di adattare i classici protocolli DV alle MANET (vedi [PB94]). Gli elementi chiave di DSDV sono:

- Un meccanismo che determina l'età di un cammino, basato sull'incremento monotono di numeri in sequenza, i quali indicano la freschezza del cammino e quali cammini utilizzare per evitare loop ed il problema del conto all'infinito.

- L'uso di aggiornamenti completi del cammino, inviati periodicamente ad ogni intervallo di aggiornamento, oppure di aggiornamenti incrementali inviati ad ogni cambio di topologia.

- Un ritardo introdotto dagli aggiornamenti per cammini che probabilmente sono instabili.

In pratica le entry delle classiche tabelle di routing sono arricchite con numeri di sequenza. Un numero dispari indica una distanza infinita ed è usato per tutte le destinazioni irraggiungibili, mentre i numeri pari sono usati dalla destinazione per marcare gli aggiornamenti di cammino. Per ogni aggiornamento, per una destinazione j, la corrispondente entry nella routing table viene aggiornata con la nuova informazione ricevuta se il cammino ha un numero di sequenza più recente oppure se il numero di sequenza è lo stesso ma la metrica è migliore. L'entry di un cammino viene eliminata dalle tabelle di routing se, per un determinato numero di intervalli di aggiornamento, non è stato ricevuto alcun messaggio.

In DSDV, i cammini considerati stabili sono propagati immediatamente, mentre gli altri possono essere ritardati in accordo ad un tempo medio di assestamento chiamato *settling time*, trascorso il quale il cammino può essere considerato stabile e dunque propagato. Facendo in questo modo è possibile prevenire la propagazione di cammini instabili, come ad esempio quei cammini che molto probabilmente verranno rimpiazzati da cammini migliori nell'immediato futuro. Per il calcolo del settling time un nodo usa una tabella che memorizza tre campi per ogni destinazione:

- L'indirizzo del nodo destinazione.

- L'ultimo settling time.
- Il settling time medio.

Gli aggiornamenti sono inviati periodicamente ad intervalli di ΔT secondi oppure incrementalmente a seconda dei cambiamenti di topologia.

2.2.2.2 OLSR: Optimized Link State Routing

L'*Optimized Link State Routing* (OLSR) è l'ottimizzazione del classico protocollo LS adattato per reti ad hoc (vedi [C01]). L'idea chiave di questo algoritmo è l'uso di nodi chiamati *Multipoint Relay* (MPR) che smistano pacchetti nella rete in modo efficiente riducendo i pacchetti duplicati nella stessa regione. Il protocollo inoltre utilizza collegamenti bidirezionali per il routing in modo tale da evitare il problema del trasferimento dei pacchetti su link unidirezionali. Ogni nodo i seleziona, in modo indipendente da ogni altro nodo, un insieme minimo di MPR denotato da MPR(i), e scelti tra i vicini che distano un solo hop. I nodi contenuti in questo insieme hanno la seguente proprietà: ogni nodo a distanza di due hop da i deve avere un link simmetrico verso l'insieme MPR(i). In altre parole, l'unione dell'insieme dei vicini ad un solo hop di MPR(i) contiene l'insieme dei vicini a due hop (vedi Figura 2.6). L'insieme MPR permette di realizzare un flooding efficiente: quando un nodo i vuole inviare un messaggio, questo lo invia solamente ai nodi contenuti nell'insieme MPR(i), i quali a loro volta inviano il messaggio ai nodi contenuti nei loro MPR e così via.

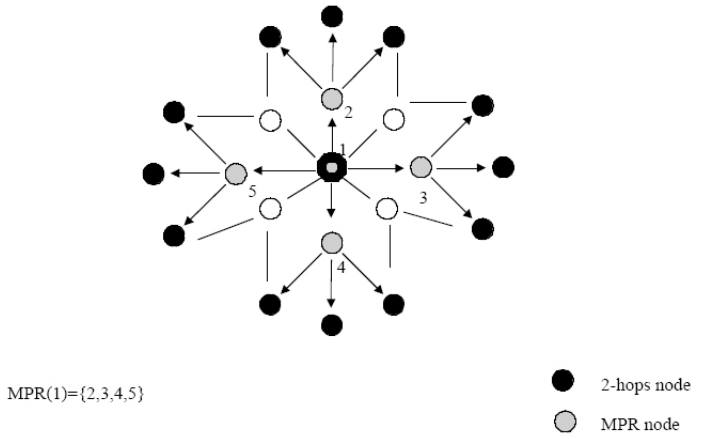

Figura 2.6: Esempio di flooding usando i nodi dell'insieme MPR.

L'insieme *Multipoint Relay Selector* (MPR Selector) di un nodo j è costituito dall'insieme dei vicini che lo hanno incluso nel loro insieme MPR. Ogni nodo periodicamente invia il proprio insieme MPR Selector ed un messaggio di controllo speciale chiamato *Topology Control* (TC). Usando questo messaggio un nodo annuncia alla rete che è raggiungibile ai nodi contenuti nel suo MPR Selector. Il messaggio TC è marcato da un numero di sequenza che cambia quando il proprio MPR Selector cambia.

Per aumentare la reattività ai cambiamenti di topologia limitando l'overhead del protocollo, l'intervallo di tempo tra due messaggi TC consecutivi può essere diminuito al minimo se viene rilevato un cambiamento nel proprio MPR Selector. Inoltre, se un nodo ha un MPR Selector vuoto potrebbe non generare alcun messaggio TC. Tuttavia, quando l'MPR Selector diventa vuoto, potrebbe ancora inviare messaggi TC vuoti per un certo periodo di tempo, invalidando i precedenti. Le informazioni ottenute dai messaggi TC sono utilizzate per costruire la topologia di rete e poi le tabelle di routing.

Un nodo che utilizza OLSR pertanto adotta le seguenti tre strutture dati:

- Tavola contenente le informazioni sui vicini.
- Tavola contenente le informazioni sulla topologia.
- Tabelle di routing.

2.2.2.3 FSR: Fisheye State Routing

Il *Fisheye State Routing* (FSR) è un protocollo basato sulla cosiddetta tecnica ad *"occhio di pesce"* proposta per ridurre la dimensione delle informazioni richieste per rappresentare dati grafici (vedi [GHP01]). Le novità in FSR sono:

- La trasmissione dei pacchetti sullo stato dei link solo ai vicini piuttosto che in flooding.
- L'introduzione della nozione di *scope* per definire porzioni di rete con differente accuratezza nelle informazioni di routing.

FSR è simile ai protocolli LS in quanto ogni nodo mantiene memorizzata la topologia dell'intera rete ed in più una tavola dei cammini ed una lista dei vicini. Diversamente dai protocolli LS invece, che inviano aggiornamenti a tutta la rete, FSR invia i pacchetti sullo stato dei link solo ai vicini ed utilizza dei numeri di sequenza per indicare la freschezza delle informazioni come in DSDV. Un aggiornamento di cammino consiste di una sequenza di coppie $< indirizzo-destinazione, lista-di-vicini >$. Per realizzare la tecnica *"fisheye"* infine FSR introduce il concetto di scope: lo scope del nodo i è definito come l'insieme dei nodi che possono essere raggiunti entro h hop da i.

Gli aggiornamenti di routing sono generati a frequenze differenti che aumentano al diminuire dello scope. Questo produce una riduzione nel numero e nella dimensione dei messaggi di aggiornamento e di conseguenza dell'overhead del protocollo. In genere FSR è basato su un compromesso tra cammino ottimo e costo del cammino. Esso mantiene informazioni di distanza e qualità sul vicinato di un nodo con dettagli progressivamente sempre minori all'aumentare della distanza. In questo modo una conoscenza imprecisa del cammino migliore verso la destinazione può portare all'utilizzo di alcuni cammini non ottimi. Tuttavia, il protocollo ha buone potenzialità anche su reti a larga scala dal momento che i cambiamenti di topologia che avvengono in regioni lontane da un nodo non producono lo stesso aumento di traffico come quello che si avrebbe con altri protocolli proattivi single-hop. Per di più, quando un pacchetto si avvicina alla destinazione, esso attraversa scope con livelli di precisione sempre maggiore sulla destinazione, e ciò mitiga la perdita di precisione delle informazioni di routing.

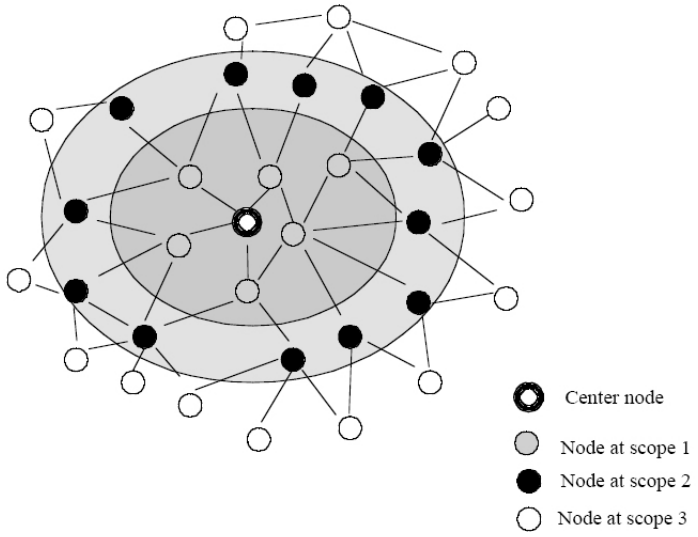

Figura 2.7: Esempi di scope nel protocollo FSR.

2.2.2.4 WRP: Wireless Routing Protocol

Il *Wireless Routing Protocol* (WRP) appartiene alla classe generale di algoritmi *Path-Finding Algorithms* (PFA), definiti come l'insieme di algoritmi distribuiti *shortest-path* che calcolano il cammino più corto verso ogni possibile destinazione.

Gli algoritmi PFA eliminano il problema del conto all'infinito ed evitano i loop a breve termine che possono essere generati da alcuni algoritmi. Il protocollo usa le seguenti quattro strutture dati:

- Distance table.
- Routing table.
- Link-cost table.
- Message Retransmission List (MRL).

La *Distance table* di un nodo i è una matrice contenente, per ogni destinazione j ed ogni vicino di i detto k, la distanza da j e dal predecessore k. La *Routing table* di un nodo i è arricchita con il predecessore ed il successore nel cammino verso la destinazione ed in più anche di un tag che specifica se l'entry corrisponde ad un cammino semplice, un loop o una destinazione non etichettata mediante tale tag. Con questa informazione, WRP è in grado di evitare il problema del conto all'infinito e di ridurre altamente loop forzando ogni nodo ad eseguire controlli di consistenza.

MRL è infine usato per memorizzare l'identificativo di tutti i nodi che non hanno confermato la ricezione di un aggiornamento di cammino. Trascorso un timeout, tali aggiornamenti sono ritrasmessi a questi nodi. Inoltre WRP utilizza la trasmissione di messaggi di HELLO per determinare la connettività con i vicini in caso non vengano ricevuti aggiornamenti di cammino per un certo intervallo di tempo.

2.2.3 Routing unicast: protocolli reattivi

Un approccio differente per il routing in ambienti MANET è il cosiddetto routing *on demand*. Questo approccio è caratterizzato dall'eliminazione delle tabelle di routing convenzionali sui nodi e conseguentemente della necessità di rilevare i cambiamenti nella topologia di rete. I protocolli on demand calcolano il cammino prima della trasmissione dei pacchetti. Se il traffico dati non viene generato dal nodo, l'attività di routing è del tutto assente. Per questa ragione essi vengono anche chiamati protocolli reattivi.

Un protocollo reattivo è caratterizzato dalle seguenti procedure di manutenzione dei cammini:

- **Path discovery**.
- **Path maintenance**.
- **Path deletion** (opzionale).

L'invio dei pacchetti viene eseguito invece in accordo a due possibili tecniche:

- **Source routing**.
- **Hop-by-hop**.

La procedura di discovery è basata su un ciclo di richiesta-risposta che adotta il flooding di query. L'eventuale destinazione è raggiunta dalla query e questa genera almeno una risposta. Il Path discovery viene attivato ogni qualvolta un nodo voglia inviare un pacchetto e non è conosciuto alcun cammino verso la destinazione. La fase di Path discovery non è richiesta per la trasmissione di ogni singolo pacchetto dati, dal momento che i cammini scoperti rimangono validi per un certo periodo di tempo, permettendo la trasmissione verso quella determinata destinazione. Come risultato della fase di Path discovery, i nodi della rete acquisiscono un nuovo *"routing state"* e memorizzano i cammini scoperti durante tale fase. Le informazioni di routing poi vengono mantenute da una procedura di maintenance fino a che non vengono più utilizzate od esplicitamente cancellate.

2.2.3.1 DSR: Dynamic Source Routing

Le caratteristiche distintive del protocollo *Dynamic Source Routing* (DSR) sono:

1. L'inoltro dei pacchetti attraverso la tecnica del source routing.
2. L'uso aggressivo di cache per memorizzare i cammini completi verso le destinazioni.

La tecnica del source routing presenta i seguenti vantaggi:

- Instradamento dei pacchetti privo di cicli.
- Evita la necessità di aggiornare le informazioni di routing presenti sui nodi intermedi che attraversa il pacchetto.

- Permette ai nodi di memorizzare le informazioni di cammino esaminando i pacchetti inoltrati.

DSR utilizza dei pacchetti di controllo per la fase di Route discovery chiamati *route request* (RREQ) e *route reply* (RREP) e vengono generati da un nodo S che tenta di inviare un pacchetto verso una destinazione D e non possiede alcun cammino in cache. Questa fase è basata sul flooding di un pacchetto RREQ per la rete, che include i seguenti campi: l'indirizzo del mittente, l'indirizzo della destinazione cercata, un numero univoco per identificare la richiesta ed un campo per il cammino.

Alla ricezione di una RREQ un nodo intermedio può:

- Rispondere ad S con una RREP se possiede nella sua cache un cammino verso la destinazione (in questo caso il cammino rispedito è la concatenazione del cammino accumulato fino al nodo da S e del cammino contenuto in cache fino al nodo D).

- Scartare il pacchetto se già ricevuto.

- Appendere il suo identificatore nel campo route ed inviare in broadcast il pacchetto ai suoi vicini, negli altri casi.

Alla ricezione di un pacchetto RREQ, la destinazione risponde ad S con un pacchetto RREP. Il pacchetto RREP è inoltrato attraverso il cammino ottenuto rovesciando il cammino memorizzato nel pacchetto di richiesta RREQ, dal momento che i link sono assunti bidirezionali. In Figura 2.8 viene mostrato il flooding di un pacchetto RREQ attraverso la rete mentre in Figura 2.9 la risposta del nodo D alla richiesta, utilizzando il cammino inverso percorso dal pacchetto RREQ.

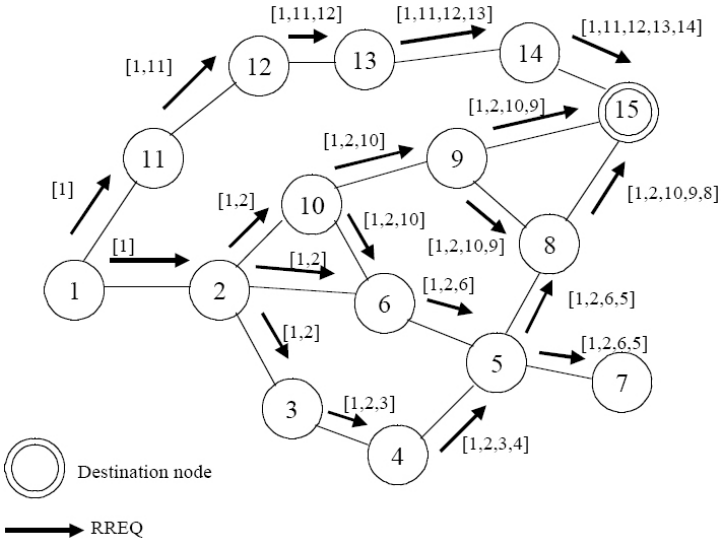

Figura 2.8: Propagazione di un RREQ in DSR.

E' stata anche proposta una variante di DSR chiamata RODA (vedi [KTC00]) che supporta link asimmetrici. In questo caso la destinazione genera una nuova fase di route discovery che diffonde un pacchetto RREQ per la sorgente, aggiungendo in *piggyback* la lista dei nodi attraversati dalla RREQ originata dalla sorgente.

Il meccanismo base per il route maintenance è il seguente. Quando un nodo intermedio rileva che il link verso il nodo next-hop verso la destinazione è caduto, esso lo rimuove dalla sua cache e ritorna un messaggio *Route error* alla sorgente. La sorgente S alla ricezione di questo messaggio di errore, genera una nuova fase di route discovery.

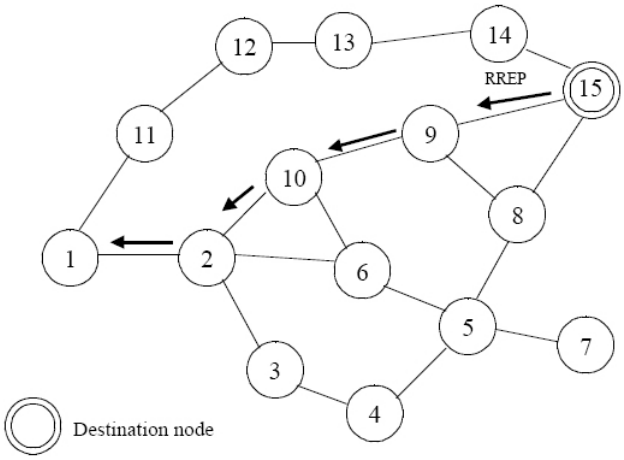

Figura 2.9: Propagazione di un RREP in DSR.

In DSR risulta critica anche la gestione della cache, al fine di ottenere buone performance specialmente in condizioni di alto traffico. Se i cammini non più esistenti non vengono immediatamente rimossi dalla cache infatti, questi possono essere usati in risposta ad altri RREQ ed essere velocemente memorizzati erroneamente da altri nodi. Per risolvere questo problema sono stati proposti diversi schemi:

- **Fixed lifetime**: Il tempo di vita di un cammino in cache è predeterminato ad un valore costante.

- **Adaptive lifetime**: Il tempo di vita di un cammino è determinato in modo euristico in base agli eventi osservati relativamente ai nodi interessati dal cammino.

- **Negative cache**: Ogni nodo inserisce in una cache i nodi che hanno dato origine a degli errori, disattivando temporaneamente il caching dei cammini che li interessano.

- **Wide error notification**: Quando viene rilevato un errore questo è propagato tramite broadcast fino ai nodi sorgente, in modo da rimuovere velocemente i cammini non più utilizzabili.

2.2.3.2 AODV: Ad hoc On demand Distance Vector

L'*Ad hoc On demand Distance Vector* (AODV) rappresenta la versione reattiva di DSDV. Esso infatti utilizza una procedura di Route discovery derivata da quella di DSDV e per di più utilizza numeri di sequenza per rimuovere dalla cache cammini invalidi e per prevenire loop (vedi [PR99] e [PRD02]). La principale differenza con DSDV però sta nel fatto che un cammino scoperto viene memorizzato localmente nel nodo piuttosto che incluso nell'header del pacchetto.

Il processo di Route discovery viene attivato da un nodo S quando vuole inviare un pacchetto ad un nodo D per il quale non possiede informazioni di routing nelle sue tabelle. Questa fase è basata sul flooding di un pacchetto RREQ simile a quello di DSR. Quando un nodo inoltra un pacchetto di richiesta, esso imposta un cammino inverso da se stesso fino al mittente di tale pacchetto, registrando l'indirizzo del vicino dal quale ha ricevuto la prima copia dell'RREQ. In modo del tutto simile, quando un pacchetto di controllo RREQ viene inoltrato verso la destinazione, un nodo automaticamente imposta il cammino inverso da tutti i nodi indietro fino la sorgente. Gli altri cammini inversi vengono cancellati dopo un certo periodo di timeout.

Ogni entry della tabella di routing è caratterizzata dai seguenti attributi:

- Indirizzo IP della destinazione.
- Prefix Size della rete destinazione.
- Numero di sequenza della destinazione.
- Indirizzo IP del prossimo hop per raggiungere la destinazione.
- Tempo di vita della entry.
- Distanza in hop della destinazione.
- Interfaccia di rete da utilizzare per raggiungere la destinazione.
- Flag di stato attivo o inattivo.

Il *Prefix Size* permette di identificare una sottorete come un'unica entità AODV. Questo permette di rappresentare con unica entry le informazioni di instradamento comuni a tutti i nodi della sottorete ed inoltre consente di eseguire una sola fase di discovery. Il tempo di vita è il tempo, scaduto il quale, l'entry sarà rimossa se non viene usata per inoltrare pacchetti o non viene aggiornata da AODV. Lo stato di attività è impiegato durante la fase di manutenzione.

AODV usa tre tipi di messaggi di controllo:

RREQ I messaggi di *Route REQuest* sono utilizzati nella fase iniziale di discovery.

RREP I messaggi di *Route REPlay* sono utilizzati nella fase finale di discovery.

RERR I messaggi di *Route ERRor* sono utilizzati nella fase di manutenzione.

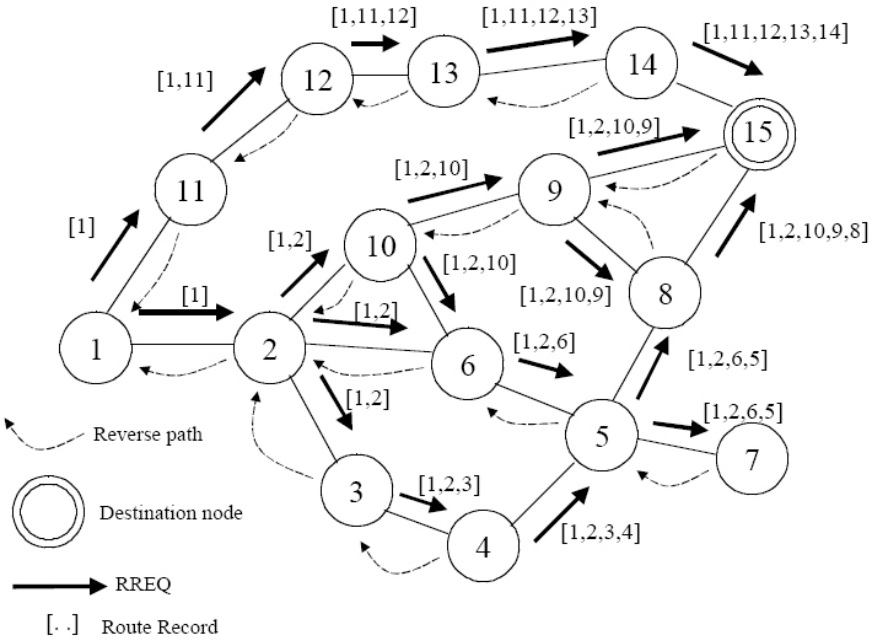

Figura 2.10: Propagazione di un RREQ in AODV.

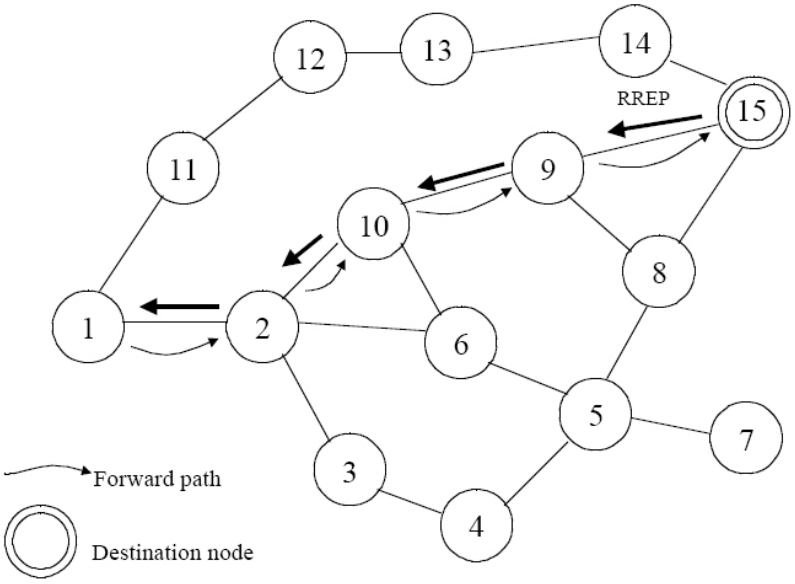

Figura 2.11: Propagazione di un RREP in AODV.

Ogni indirizzo, in ogni messaggio di controllo, ha associato un numero di sequenza, il cui significato dipende dallo scopo dell'indirizzo e del messaggio.

Al termine della fase di discovery, come risultato della trasmissione della fase request-reply, viene creato un nuovo stato di routing sul nodo. Lo stato è composto dalle entry delle routing table che memorizzano i nodi next-hop del cammino fino la destinazione. Un cammino viene cancellato se non viene usato per un dato intervallo di tempo. Ogni volta che un cammino viene utilizzato questo intervallo di tempo viene azzerato.

La fase di route maintenance è basata sulla trasmissione periodica di messaggi HELLO. Quando viene rilevata la rottura di un link, un nodo invia un pacchetto *Unsolicited Route Reply* a tutti i suoi vicini attivi invalidando tutti i cammini che usano il link caduto. Questi nodi, a loro volta, inoltrano i pacchetti ai loro rispettivi nodi attivi in modo tale che tutte le sorgenti attive siano notificate. Alla ricezione di un Unsolicited Route Reply, la sorgente deve inviare una nuova route request.

AODV evita il problema del *broadcast storm*[1] nella fase di propagazione dei messaggi RREQ identificando ogni RREQ e adottando una cache per essi su ogni nodo. Le RREQ sono identificate attraverso l'indirizzo dell'origine ed il valore di un contatore sul nodo origine stesso.

2.2.3.3 TORA: Temporally Ordered Routing Algorithm

Il *Temporally Ordered Routing Algorithm* (TORA) appartiene alla cosiddetta famiglia degli algoritmi *"link reversal"* (vedi [PC97]). TORA è progettato per reagire efficientemente ai cambiamenti di topologia e per trattare anche le partizioni di rete. Il nome del protocollo deriva dall'assunzione di avere un clock sincronizzato (ad esempio un segnale GPS) per ogni nodo della rete, al fine di ordinare temporalmente gli eventi che avvengono nella rete.

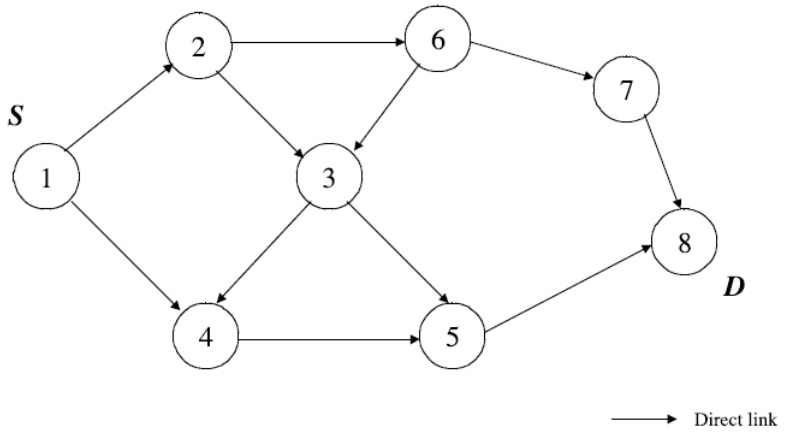

Figura 2.12: Esempio di DAG per il nodo destinazione 8.

[1] Il broadcast storm è un effetto collaterale che consiste nella ricezione e ritrasmissione di copie multiple dello stesso messaggio durante il flooding.

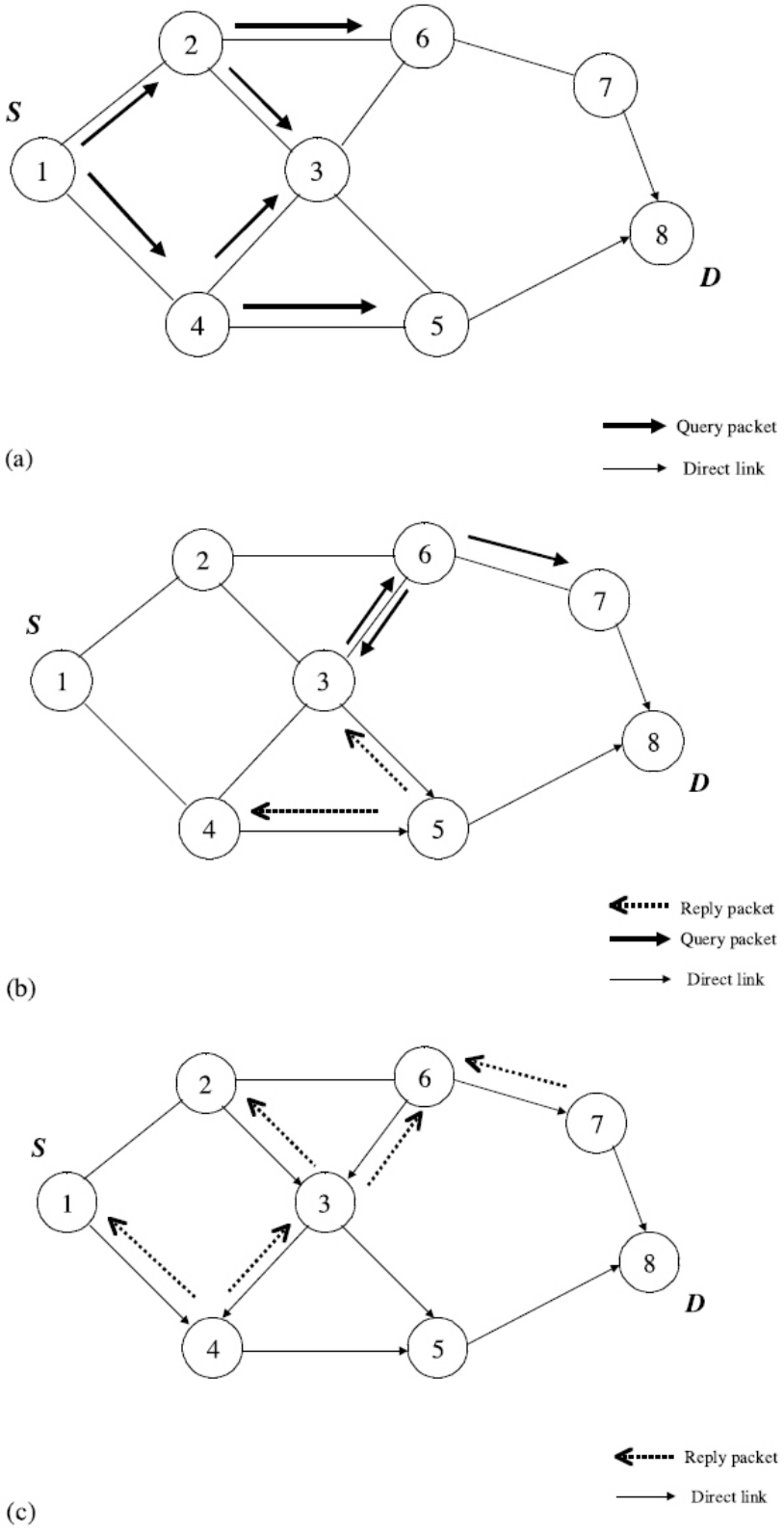

Figura 2.13: Esempio di ciclo query-reply in TORA: (a)generazione e propagazione di una richiesta; (b)propagazione richiesta e generazione risposta; (c)propagazione risposta.

TORA fornisce il routing sfruttando un approccio completamente differente da quelli visti fin'ora. In questo protocollo l'ottimalità del cammino è un concetto secondario. L'obiettivo principale di TORA è trovare un cammino stabile che possa essere velocemente e localmente riparato. Il protocollo costruisce un grafo aciclico diretto (DAG) verso una determinata destinazione. Il DAG è ottenuto assegnando una direzione logica ai collegamenti, sulla base di un'*altezza* oppure di un livello di riferimento assegnato ai nodi. Le altezze dei nodi e la direzione dei link del DAG sono tra loro strettamente correlate. Vale infatti il seguente teorema:

> Siano H_i e H_j rispettivamente le altezze di un nodo i e j, allora dato un DAG, è sempre possibile determinare le altezze dei nodi in modo tale che se esiste il link (i, j) allora $H_i > H_j$.

Se la coppia (i, j) è un link diretto del DAG, i viene chiamato il nodo *upstream* e j il nodo *downstream*. Il grafo DAG ha la seguente proprietà: c'è solo un punto del grafo (la destinazione) dal quale non parte nessun link, mentre tutti gli altri nodi del grafo ne hanno almeno uno, di solito, uscente. Come si nota in Figura 2.12 il nodo D può essere raggiunto seguendo questi link uscenti dai nodi. I loop vengono semplicemente evitati grazie a questa proprietà del DAG.

Il funzionamento del protocollo può essere suddiviso nelle tre tipiche fasi: Route discovery, Route maintenance e Route deletion.

La prima fase si basa sullo scambio di corti pacchetti di controllo query-reply (vedi Figura 2.13). Durante la trasmissione dei pacchetti di risposta, eseguita tra l'altro mediante flooding, i link ricevono una direzione logica (upstream o downstream) basata sulla loro relativa altezza logica, in modo tale che alla fine di tale fase venga creato un DAG che mostri un cammino fino la destinazione. Questo stato di routing può essere visto come una rete di tubi, attraverso i quali fluisce dell'acqua verso la destinazione, i quali possiedono le più basse altezze nella rete.

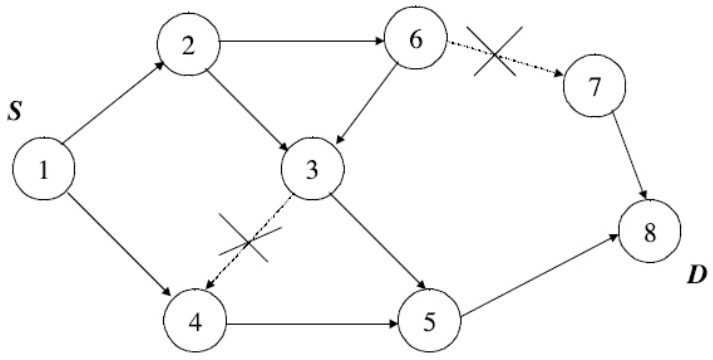

Figura 2.14: Esempio di cambiamento topologico che non richiede operazioni link reversal.

La fese di route maintenance viene attivata per mantenere il DAG ed è basata su una sequenza finita di operazioni *"link reversal"*. Una caratteristica chiave di TORA è che molti cambiamenti topologici potrebbero non generare una reazione a tutti i nodi della

rete. Infatti, se uno dei collegamenti uscenti di un nodo fallisce, ma il nodo continua a possedere almeno un'altro downstream, la destinazione può essere ancora raggiunta attraverso un'altro cammino, ed in questo modo non verrebbe richiesta alcuna fase di riparazione del cammino. Un esempio di questo caso ci è mostrato dalla Figura 2.14 in cui, anche a seguito della rottura di due collegamenti i nodi 6 e 3 possiedono ancora nodi downstream verso la destinazione e il nodo 8 è ancora raggiungibile.

Al contrario, quando un nodo rileva che non possiede alcun nodo downstream, esso genera un nuovo livello di riferimento al fine di diventare massimo globale. Il nuovo livello di riferimento viene propagato nella rete causando un parziale link reversal di quei nodi che, come conseguenza del nuovo livello di riferimento, hanno perso tutti i cammini verso la destinazione. Alla fine dell'attività di riparazione, localizzata vicino al nodo, il DAG viene ristabilito.

TORA è inoltre in grado di rilevare le partizioni di rete. Al rilevamento di una partizione della rete, un nodo invia in flooding un pacchetto vuoto che resetta lo stato di routing.

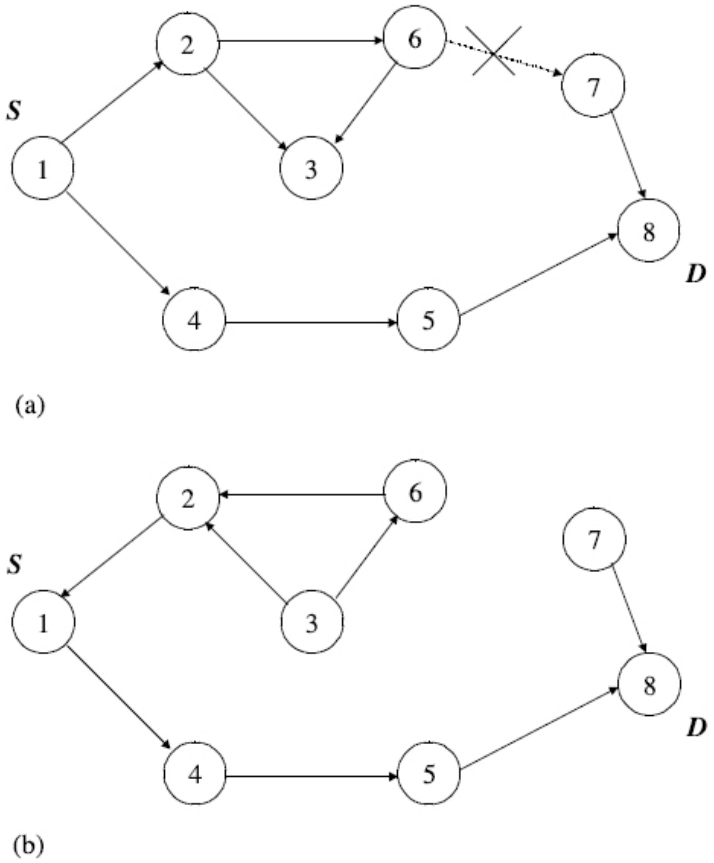

Figura 2.15: Esempio di cambiamento topologico che richiede operazioni di link reversal: (a)Rottura; (b)Nuovo DAG ottenuto.

Un esempio di quest'ultimo caso ci è mostrato in Figura 2.15. Alla rottura del link tra il nodo 6 e 7 il nodo 6 non possiede più alcun nodo downstream verso la destinazione e

solamente grazie ad una fase di rovesciamento dei link è possibile ristabilire il DAG verso il nodo 8.

2.2.3.4 ABR: Associativity Base Routing

L'*Associativity Base Routing* (ABR) è un protocollo on demand attentamente progettato per lavorare in ambienti mobili (vedi [T97]). L'idea chiave di questo protocollo è l'utilizzo della durata dei cammini anzichè della lunghezza come criterio principale di selezione. In ABR, un cammino a più lunga vita è preferito di uno a durata inferiore, anche se la lunghezza del primo è minore di quella del secondo. Operando in questo modo il protocollo utilizza sempre i cammini più stabili, cioè quelli che probabilmente richiedono meno operazioni di maintenance. La stima della durata di un cammino include la combinazione di diverse misurazioni (potenza del segnale, batteria residua, ecc...) e l'introduzione di una nuova metrica, la cosiddetta *"associatività"* tra nodi, che cattura il grado di stabilità di un nodo e lo confronta con quello di altri nodi.

La tecnica proposta per catturare l'associatività è la seguente. Ogni nodo genera un segnale periodico e conta i segnali ricevuti dai suoi vicini per aggiornare i loro *"associativity ticks"*. Questi vengono resettati qualora non venga ricevuto il segnale periodico per un certo intervallo di tempo. Il valore degli associativity ticks permette di classificare i nodi fornendo per ognuno un valore dello stato di mobilità nei confronti dei vicini. Un alto valore per un nodo i rispetto ad un nodo j indica, infatti, che il nodo i era in grado di ricevere molti segnali consecutivi dal nodo j. In questo modo il protocollo assume che è molto probabile per due nodi rimanere vicini tra loro. Il nodo i poi esibisce uno stato di bassa mobilità nei confronti di j ed il link tra i e j viene classificato come a lunga vita. Al contrario un basso valore per gli associativity ticks indica un vicino in transito e dunque un alto valore dello stato di mobilità.

Il protocollo consiste di tre fasi: Route discovery, Route reconstruction e Route deletion. Come per gli altri protocolli on demand la fase di route discovery si basa sul flooding di richieste alle quali non è permesso rispondere ai nodi intermedi. Il cammino è selezionato dal nodo destinazione e memorizzato nei nodi intermedi tra i cammini attivi in modo hop-by-hop. Un nodo sorgente acquisisce un cammino verso la destinazione inviando in broadcast un pacchetto di controllo chiamato *Broadcast Query* (BQ). Quando il pacchetto transita nei nodi intermedi, questi appendono il loro ID e un valore di qualità del cammino che comprende gli associativity ticks. Il nodo destinazione attende un certo periodo di tempo dopo aver ricevuto il primo pacchetto BQ in modo tale da ricevere ulteriori pacchetti ancora lungo il cammino. In questo modo il nodo può scegliere il miglior cammino tra quelli giunti in base al seguente criterio: se un cammino è composto di nodi con alto valore di associativity ticks, allora questo è preferito rispetto ad un cammino con numero di hop minore. Il numero di hop è considerato solo per selezionare il miglior cammino tra due cammini con lo stesso grado di associatività. Una volta selezionato il cammino la destinazione invia un pacchetto BQ reply (BQ-reply) alla sorgente lungo il cammino selezionato. Quando viene propagato il BQ-reply i nodi intermedi coinvolti nella sua trasmissione sono in grado di impostare una route entry nelle loro tabelle come avveniva in AODV.

La fase di route reconstruction (RRC) viene invocata quando l'associazione di stabilità

viene violata. In alcuni casi la procedura può tentare di riparare il cammino corrotto, generando una nuova route discovery, senza la notifica della rottura alla sorgente. Nel caso peggiore viene inviato alla sorgente un messaggio di controllo Route Notification (RN) e viene iniziata una nuova fase di route discovery.

La fase finale di ABR è la route deletion che consiste nell'inviare in flooding nella rete un pacchetto di controllo Route Deletion (RD). Come alternativa a questa costosa procedura, ABR ne propone inoltre una meno costosa chiamata approccio *"state-soft"* che cancella le route entry dopo un certo timeout.

2.2.4 Routing unicast: protocolli ibridi

Questo tipo di protocolli combinano entrambe le tecniche analizzate finora di cui il più importante è il protocollo ZRP.

2.2.4.1 ZRP: Zone Routing Protocol

Il *Zone Routing Protocol* (ZRP) è un protocollo ibrido che mira a combinare i vantaggi di entrambe le tecniche proattiva e reattiva, principalmente per ridurre la latenza necessaria ad acquisire un nuovo cammino e l'overhead di protocollo (vedi [PH99]). Il concetto principale del protocollo è la nozione di *zona*. Una zona $Z(k, n)$ per un nodo n con raggio k, è definita come l'insieme dei nodi che distano dal nodo non più di k hop:

$$Z(k, n) = \{i | H(n, i) \leq k\} \tag{2.1}$$

dove $H(i, j)$ è la distanza in numero di hop tra il nodo i e j. Il nodo n è chiamato nodo *centrale* della zona di routing, mentre il nodo b per cui vale $H(n, b) = k$ è chiamato nodo *periferico* di n. Il valore di k è di solito comparato al diametro della rete e può essere ottimizzato per differenti scenari caratterizzati da diversi gradi di mobilità e traffico.

L'architettura del protocollo è organizzata in quattro grandi componenti: *IntrAzone Routing Protocol* (IARP), *IntErzone Routing Protocol* (IERP), *Borderast Protocol* (BRP) ed un layer-2 *Neighbour Discovery/Maintenance Protocol* (NDP) (vedi [HPS01a] e [HPS01b]). Il primo fornisce cammini in modo proattivo a quei nodi localizzati nella zona di routing del nodo sorgente. Questo può essere basato su qualunque protocollo proattivo con l'unica differenza che gli aggiornamenti di routing non sono propagati ad una distanza maggiore di k hop. IARP usa NDP per conoscere i vicini di un nodo.

Per quei nodi localizzati ad una distanza $k' > k$ dalla sorgente, ZRP utilizza IERP per calcolare, on demand, un cammino interzona. IERP utilizza una forma di flooding selettivo che sfrutta le zone generate dal protocollo IARP. Più precisamente, il flooding è basato sull'invio di pacchetti di query solo ai nodi periferici utilizzando uno speciale tipo di trasmissione multicast, chiamata *bordercast* (vedi Figura 2.17). Quando un nodo riceve il pacchetto query, esso può sia rispondere alla sorgente, se D è un membro della zona di routing, oppure inviare in bordercast la query ai suoi nodi periferici. Eventualmente il pacchetto di richiesta raggiunge un nodo che ha D come membro della sua zona, il quale risponde alla sorgente con un pacchetto di reply. Come per DSR, un cammino viene accumulato nel pacchetto di richiesta durante il forwarding verso la destinazione, oppure,

per ridurre la lunghezza del pacchetto di richiesta, durante la fase di risposta nel pacchetto reply.

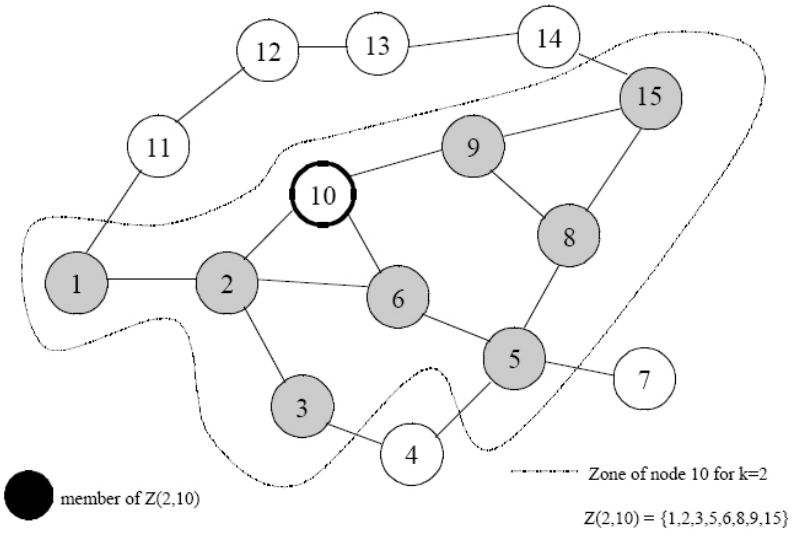

Figura 2.16: Esempio di zona in ZRP.

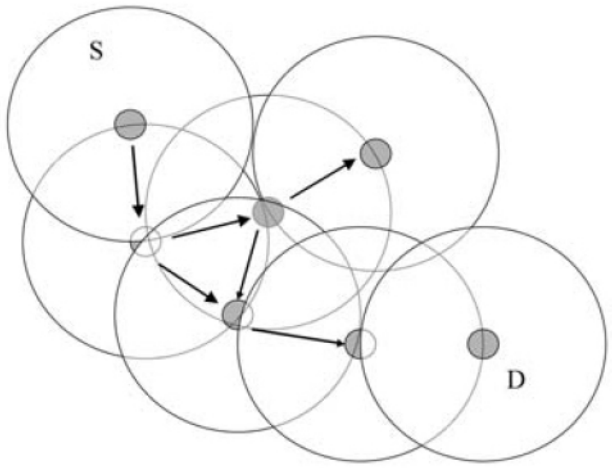

Figura 2.17: Propagazione di una query mediante bordercast in ZRP.

L'inoltro dei pacchetti attraverso un cammino interzona adotta una modifica al source routing. Un cammino di routing contiene solamente i nodi periferici che sono stati attraversati quindi l'inoltro attraverso tali nodi avviene in modalità *table driven* dal momento che la distanza tra questi nodi è k.

La fase di route maintenance è responsabile del mantenimento dei cammini interzona. Viene inoltre suggerito l'uso di una procedura di riparazione locale mirata a riparare i link

rotti da una ricerca di cammino, al fine di ridurre la necessità di route discovery globali. Un miglioramento di ZRP, il *Distributed Dynamic Routing algorithm* (DDR), è stato proposto in [NLB00]. Tale protocollo è basato sulla costruzione di un insieme di zone dinamiche non sovrapposte.

2.2.5 Routing unicast: protocolli position-based

In questo tipo di protocolli i nodi si scambiano informazioni di tempo e posizione, ottenute ad esempio tramite GPS. Quando un nodo deve instradare un pacchetto lo inoltra al nodo più vicino alla destinazione. Il protocollo più importante tra questi è il LAR.

2.2.5.1 LAR: Location Aided Routing

Quando sono disponibili informazioni di posizione e velocità è possibile ottenere un servizio di routing decisamente più efficiente dei protocolli precedentemente visti (vedi [KV00]).

Il *Location Aided Routing* (LAR) è un protocollo reattivo che sfrutta informazioni di posizione e velocità di un nodo, precedentemente acquisite, per stimarne la posizione attuale e "orientare" la fase di Route Discovery. LAR ricorre al flooding solo se non vi sono dati precedenti di posizione: in questo modo diminuisce il traffico dovuto ai pacchetti di controllo del protocollo. Ogni nodo rileva e memorizza continuamente la posizione e la velocità dei propri vicini oltre al tempo al quale l'acquisizione è avvenuta. L'inoltro dei pacchetti ordinari avviene in modalità table-driven. Quando un nodo destinazione D non è presente nella tabella del nodo sorgente S, LAR inizia la fase di discovery (vedi Figura 2.18).

Se precedentemente il nodo D era stato vicino di S allora questo conoscerà i dati di posizione e velocità di D al tempo dell'acquisizione t_0. In base a questi parametri S calcola la *expected zone*, o zona attesa, entro la quale è probabile che il nodo D si trovi al tempo t_1. La *zona attesa* è una sfera centrata sulla posizione del nodo al tempo t_0 e il cui raggio dipende dalla velocità che il nodo aveva in t_0.

LAR prevede due diversi schemi di discovery. Nel primo la zona in cui la query viene propagata, detta *request zone*, è il più piccolo parallelepipedo che include sia S che la zona attesa di D: la query non può uscire dalla request zone durante il floodinge. Le informazioni geometriche della request zone stessa sono incluse nella query in modo che i nodi che la ricevano possano determinare se propagarla o meno. Nel secondo schema ogni query contiene la posizione stimata di D e la distanza stimata $Dist$ del nodo che ritrasmette la query dal nodo D stesso. Sia D_i la distanza di i dalla destinazione D. Quando il nodo sorgente S crea il pacchetto di query pone $Dist = D_S$. Se un nodo intermedio i ritrasmette la query pone $Dist = D_i$. Quando un nodo j riceve la query per D dal nodo i la ritrasmette solo se:

$$Dist + \delta > D_j \tag{2.2}$$

cioè solo se j è più vicino a i almeno della lunghezza δ. Tale valore è necessario per tenere conto degli errori di posizione, soprattutto se i e j sono vicini.

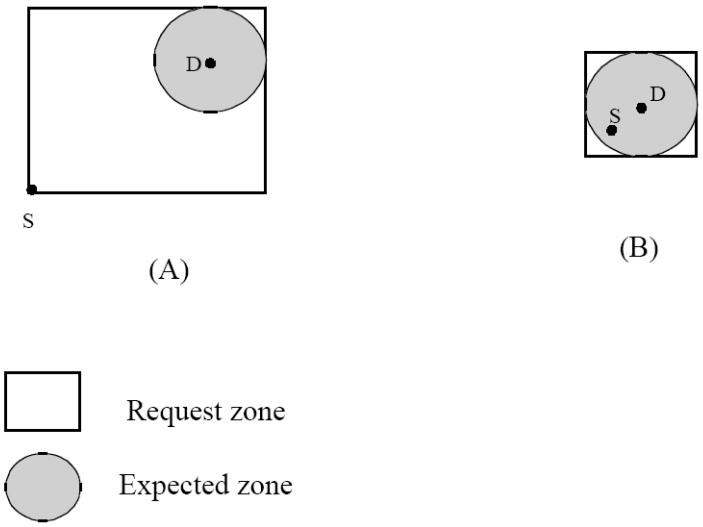

Figura 2.18: Esempi di request e routing zone: (a)S fuori della expected zone di D; (b)S dentro la expected zone di D.

2.2.6 Routing multicast

Con il termine multicast si intende la trasmissione di pacchetti ad un gruppo di zero o più host identificati da un singolo indirizzo di destinazione. Questa tecnica di trasmissione è molto utile nelle applicazioni *group-oriented*, quando si vuole inviare uno stesso messaggio a più membri di un gruppo contemporaneamente.

Nell'ambiente altamente dinamico delle MANET i tradizionali approcci multicast per reti cablate non sono adatti a causa del movimento casuale dei nodi e dei cambiamenti frequenti e impredicibili di topologia. Inoltre, si hanno delle limitazioni sulla banda e sulla durata della batteria. Tutti questi vincoli rendono il multicasting in ambito ad hoc estremamente difficile. Le soluzioni generali proposte per risolvere questi problemi sono:

- Evitare il flooding globale e gli advertisement.
- Costruire dinamicamente i cammini e mantenere l'appartenenza ai gruppi.

In ambito MANET ci sono tre categorie base di algoritmi multicast. Un approccio *"naive"* che prevede di inviare semplicemente in rete pacchetti via flooding. Un nodo quindi, alla ricezione di un pacchetto, lo ritrasmette a tutti i nodi vicini e così via per tutti i nodi della rete. Questo comportamento pertanto agisce come una reazione a catena che cresce esponenzialmente. Un secondo approccio *proattivo* prevede di prestabilire i cammini verso tutte le possibili destinazioni e di memorizzare queste informazioni nelle routing table. Per mantenere un database aggiornato, le informazioni di routing vengono periodicamente distribuite attraverso la rete. Il terzo ed ultimo approccio prevede di creare i cammini *on demand*. L'idea si basa su un meccanismo di richiesta-risposta o di multicast reattivo. Nella fase di query un nodo esplora l'ambiente e una volta che la query ha raggiunto la destinazione, inizia la fase di risposta e si stabilisce il cammino.

2.2.6.1 ODMRP: On Demand Multicast Routing Protocol

L'*On Demand Multicast Routing Protocol* (ODMRP) è un protocollo mesh-based ed utilizza il concetto di forwarding di gruppo, cioè solo un subset di nodi inoltra i pacchetti in multicast (vedi[BLSG00]). ODMRP utilizza un approccio soft-state per mantenere il multicast tra i membri del gruppo e non è richiesto alcun messaggio esplicito di controllo per lasciare il gruppo.

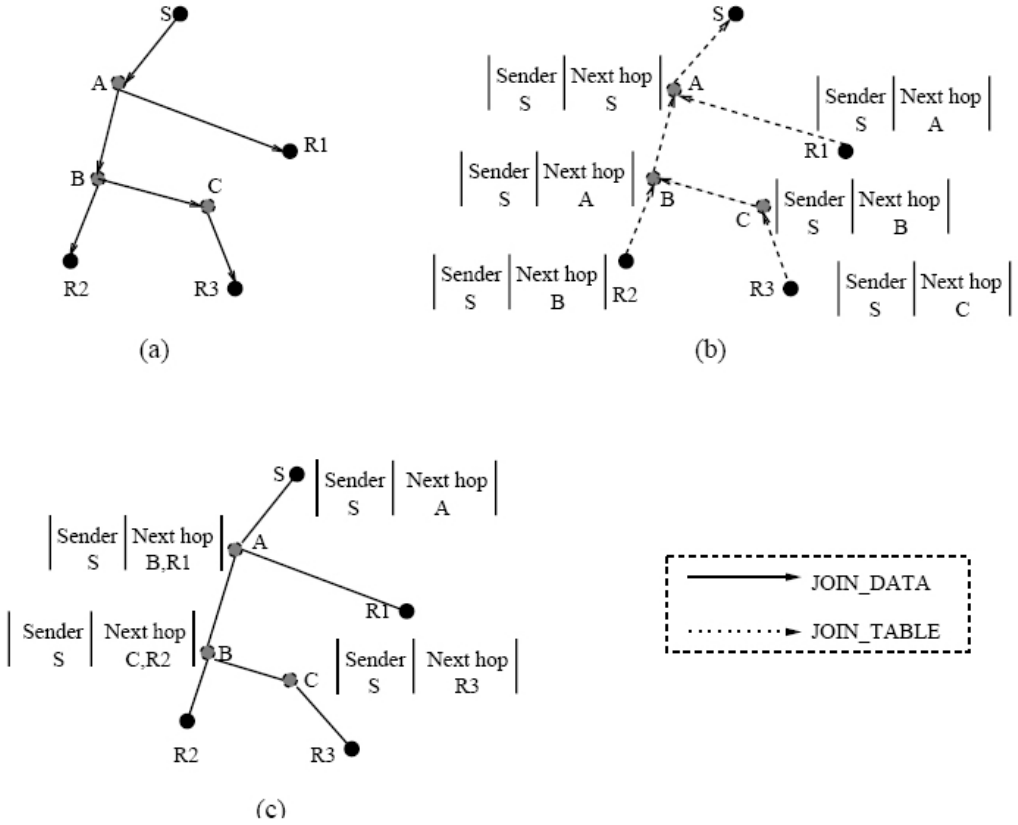

Figura 2.19: Esempio di ODMRP: (a)Propagazione di pacchetti JOIN DATA; (b)Propagazione di pacchetti JOIN TABLE; (c)Tavole multicast finali.

In ODMRP l'appartenenza al gruppo ed i cammini multicast sono stabiliti ed aggiornati dalla sorgente on demand. Considerando la Figura 2.19a, la sorgente S, che desidera inviare un pacchetto in multicast ad un gruppo ma non ha un cammino verso il gruppo, invierà in broadcast un pacchetto di controllo *JOIN DATA* all'intera rete. Questo pacchetto è periodicamente inviato in broadcast per rinfrescare le informazioni di membership ed aggiornare i cammini. Quando il pacchetto JOIN DATA raggiunge un ricevitore multicast, quest'ultimo crea ed invia in broadcast a tutti i suoi vicini un pacchetto *JOIN TABLE*. Quando un nodo riceve questo pacchetto esso controlla se l'identificatore del nodo next-hop di una delle entry della tavola coincide con il suo identificatore. Se lo è, il nodo capisce di trovarsi sul cammino che va dalla sorgente alla destinazione, cioè fa parte del forwarding group, e quindi setta un flag chiamato *FG FLAG* (Forwarding Group Flag). Esso poi invia

in broadcast la join table costruita con le entry che combaciano. Il campo identificatore del nodo next-hop viene riempito estraendo informazioni dalle proprie routing table. In questo modo ogni membro appartenente al forward group propaga la JOIN TABLE fino al raggiungimento della sorgente multicast S attraverso il cammino selezionato (che risulta essere anche il più corto). La Figura 2.19b mostra come questi pacchetti vengono inoltrati fino alla sorgente. Alla ricezione di una JOIN TABLE un nodo inoltre deve costruire la sua tavola multicast per il futuro forwarding di pacchetti. Per esempio quando B riceve la JOIN TABLE di R_2, esso aggiungerà R_2 come suo next-hop. Nell'ultima Figura 2.19c viene mostrata la tavola multicast finale.

A seguito del processo di forwarding e della costruzione del cammino, le sorgenti possono inviare in multicast i pacchetti verso i ricevitori del gruppo attraverso i cammini selezionati. Prima di inviare i pacchetti la sorgente invia anche periodicamente dei pacchetti JOIN DATA per rinfrescare il gruppo di forwarding ed i cammini. Quando un nodo riceve un pacchetto multicast, lo inoltra solo nel caso non sia un duplicato ed il settaggio del flag FG FLAG per il gruppo multicast non sia scaduto. Questa procedura minimizza l'overhead di traffico e previene l'invio di pacchetti attraverso cammini invalidi.

In ODMRP, non vi è necessità di invio di espliciti pacchetti di controllo per il join o leave da un gruppo. Se una sorgente multicast vuole lasciare un gruppo deve semplicemente fermare l'invio di pacchetti JOIN DATA dal momento che non ha pacchetti dati multicast da inviare a nessun gruppo. Se un ricevitore invece non vuole più ricevere pacchetti di un determinato gruppo esso non invierà il messaggio di risposta al join al gruppo.

2.2.6.2 MAODV: Multicast Ad hoc On demand Distance Vector

Il protocollo *Multicast Ad hoc On demand Distance Vector* (MAODV) scopre cammini multicast on demand usando un meccanismo di Route discovery broadcast (vedi [RP99]). Un nodo mobile origina un messaggio route request (RREQ) quando desidera unirsi ad un gruppo multicast oppure quando possiede dati da inviare ad un gruppo multicast ma non possiede alcun cammino verso di esso. La Figura 2.20a mostra la propagazione di un pacchetto RREQ (indicato dalle frecce orientate) che parte dal nodo sorgente S. Se la RREQ è una richiesta di join qualche nodo del gruppo potrebbe rispondere, mentre se non lo è, nessun nodo con un cammino abbastanza recente verso il gruppo (basandosi su numeri di sequenza) potrebbe rispondere. Se un nodo intermedio riceve una richiesta di join per un gruppo del quale non è membro, oppure riceve una RREQ ma non ha cammini verso quel gruppo, esso rinvia la RREQ verso i suoi vicini.

Se un nodo riceve un join RREQ per un gruppo multicast, esso potrebbe rispondere se è un membro del gruppo ed il suo numero di sequenza per il gruppo è maggiore di quello contenuto nel pacchetto. Il nodo che risponde alla richiesta aggiorna il suo cammino e le tabelle di routing multicast mettendo le informazioni next-hop del nodo richiedente nelle tabelle, ed in seguito invia in unicast un pacchetto di risposta RREP al richiedente (indicato in Figura 2.20b dalla freccia tratteggiata).

Quando una sorgente invia in broadcast un RREQ per un gruppo multicast, spesso riceve più di una risposta. Il nodo mantiene per un certo periodo di tempo i cammini ricevuti con il più grande numero di sequenza e con il più basso numero di passi verso

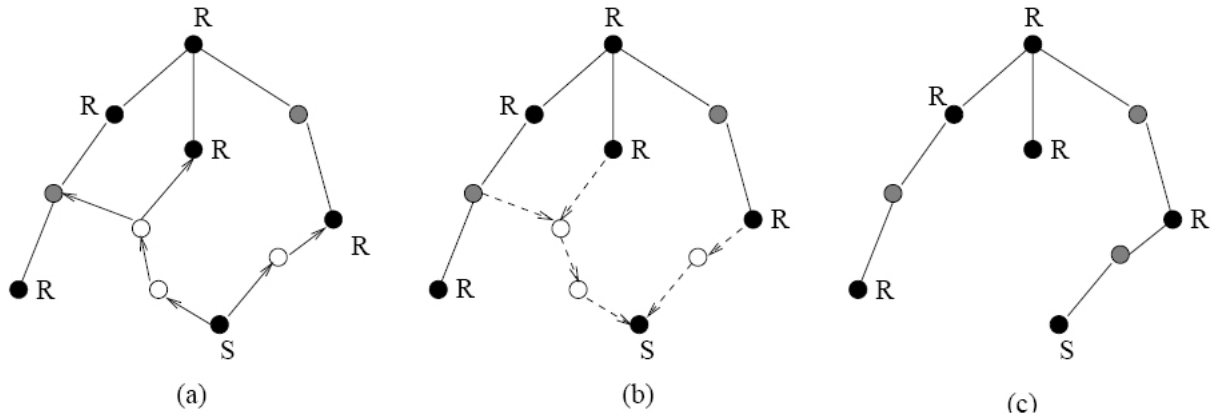

Figura 2.20: Esempio di MAODV: (a)Propagazione di un pacchetto RREQ; (b)Propagazione di un pacchetto RREP; (c)Albero multicast finale.

un membro del gruppo multicast. Al termine di questo periodo esso attiva il next-hop selezionato nella sua tavola multicast inviandogli in unicast un messaggio di attivazione. Il next-hop alla ricezione di questo messaggio, attiva le entry per la sorgente nelle sue tavole multicast. Se questo nodo fa parte del gruppo non propaga ulteriormente il messaggio, mentre se non lo è, esso avrà ricevuto uno o più RREP dai suoi vicini. Esso manterrà quindi il miglior next-hop per il suo cammino verso il gruppo e lo attiva inviandogli in unicast un messaggio di attivazione. Questo processo continua fino a che viene raggiunto il nodo che ha originato la RREP. Il messaggio di attivazione assicura che l'albero multicast non possiede cammini multipli. Come indicato in Figura 2.20c i nodi propagano pacchetti solo lungo i cammini attivi dell'albero multicast.

Il primo membro del gruppo multicast diventa leader di quel gruppo. Esso è il responsabile del mantenimento dei numeri di sequenza del gruppo e dell'invio di tali numeri ai membri del gruppo. Questo viene eseguito mediante messaggi di HELLO che contengono l'indirizzo IP multicast ed il numero di sequenza del gruppo. I nodi poi usano tali informazioni per aggiornare le proprie tabelle di routing.

Capitolo 3

Il middleware AGAPE

La crescente disponibilità di connettività wireless in ambienti domestici e lavorativi, la grande diffusione di dispositivi portatili e la nascita di nuovi tipi di reti come le MANET, hanno creato nuove opportunità per applicazioni di tipo collaborativo, che permettono cioè una collaborazione improvvisa tra utenti che condividono interessi comuni.

Tuttavia, la fornitura di servizi collaborativi in ambienti privi di un'infrastruttura di rete fissa e con costanti cambiamenti delle condizioni operative, innalza tutta una serie di problemi e rende necessario riprogettare i tradizionali sistemi di gestione di gruppo. Ora, dato che la topologia non è predefinita e fissa, non è più possibile fare assunzioni a priori sullo stato e sulla disponibilità dei membri di un gruppo, che appaiono e scompaiono in maniera impredicibile e cambiano frequentemente il loro punto di accesso ai servizi collaborativi. In questo scenario eventi come il roaming, le disconnessioni e le partizioni della rete sono molto comuni ed inoltre, la scarsa ampiezza di banda fornita dalle tecnologie di rete wireless, rende la congestione un evento molto frequente.

In uno scenario di questo tipo sono richieste nuove soluzioni in grado di risolvere i problemi della fornitura di servizi collaborativi. Il middleware AGAPE (*Allocation and Group Aware Pervasive Environment*) si prefigge l'obiettivo di rappresentare una soluzione a questo tipo di problemi.

Il middleware AGAPE fornisce un set di servizi di supporto per il progetto e l'impiego di applicazioni collaborative in ambito MANET. AGAPE adotta la metafora *gruppo* per abilitare i servizi e la condivisione delle risorse tra entità. In particolare AGAPE permette ad utenti vicini con interessi comuni di creare ed unirsi a gruppi. Una caratteristica peculiare di AGAPE è infatti quella di essere un middleware *location-aware*, in quanto gestisce i gruppi in base alla locazione degli utenti e dei dispositivi. Questo viene realizzato basando la gestione sull'assunzione che, utenti situati nella stessa *località*, collaboreranno più facilmente di utenti situati in località differenti.

AGAPE inoltre fornisce tutte le funzionalità per propagare a livello applicazione la visibilità *context-dependent* dei gruppi e dei membri presenti in località. AGAPE infatti è progettato in modo tale da ridurre la necessità di avere delle *viste* globali di tutti gli utenti presenti nella rete, mentre realizza delle viste dipendenti dalla località che sono rapidamente disponibili anche in caso di partizionamenti della rete. Queste viste conterranno tutte le entità presenti in una determinata località e dunque, per avere una vista globale dell'intera

rete, è necessario unire le viste di tutte le località esistenti.

Nel nuovo scenario delineato dalle reti MANET, si innalza l'ulteriore problema della fornitura di un servizio di *naming* affidabile. Nelle reti ad hoc risulta infatti impossibile garantire nomi univoci mediante un sistema di nomi centralizzato oppure mediante l'accordo globale di tutti i membri della rete, il che rende le soluzioni tradizionali non in grado di supportare la comunicazione in questo scenario. AGAPE propone di sfruttare la visibilità delle informazioni di contesto, come la posizione fisica dei dispositivi, le preferenze del gruppo e lo stato della rete, per mantenere e organizzare i membri del gruppo e permettere la comunicazione. In pratica, l'approccio *context-aware* di AGAPE, permette di identificare le entità comunicanti sulla base di informazioni di locazione e di *profilo* piuttosto che sulla base del loro nome.

AGAPE infine, esegue una distinzione fondamentale tra i dispositivi sulla base delle differenti capacità computazionali. All'interno del middleware solamente i dispositivi con ricche risorse di calcolo supportano le operazioni di gestione del gruppo, come la creazione di nuovi gruppi e la distribuzione delle viste di contesto, mentre i dispositivi con limitate capacità sfruttano i servizi di gestione forniti dai dispositivi più ricchi, e si limitano ad unirsi a gruppi ed a collaborare con gli altri membri co-locati[1].

3.1 Modello di gruppo

AGAPE è un framework di gestione di gruppi context-aware che permette alle entità di creare, unirsi e lasciare uno specifico gruppo e di ricevere la vista dei membri del gruppo co-locati.

> Con il termine *gruppo* si indica un'insieme di utenti e dispositivi che collaborano al fine di raggiungere obiettivi comuni o per condividere risorse di interesse.
>
> Con il termine *vista* invece si indica l'insieme di tutti i membri appartenenti allo stesso gruppo e posizionati all'interno della medesima località, ossia co-locati.

L'insieme dei membri che compone un gruppo di AGAPE non può essere determinato a priori, in quanto esso varia dinamicamente a causa delle operazioni di join e leave dei membri al gruppo. La mobilità degli utenti e le temporanee disconnessioni dei dispositivi causano cambiamenti solo alle viste ricevute dagli utenti ma non all'appartenenza al gruppo.

Quando un utente entra, lascia oppure si disconnette da una località di rete, AGAPE riporta i cambiamenti mediante la vista a tutti i membri co-locati appartenenti al gruppo. Quando un membro invece si muove da una località ad un'altra, AGAPE si incarica di fornire al membro in movimento, la vista della nuova località.

Le viste dipendenti dal contesto, chiamate anche *context-view*, hanno il grande vantaggio di essere rapidamente e direttamente disponibili anche in caso di partizionamento della rete. Se avviene un partizionamento di rete infatti, almeno uno degli utenti vicini presenti nella vista e che appartiene alla stessa partizione può continuare a collaborare. E' possibile

[1] Il termine indica i membri localizzati all'interno della stessa località.

inoltre fornire, quando necessario, una vista globale ottenuta mediante il merge di tutte le viste ricevute, coordinando i componenti del middleware di ogni località. Tuttavia, è giusto evidenziare che il modello di gruppo di AGAPE rilassa la necessità di possedere una vista globalmente consistente. Infatti le frequenti partizioni di rete , le disconnessioni dei dispositivi e i cambiamenti di locazione rendono inefficiente ed anche inutile assicurare una globale consistenza della vista di tutti i membri del gruppo. Il modello di lavoro è dunque, a causa dell'alta dinamicità delle reti MANET, di tipo *best-effort*.

Ogni gruppo AGAPE è caratterizzato da un identificatore unico di gruppo (GID) e da un profilo specifico che definisce gli interessi, le preferenze e le attività desiderate che dovrebbero essere concordi a tutti i membri del gruppo. Se il profilo di un membro che intende unirsi ad un gruppo combacia con quello del gruppo, tale utente è ammesso a far parte del gruppo. Ciascun membro appartenente ad un gruppo è caratterizzato da un identificatore personale (PID), il quale è statisticamente unico entro il gruppo, e da un profilo che esprime gli attributi e le caratteristiche simili a quelle del gruppo come gli interessi, i compiti e gli obiettivi.

In Figura 3.1 viene mostrato il modello di gruppo di AGAPE. Esso indica due entità principali: le *Managed Entity* (ME) e le *Locality Manager Entity* (LME). Un'entità ME è un membro di un gruppo che utilizza i servizi di supporto alla collaborazione forniti da AGAPE. Un'entità LME invece è un membro di un gruppo che non solo collabora, ma supporta anche le operazioni di gestione di se stesso e degli ME: un LME promuove a tempo di esecuzione la creazione di un nuovo gruppo, permette alle entità di unirsi o lasciare gruppi e di mantenere una lista aggiornata dei membri co-locati. In ogni momento, LME ed ME, possono unirsi o lasciare gruppi di loro interesse. Tuttavia, in ogni istante un ME può appartenere al massimo ad un singolo gruppo, ma in differenti istanti di tempo anche a più gruppi cambiando la propria appartenenza mediante operazioni di join e leave. Un LME invece può appartenere in ogni momento anche a più gruppi.

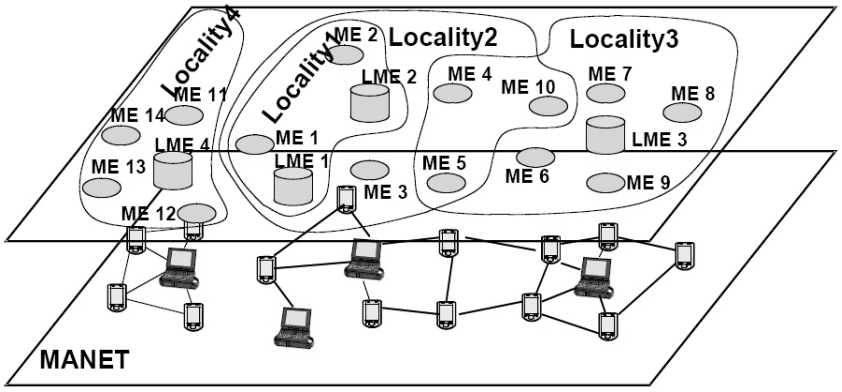

Figura 3.1: Modello di gruppo di AGAPE (Località a 2 hop).

I gruppi AGAPE sono partizionati in differenti località. Ogni LME supporta la gestione delle operazioni di gestione di gruppo solamente per i membri posizionati entro la sua località.

> Una *località* è definita come l'insieme di tutte le entità AGAPE i cui dispositivi sono connessi al dispositivo dell'LME da un cammino di lunghezza massima di h hop. L'LME rappresenta il centro di una località identificata dal suo PID e dalle sue coordinate geografiche. Il valore h esprime il massimo raggio di una località misurato in numero di hop. Questo valore viene scelto sulla base dello scenario applicativo.

Ogni membro di un gruppo riceve una vista (vista *context-dependent*) che contiene la lista dei soli membri appartenenti a tale gruppo e localizzati entro i limiti della località dell'LME gestore. In particolare, ciascuna vista context-dependent include la coppia PID/GID ed il livello di batteria residuo dell'LME che l'ha generata ed è composta da una lista di entry che associa ogni membro del gruppo con le informazioni di profilo dell'utente e del dispositivo. Quando un membro si connette o disconnette dalla rete oppure quando cambia il dispositivo di accesso o il proprio profilo, AGAPE riporta i cambiamenti della vista a tutti i membri del gruppo in località utilizzando le funzionalità broadcast supportate.

La Figura 3.1 mostra un caso di località con $h = 2$. Ogni località della figura contiene sia LME sia ME, e come si può vedere più località si possono anche sovrapporre a causa della vicinanza fisica di differenti LME. Gli ME possono muoversi liberamente e possono appartenere anche a più di una località nello stesso momento (vedi il membro ME5). In questo caso, un membro appartenente a più di una località, riceve le viste da parte di più LME e dunque esso ottiene la vista globale del gruppo eseguendo il merge delle viste ricevute. Può anche accadere inoltre che due LME definiscono la stessa località, come nel caso in cui due LME sono allocati ad un hop di distanza.

3.2 Architettura

I servizi per la gestione dei gruppi di AGAPE sono organizzati su due livelli logici situati al di sopra della Java Virtual Machine. Questi due livelli sono denominati rispettivamente *Basic Service Layer*, che fornisce i servizi di base per realizzare la comunicazione, e *Group Management Layer*, che fornisce i servizi di gestione dei gruppi.

La Figura 3.2 mostra l'architettura interna del middleware AGAPE.

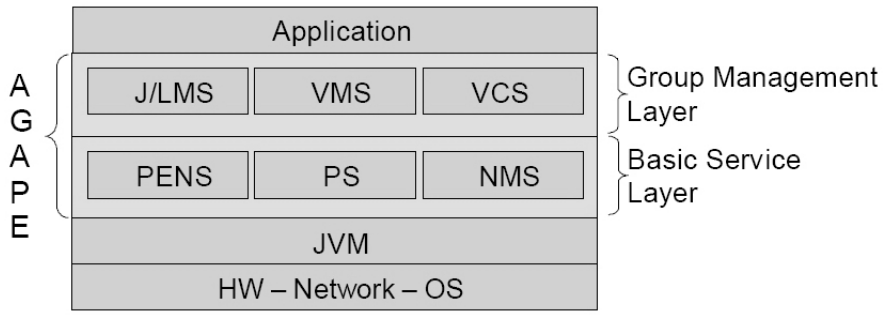

Figura 3.2: Architettura di AGAPE.

3.2.1 Basic Service Layer

Il *Basic Service Layer* include delle funzionalità di basso livello per permettere ai dispositivi di accedere alla rete MANET, per il supporto al naming, per attivare il discovery dei gruppi e dei membri ed infine per monitorare lo stato sulla disponibilità dei membri.

Il *Network Manager Service* (NMS) fornisce il necessario supporto ai membri del gruppo di scambiarsi messaggi attraverso la MANET. NMS implementa i pattern di comunicazione di tipo punto-punto e punto-multipunto: il supporto alla comunicazione punto-punto dell'NMS permette di inviare messaggi ad un host identificato da un indirizzo IP conosciuto, mentre il supporto alla comunicazione punto-multipunto permette di inviare uno stesso messaggio in broadcast a diverse entità AGAPE.

La comunicazione multipunto fornita dall'NMS estende i protocolli di *flooding*[2] probabilistici limitando la disseminazione dei messaggi entro un determinato numero di hop, introducendo un meccanismo basato sul concetto del *Time-to-Live* (TTL). In particolare, NMS inserisce in ogni messaggio un campo TTL che viene decrementato ad ogni hop. Possono essere inoltrati solamente i messaggi che hanno TTL diverso da zero, altrimenti vengono eliminati. E' da notare però che la natura probabilistica del protocollo di flooding, fornisce un approccio di tipo best-effort per la consegna dei messaggi a tutti i vicini.

L'NMS inoltre fornisce un accodamento locale dei messaggi entranti ed uscenti che vengono gestiti con politica FIFO.

Il *Proximity Service* (PS) permette ai membri dei gruppi, sia LME che ME, di pubblicare a tutti i membri presenti la loro disponibilità in una località.

Ad intervalli di tempo regolari il servizio richiede all'NMS di inviare in broadcast sulla MANET un messaggio chiamato *beacon*, che include la coppia GID/PID dell'entità, il ruolo dell'entità (LME o ME) all'interno del gruppo ed il proprio indirizzo IP. L'intervallo di tempo tra due successivi beacon può essere settato in accordo sia alle specifiche dell'applicazione sia alla mobilità dei nodi. In particolare, la frequenza di trasmissione dei beacon, dovrebbe incrementare direttamente all'aumentare della mobilità degli utenti. E' da notare però che il PS si basa sull'NMS per la disseminazione dei beacon, e pertanto c'è solo una garanzia probabilistica che tutti i membri in una località ricevano i beacon disseminati. La probabilità di ricezione dei beacon dipende dalla topologia di rete e dalla probabilità di ritrasmissione dei messaggi del protocollo di flooding probabilistico dell'NMS.

Il *Proximity Enabled Naming Service* (PENS) genera statisticamente in modo casuale gli identificatori univoci per i gruppi (GID) e gli identificatori personali per le entità (PID), sfruttando un approccio di naming simile ad uno proposto in ambito peer-to-peer (vedi [R01]).

Oltre a questa importante funzione, il PENS rileva i beacon in arrivo dagli altri LME/ME co-locati e costruisce una tavola contenente i membri correntemente disponibili in località. Ciascuna entry della tavola contiene la coppia GID/PID della corrispondente entità AGAPE, il suo ruolo ed il suo indirizzo IP. Inoltre, ad ogni entry della tavola è associato un timestamp. Se il PENS non riceve beacon da un membro per un certo intervallo di

[2]Sono dei protocolli di routing probabilistici per la disseminazione di messaggi in broadcast a tutti i nodi di una rete ed appartengono alla famiglia dei protocolli di GOSSIP (vedi [OV01]).

tempo, la corrispondente entry della tavola viene rimossa e il dispositivo dell'entità viene considerato disconnesso.

3.2.2 Group Management Layer

Il *Group Management Layer* fornisce i servizi richiesti per scoprire, creare e cancellare i gruppi, oltre a permettere tutte le operazioni di join/leave e di invio/ricezione delle viste.

Il *Join/Leave Manager Service* (J/LMS) permette alle entità AGAPE (LME ed ME) di creare, di scoprire, di unirsi e di lasciare gruppi. Gli LME utilizzano il J/LMS per promuovere dinamicamente la formazione di nuovi gruppi. La promozione di un nuovo gruppo richiede la definizione di un profilo di gruppo al livello applicazione. IL J/LMS si coordina poi con il PENS per ottenere l'identificatore univoco del gruppo GID ed il proprio identificatore personale PID all'interno del gruppo. Si coordina infine con il PS per avvertire il vicinato della disponibilità di un nuovo gruppo e con il VMS (vedi oltre) per inizializzare una nuova vista di gruppo. Sia gli LME che ME possono utilizzare il J/LMS per scoprire ed unirsi a gruppi di interesse. La fase di discovery del gruppo richiede al J/LMS di coordinarsi con il PENS per ottenere la lista dei gruppi disponibili in località. Le entità AGAPE che non appartengono a nessun gruppo possono utilizzare i servizi offerti dal J/LMS per tentare di unirsi ad un gruppo di interesse disponibile. Al fine di unirsi ad un nuovo gruppo l'entità deve specificare il profilo di gruppo desiderato a livello applicazione. Non appena un nuovo membro si unisce ad un gruppo il J/LMS ritorna la sua coppia GID/PID (coordinandosi col PENS) ed il profilo completo del gruppo.

Il J/LMS infine permette ai membri, sia LME che ME, di lasciare un gruppo precedentemente acceduto, specificando il suo GID. Il J/LMS quindi, si coordina con gli altri componenti di AGAPE, come ad esempio il PS ed il VMS al fine di terminare, da parte del PS, l'invio dei beacon delle entità non più presenti e delegando al VMS la cancellazione di ogni informazione relativa ai membri non più appartenenti al gruppo, come la coppia GID/PID ed il profilo.

Il *View Manager Service* (VMS) è il componente responsabile della gestione delle viste. Esso permette agli LME di creare e disseminare le viste di gruppo ai membri del relativo gruppo gestito ad intervalli di tempo regolari, mentre agli ME di ricevere ed utilizzare tali viste. In seguito si vedrà come la soluzione proposta per questo componente faccia cadere quest'ultima specifica, proponendo una gestione adattativa del gruppo e dunque un'invio delle viste non più ad intervalli di tempo regolari ma dipendente dal contesto.

Il VMS si coordina con il PENS per ottenere la notifica di eventi, come l'arrivo, la partenza o la disconnessione di un'entità del gruppo. In particolare, questi eventi innescano sul VMS l'aggiornamento della vista di gruppo relativa, inserendo o rimuovendo le nuove o vecchie entità membro. Per di più, le viste possono cambiare anche a causa di variazioni del profilo utente e/o del dispositivo di accesso, ma queste ultime accadono più raramente. Quando un'entità, sia LME che ME, cambia le proprie informazioni di profilo, essa delega il VMS di aggiornare opportunamente la vista del gruppo con le nuove informazioni. In particolare l'entità fornisce al VMS il nuovo profilo utente da essere incluso all'interno della relativa entry nella vista.

E' da notare che anche il VMS garantisce una consistenza probabilistica delle viste

di gruppo. Questo a causa del fatto che il VMS, al fine di rilevare la disponibilità dei membri del gruppo co-locati e quindi costruire le viste, si basa sul protocollo di flooding probabilistico per la disseminazione dei messaggi implementato al livello inferiore nell'NMS.

Il *View Controlling Service* (VCS) permette agli LME di decidere se e quando distribuire le viste context-dependent. Il VCS aiuta a ridurre la propagazione delle viste non necessarie, per esempio quando più LME appartenenti allo stesso gruppo e che definiscono la stessa località, tentino di disseminare la stessa vista a tutti i membri del gruppo. Il VCS permette pertanto di preservare la banda e quindi la batteria dei dispositivi evitando l'invio di viste superflue ai membri del gruppo. In particolare, ogni volta che un LME tenti di disseminare la sua vista, il suo VMS richiede il permesso al VCS. Il VCS quindi confronta le entry delle viste ricevute dagli LME co-locati e, se la vista che si vuole inviare non contiene le stesse entry, il VCS concede il permesso al VMS di propagare la vista. Se invece le viste ricevute sono identiche, il VCS concede ad un solo LME di propagare le viste, in quanto le altre sarebbero superflue. L'LME scelto per inviare la vista è quello avente il livello di batteria residuo più alto di un predefinito livello limite. Se tutti gli LME hanno lo stesso livello di batteria il VCS seleziona quello avente il PID più alto.

3.3 Dettagli implementativi

Esistono due differenti implementazioni del middleware AGAPE. Una di queste supporta le operazioni di gestione degli LME e tutti i servizi di AGAPE. Questa release è adatta e rivolta a quei dispositivi portatili ad esempio come i laptop, caratterizzati da risorse con ricche capacità computazionali (grande quantità di memoria, file system, cpu potente e batterie a lunga durata). L'altra versione disponibile supporta invece le operazioni degli ME ed include solamente un sottoinsieme dei servizi supportati da AGAPE, come il lato client del J/LMS e del VMS insieme al completo strato inferiore del middleware contenente il PS, il PENS e l'NMS.

In questa sezione verranno esposti alcuni dettagli di implementazione e di funzionamento di alcuni servizi di gestione di gruppo che supportano la formazione dinamica e la gestione di gruppi ad hoc, come il J/LMS, il VMS ed il VCS. Il principio che ha guidato l'implementazione dei servizi di AGAPE è l'ottimizzazione delle risorse, al fine di tenere in conto i limiti dei dispositivi portatili odierni. In particolare verrà dettagliato in che modo lavorano questi tre componenti di AGAPE in differenti scenari operativi, come il caso di LME co-locati appartenenti allo stesso gruppo e che definiscono località sovrapposte o innestate, ed il caso di località partizionate. Tutti questi casi, in situazione reale, sono scenari abbastanza comuni, come nel caso di uno scenario di soccorso con un largo numero di utenti molto vicini e con dispositivi eterogenei.

La seguente figura mostra tre scenari di funzionamento che saranno descritti separatamente: la 3.3a mostra due località innestate, la 3.3b due località parzialmente sovrapposte e la 3.3c due località partizionate.

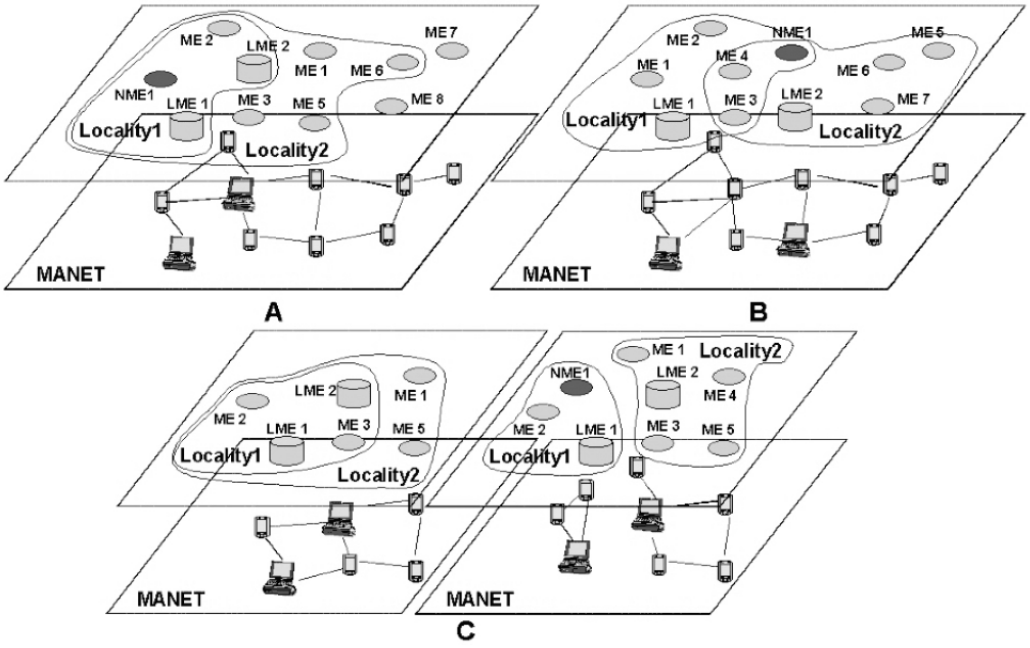

Figura 3.3: Funzionamento di AGAPE in tre differenti scenari operativi.

3.3.1 Località innestate

La Figura 3.3a mostra una situazione operativa in cui due LME (LME1 ed LME2) appartengono allo stesso gruppo e pertanto gestiscono lo stesso GID e definiscono due località innestate, in cui la Località 1 è tutta contenuta all'interno della Località 2. In questo scenario come nei successivi, i dispositivi aventi ruolo di LME implementano tutti i servizi di AGAPE, mentre gli ME solamente un sottoinsieme di questi, come appena descritto sopra.

Joining. Si consideri il caso di un'entità non appartenente al gruppo e indicata con NME1 che entra in Località 1 e tenta di unirsi ad un gruppo di interesse. Il J/LMS di NME1 ottiene, mediante un'interrogazione al PENS, la lista di tutti gli LME in Località 1, in questo caso LME1 ed LME2, insieme ai loro GID, PID e IP. Il J/LMS di NME1 scansiona quindi la lista ottenuta e riconosce che entrambi gli LME appartengono allo stesso gruppo, in quanto hanno lo stesso GID. Il J/LMS di NME1 seleziona quindi in modo casuale uno dei due LME (ad esempio LME1) per tentare di accedere al gruppo. Il J/LMS invia quindi una richiesta, mediante un messaggio di *Join*, all'LME1, il quale include le preferenze di gruppo (in termini di obiettivi ed attività del gruppo), ed anche il profilo utente. Alla ricezione di tale richiesta, il J/LMS di LME1 controlla se gli attributi del gruppo richiesto combaciano con le preferenze di gruppo espresse dall'utente NME1. Se LME1 appartiene allo stesso gruppo di interesse di NME1, LME1 concede il permesso all'entità di entrare a far parte del gruppo inviando un messaggio di *Acknowledgement* a NME1 che include il PID del nuovo membro, ottenuto dal PENS, e l'intero profilo del gruppo. Inoltre, il J/LMS di LME1 si coordina con il VMS al fine di includere il profilo del nuovo utente nella vista

di gruppo.

E' da notare che anche il protocollo di joining è progettato per salvaguardare le limitate risorse degli ME (cpu, memoria e batteria). Quando possibile infatti, le operazioni di joining più pesanti vengono delegate ai più performanti LME, come l'operazione di *profile matching* per il corretto inserimento di un membro al gruppo. La delega gioca un ruolo importante specialmente quando un singolo ME tenta di unirsi ad un gruppo, e sono disponibili un grande numero di LME.

Propagazione delle viste. Sia LME1 che LME2 mantengono una vista del gruppo aggiornata di tutti i membri correntemente disponibili nella loro località. Ad intervalli di tempo regolari, i VMS di entrambi gli LME propagano le viste a tutti i membri della Località 1 e della Località 2. Prima di propagare queste viste, ogni VMS deve chiedere il permesso per l'invio al proprio VCS. In particolare, prima che NME1 si unisca al gruppo, LME2 ha il permesso di disseminare la sua vista di gruppo a tutti i membri della Località 2 mentre LME1 non può farlo, in quanto la sua vista è un sottoinsieme di quella di LME2. Il VCS di LME1 infatti riconosce questa inclusione osservando la vista ricevuta da LME2.

Quando NME1 si unisce al gruppo (contattando LME1), cambiano i permessi di trasmissione tra gli LME. Quando il VMS di LME1 include il nuovo membro NME1 nella sua vista di gruppo, il VCS di LME1 riconosce che NME1 non è contenuto nella vista ricevuta da LME2 e concede quindi il permesso alla trasmissione. Si ha quindi una fase di transizione in cui entrambi gli LME propagano la propria vista a tutti i membri. In particolare, LME2 riceve la vista da parte di LME1. Il suo VMS scansiona la vista ricevuta e riconosce che un nuovo membro NME1 non fa parte della sua vista di gruppo locale, e verifica se il membro è posizionato in Località 2 coordinandosi col PENS. Come mostrato in figura, NME1 è posizionato in Località 2 e così LME2 include NME1 nella sua vista di gruppo.

Quando LME2 dissemina la sua vista di gruppo a tutti i membri della Località 2, anche LME1 riceve tale vista dato che si trova all'interno della Località 2. Quando il VMS di LME1 tenta di propagare la propria vista di gruppo, esso è prevenuto dal farlo in quanto la vista di gruppo ricevuta da LME2 include ora anche NME1. Si è tornati dunque nella situazione di partenza in cui la vista di LME1 è un sottoinsieme di quella di LME2. Queste considerazioni valgono sia nel caso in cui NME1 sia un ME sia un LME.

In normali condizioni operative tutti i membri del gruppo co-locati in Località 2 (inclusi quelli contenuti in Località 1), ricevono le viste di contesto da LME2. Tuttavia, quando NME1 si unisce al gruppo, tutti i membri in Località 1 subiscono una fase di transizione in cui ricevono le viste da parte di entrambi gli LME. Durante questa fase i membri in Località 1 uniscono le due viste ricevute mediante un'operazione di merge.

La soluzione rappresentata dal VCS permette di salvaguardare memoria sui dispositivi vincolati dalle risorse ma non risolve un degrado di batteria dovuto alla richiesta computazione per le operazioni di merging sulle viste. Tuttavia sono possibili anche differenti soluzioni, come ad esempio mantenere due distinte tavole per la memorizzazione delle viste, ciascuna associata ad un LME. La scelta dipende dal livello di *trade-off* che si vuole ottenere tra memoria, cpu e risparmio energetico.

Roaming. Si consideri ora il caso di un entità membro ME1, appartenente allo stesso gruppo di LME1 ed LME2, che entra in Località 2. Il VMS di ME1 riceve le viste di gruppo aggiornate da parte di LME2 e riconosce di non essere incluso in tale vista. Al

fine di aggiungersi a tale vista di gruppo, il VMS di ME1 invia il proprio profilo completo (ottenuto dal J/LMS) a LME2. Alla ricezione del profilo LME2 include ME1 nella sua vista di gruppo. La stessa procedura avviene anche nel caso in cui il membro a muoversi sia un LME.

3.3.2 Località parzialmente sovrapposte

La Figura 3.3b mostra una situazione operativa in cui LME1 ed LME2 appartenenti allo stesso gruppo definiscono due distinte località, rispettivamente Località 1 e Località 2, che sono parzialmente sovrapposte. I membri localizzati nell'intersezione tra le due località sono inclusi sia nella vista di gruppo di LME1 sia in quella di LME2.

Joining. Si consideri il caso di un'entità non appartenente al gruppo NME1 che entra nell'intersezione tra la Località 1 e la Località 2, e tenti di unirsi ad un gruppo di interesse. Il J/LMS di tale entità NME1 ottiene la lista di tutti gli LME disponibili in località come nel caso precedente, che in questo caso sono LME1 ed LME2. Il J/LMS di NME1 poi scansiona la lista ottenuta e riconosce che sia LME1 che LME2 appartengono allo stesso gruppo. Il membro quindi ne seleziona uno dei due in modo casuale ed accede al gruppo, ad esempio tramite LME1.

Propagazione delle viste. Come mostra la figura, i due LME non sono co-locati, e di conseguenza essi non possono inviarsi la propria vista di gruppo reciprocamente. Il VCS di entrambi infatti permette ai rispettivi VMS di propagare le viste di gruppo a tutti i membri di entrambe le località. E' chiaro dunque che tutti i membri inclusi nell'intersezione tra le due località ricevono le viste di gruppo sia da parte di LME1 sia di LME2. Questi membri poi fondono le viste ricevute da entrambi gli LME mediante operazioni di *merge*: le due viste ricevute vengono unite in una vista unica sulla base dei campi GID e PID inclusi in ogni entry della vista.

Il nuovo membro NME1, al termine del join al gruppo, viene incluso solo all'interno della vista di LME1 della Località 1. Quando NME1 riceve la vista disseminata da LME2 esso riconosce di non essere incluso all'interno di essa anche se appartiene al medesimo gruppo. Questo evento genera l'aggiornamento da parte di LME2 della sua vista di gruppo: NME1 infatti invia il suo profilo completo (ottenuto dal J/LMS) a LME2. Alla ricezione del profilo LME2 include NME1 nella vista di gruppo della Località 2.

Roaming. Si consideri ora un'entità ME3 appartenente allo stesso gruppo di LME1 ed LME2 che entra nell'intersezione tra la Località 1 e la Località 2. Il VMS di ME3 rileva l'arrivo di viste di gruppo da parte sia di LME1 sia di LME2 e riconosce di non essere incluso all'interno di esse. Al fine di aggregarsi al gruppo ME3 invia il suo profilo completo sia ad LME1 che LME2. Alla ricezione del profilo i due LME includono ME3 nella loro vista context-dependent. Simili considerazioni possono essere fatte anche nel caso del roaming di un LME.

3.3.3 Località partizionate

La Figura 3.3c mostra infine una situazione operativa in cui due LME (LME1 ed LME2) appartengono allo stesso gruppo e definiscono due località innestate, con la Località 1 con-

tenuta entro la Località 2. In seguito, a causa del movimento reciproco dei due dispositivi, le due località si partizionano.

Joining. Si consideri il caso di un'entità non appartenente al gruppo NME1 che entra nella località definita da LME1 e tenta di unirsi ad un gruppo di interesse. Il J/LMS di istanza su NME1 ottiene la lista di tutti gli LME disponibili in località, in questo caso solo LME1. Il J/LMS di NME1 tenta perciò di unirsi al gruppo inviando una richiesta ad LME1. Le stesse operazioni avvengono in modo del tutto simile se l'entità si posiziona entro la località definita da LME2.

Propagazione delle viste. Una volta che le località si sono disgiunte gli LME non sono più co-locati e definiscono due località distinte. Inoltre LME1 ed LME2 non possono comunicarsi la propria vista context-dependent in modo reciproco. I VMS di entrambi gli LME inoltrano pertanto le proprie viste di gruppo in maniera indipendente in quanto i rispettivi VCS non sentono la presenza dell'altro.

Roaming. Si consideri ora un'entità membro ME1 appartenente allo stesso gruppo di LME1 ed LME2 e che si muove dalla Località 1 alla Località 2. Da un lato, quando ME1 si porta abbastanza distante da LME1, quest'ultimo non rileva più i beacon inviati da ME1 e di conseguenza lo elimina dalla propria vista di gruppo. Dall'altro lato invece, il VMS di ME1 rileva l'arrivo di viste di gruppo da parte di LME2 e riconosce di non essere incluso. Al fine di aggregarsi a tale vista di gruppo, ME1 invia il proprio profilo completo ad LME2. Alla ricezione del profilo LME2 include ME1 nella propria vista di gruppo. Considerazioni simili possono essere effettuate anche nel caso di roaming di un membro LME.

Capitolo 4

View Manager Service

Le applicazioni collaborative in ambito ad hoc forniscono il supporto necessario per la collaborazione improvvisa tra entità, al fine di raggiungere obiettivi comuni e condividere risorse di interesse. Per poter permettere l'interazione tra le entità, questo tipo di applicazioni basano la loro computazione sul concetto fondamentale di *gruppo*. E' necessario infatti, ai fini della collaborazione, che tutte le entità appartenenti ad uno stesso gruppo abbiano la *visibilità* dei membri vicini con cui poter interoperare. Il requisito fondamentale richiesto pertanto alle applicazioni group-aware, è la fornitura di un servizio di gestione di gruppo che permetta ad ogni membro di fruire della visibilità del proprio vicinato.

Questa necessità si fa ancora più insistente nelle reti ad hoc, a causa della loro elevata dinamicità. In questo ambito infatti la topologia cambia in modo improvviso ed è necessario conoscere in ogni istante quali sono i cambiamenti topologici e quali gli attuali membri vicini. Questo servizio risulta necessario pertanto in ogni middleware o applicativo group-aware orientato alle reti ad hoc, e nel caso di AGAPE, esso viene realizzato dal *View Manager Service* (VMS).

Il VMS fornisce il giusto supporto, ad applicativi di tipo collaborativo, per la corretta visibilità in ogni istante dei membri del gruppo co-locati.

4.1 Requisiti

Lo scopo fondamentale del VMS, è quello di rinfrescare le informazioni di rete dei membri di un gruppo, fornendo loro in ogni istante i dati riguardanti la topologia attuale. Per poter eseguire questa funzione, il VMS basa tutta la sua computazione sui concetti fondamentali di località e vista, già in parte discussi nel Capitolo 3.

> Una *località* è definita come l'insieme di tutte le entità AGAPE i cui dispositivi sono connessi al dispositivo dell'LME da un cammino di lunghezza massima di h hop.
>
> Una *vista* invece, è l'insieme di tutti i membri appartenenti allo stesso gruppo e posizionati all'interno della medesima località, cioè co-locati.

All'interno del framework AGAPE le entità possono coprire diversi ruoli. Si hanno pertanto delle entità *Locality Manger Entity* (LME) che si occupano della gestione delle

viste e dei gruppi, e delle entità *Managed Entity* (ME), che sfruttano il servizio offerto dagli LME. Questa suddivisione di ruoli si riflette anche nei compiti del VMS il quale, nel caso LME, propaga le viste context-dependent a tutti i membri di uno stesso gruppo nella località da esso definita, mentre nel caso ME, si limita a ricevere le viste sfruttando il servizio fornito dagli LME.

La vista propagata dagli LME contiene l'elenco di tutte le entità AGAPE co-locate appartenenti allo stesso gruppo, ed in più, mantiene delle informazioni riguardo al gruppo a cui si riferisce ed al nodo da cui è stata trasmessa. In particolare una vista è caratterizzata dai seguenti parametri:

- **GID**: L'identificatore del gruppo al quale si rifersice.
- **PID**: L'identificatore interno al gruppo dell'LME che ha trasmesso la vista.
- **Address**: L'indirizzo IP dell'LME cha ha trasmesso la vista.
- **Battery**: Il livello di batteria residuo dell'LME cha ha trasmesso la vista.
- **Version**: Un identificatore che rappresenta il numero di versione della vista.

Oltre a tutti questi parametri, una vista contiene una lista di entry ciascuna corrispondente ad un nodo attualmente presente in località e con il quale ogni altro nodo contenuto nella vista può collaborare. Il VMS di un LME infatti trasmette la vista a tutti i nodi presenti in essa. Le informazioni contenute in ogni entry della vista sono quelle strettamente necessarie per poter identificare un nodo all'interno della rete e con cui poter instaurare una collaborazione. Queste informazioni sono:

- **PID**: L'identificatore del membro all'interno del gruppo al quale si riferisce la vista.
- **Battery**: Il livello di batteria residuo del dispositivo del membro.
- **Distance**: La distanza del membro, in numero di hop, dall'LME che ha trasmesso la vista.
- **Profile**: Il profilo del membro contenente tutte le informazioni utente, l'indirizzo IP del dispositivo ed il tipo di interfaccia.

Quando un membro del gruppo intende instaurare una cooperazione con un'altro membro co-locato, esso estrae dalla vista ricevuta la relativa entry e, mediante le informazioni in essa contenute, intraprende la comunicazione con il membro scelto.

A seconda del ruolo svolto dall'entità AGAPE, si distinguono, per il View Manager Service, due diverse funzionalità: una release implementata per le entità LME, che realizza la funzione di creazione delle viste, della loro modifica e della loro propagazione, ed una release implementata per le entità ME, che realizza semplicemente la funzione di ricezione ed utilizzo di tali viste. I casi d'uso del VMS si possono distinguere pertanto in queste due release.

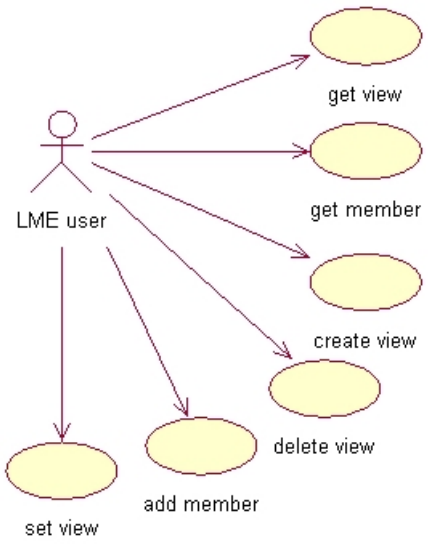

Figura 4.1: Use cases del VMS nel caso LME.

Se l'utilizzatore del VMS è un'attore LME, come indicato in Figura 4.1, esso può disporre di diverse funzionalità, come ad esempio creare una nuova vista, modificare una vista esistente mediante l'aggiunta o l'eliminazione di membri oppure estrarre membri di interesse da una vista con cui instaurare la collaborazione.

Come indicato in figura, le operazioni possibili ad un'entità LME sono molteplici, però occorre distinguere ulteriormente il tipo di utilizzatore. Ad usufruire dei servizi del VMS non è infatti solamente un'applicazione collaborativa, ma anche gli altri componenti del middleware (mostrati dall'architettura di Figura 3.2) con cui il VMS interagisce e si coordina. Ad esempio, la creazione di una vista non viene inizializzata direttamente a livello applicazione, ma viene invocata dal J/LMS ogni qualvolta un'utente LME intende creare un nuovo gruppo. Così, anche l'aggiunta di un nuovo membro ad una vista, è un'operazione svolta internamente dal VMS, che coordinandosi col PENS, si accorge della presenza nella sua località di un nuovo membro appartenente al gruppo. E' bene distinguere pertanto, che solamente un subset degli use cases di Figura 4.1 sono disponibili al livello applicativo, ed in particolare sono le funzioni di utilizzo delle viste, come l'estrazione della vista e l'estrazione di un membro dalla vista. Al livello applicativo dunque, vengono nascoste tutte le altre funzioni di gestione delle viste che sono di competenza esclusiva del middleware.

Se l'utilizzatore del VMS è invece un'entità ME, le funzionalità a sua disposizione sono molto limitate. Al livello applicazione vengono fornite le medesime operazioni di utilizzo della vista già indicate negli use cases dell'LME, che sono l'estrazione di una vista e l'estrazione di un membro della vista. Le funzionalità a disposizione degli altri componenti di AGAPE, sono invece ristrette ai soli casi di cancellazione di una vista, nel caso in cui un nodo esegua leave dal gruppo, e di impostazione di una nuova vista, nel caso in cui un ME esegua join ad un nuovo gruppo di interesse. Le funzionalità per la modifica e per la creazione di nuove viste vengono soppresse, in quanto solamente un'entità LME può creare

nuovi gruppi e modificare le viste di quelli da essa gestiti.

La Figura 4.2 mostra gli use cases per l'entità ME.

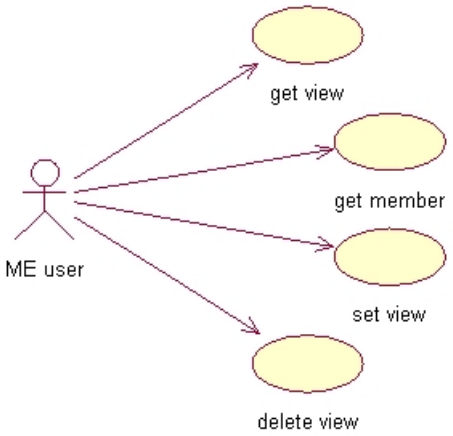

Figura 4.2: Use cases del VMS nel caso ME.

4.2 Analisi

In questa sezione si intende formalizzare quale sarà il modello di funzionamento del componente, quali saranno le sue peculiarità e quali le linee di progetto che guideranno il suo sviluppo.

4.2.1 Linee Guida

Nel progetto di applicazioni collaborative in ambito MANET, è necessario fare i conti con la limitatezza delle risorse dei dispositivi wireless, come la CPU, la memoria e la batteria. Data la presenza di ruoli fra le entità in gioco che vedono membri LME, dotati di dispositivi con ricche risorse computazionali, e membri ME, equipaggiati con dispositivi a risorse limitate, si delega la gestione dei gruppi alle entità caratterizzate da dispositivi ricchi. Gli LME pertanto svolgono tutte le funzioni di gestione dei gruppi mentre gli ME si limitano ad essere gestiti da questi.

La funzione principale del VMS (caso LME) è quella di distribuire le viste di contesto a tutti i membri di uno o più gruppi presenti in località, pertanto, al fine di risparmiare anche su questi dispositivi l'uso di risorse, è bene dotare il componente di un certo livello di complessità al fine di essere in grado di risparmiare risorse quando possibile.

Considerando il fatto che una buona percentuale del consumo totale di energia è dovuto ad operazioni di trasmissione/ricezione, è bene limitare al minimo l'impiego di banda utilizzando sempre quella strettamente necessaria. Per poter fare questo è necessario distinguere gli avvenimenti di rete in modo tale da poter reagire in maniera specializzata ad ognuno di essi, mediante l'invio, non di un'unico oggetto, ma di oggetti diversi a seconda

del caso. E' buona norma ad esempio aggregare pacchetti in un'unica trasmissione o inviare frame di un singolo byte nel caso non avvengono cambiamenti di topologia. Si sceglie ad esempio di lavorare su base temporale piuttosto che ad eventi e si rende il componente in grado di decidere, in base a delle euristiche, se ritrasmettere o meno aggiornamenti. Il VMS cerca pertanto di fornire sempre il miglior servizio possibile privilegiando il risparmio delle risorse utilizzate ed ottenendo come risultato un servizio di tipo *best-effort*.

La prima linea guida nel progetto del VMS è pertanto la *preservazione della banda*, e quindi indirettamente, anche dell'*energia*, due risorse di altissimo valore in ambito ad hoc.

A causa della grande dinamicità delle MANET, si possono avere situazioni in cui i nodi subiscono grandi spostamenti e ad alte velocità, ma anche situazioni in cui i nodi rimangono molto stazionari e subiscono piccoli spostamenti. Mediante un approccio in cui l'invio delle informazioni di topologia avviene ad intervalli costanti, in entrambi i casi appena esposti si avrebbe un degrado di performance: se la rete è molto dinamica il fatto di andare sempre alla stessa frequenza non coglierebbe i cambiamenti topologici in maniera reattiva, mentre nel caso molto stazionario sarebbe alquanto dispendioso procedere ad intervalli stretti, potendo eseguire anche in modo più moderato risparmiando risorse. E' necessario quindi un'approccio *context-aware*, che mi permetta di eseguire le funzioni di gestione in base alla dinamicità attuale della rete.

Si intende pertanto progettare il VMS secondo questa esigenza, rendendolo in grado di seguire i cambiamenti topologici. La gestione del gruppo diventa pertanto dipendente dal contesto della rete (context-aware). Questa seconda linea guida di progetto rende il sistema adattativo rispetto al contesto ed, indirettamente, permette di risparmiare anche sul consumo di batteria.

Nel caso di rete molto dinamica infatti, il fatto di aumentare la frequenza di esecuzione, non significa altro che monitorare la situazione di rete ad intervalli più stretti e quindi, per il Teorema sul campionamento di Shannon, avere meno variazioni alla topologia da un'istante all'altro di attivazione. Tutto questo si rilfette sulla quantità di informazioni da inviare ai membri del gruppo per aggiornarli sulle variazioni della rete e dunque, avere meno operazioni sul mezzo wireless, significa risparmiare batteria residua.

Come terza ed ultima linea guida di progetto si ha la limitazione dell'*occupazione di memoria*. Dato che la maggioranza dei dispositivi wireless è dotato di limitata quantità di memoria, è bene preservare anche questa risorsa occupando sempre la quantità strettamente necessaria.

4.2.2 Primitive di sistema

In accordo ai casi d'uso previsti per il componente, è possibile distinguere una primitiva per ognuno di essi. Se l'utilizzatore del VMS è un'applicazione collaborativa, le uniche primitive utilizzabili, sia nel caso LME che ME, sono la *getView* e la *getMember*, se invece l'utilizzatore è un'altro componente AGAPE, le primitive si estendono a tutti i casi d'uso previsti.

Nella stesura di queste API è necessario tenere in considerazione che il componente deve essere integrato all'interno di un middleware esistente e quindi è necessario definire

primitive di sistema interfacciandole in modo opportuno e con tutte le funzionalità richieste dalle altre entità del middleware. Le principali primitive realizzate sono le seguenti:

- **getView**: Restituisce la vista attuale relativa ad un gruppo. Questo metodo può essere invocato sia da un'altro componente di AGAPE sia da un'applicazione ogni volta che vuole conoscere i membri del gruppo a cui appartiene. (**ME**;**LME**)

- **initializeView**: Crea una nuova vista. Quando un LME vuole creare un nuovo gruppo deve anche inizializzare una nuova vista da inviare a tutti i membri. La creazione di un nuovo gruppo avviene negli LME da parte del J/LMS che prima crea il gruppo e poi invoca questo metodo per inizializzare nel VMS una nuova vista di gruppo. Inoltre tale metodo viene invocato anche nel caso in cui un LME inizi a gestire un gruppo creato da un'altro LME, cominciando a disseminare viste proprie per il gruppo. Questo metodo è chiaramente ad uso esclusivo degli LME. (**LME**)

- **deleteView**: Questa primitiva cancella una vista memorizzata. Quando un nodo esce da un gruppo in modo esplicito eseguendo un'operazione di leave, esso deve anche cancellare la vista relativa a tale gruppo, e lo fa mediante questa funzione. (**ME**;**LME**)

- **setView**: Imposta una nuova vista in memoria in seguito ad un'operazione di join ad un gruppo. Se il nodo che ha eseguito l'operazione di join è un ME tale metodo imposta come unica vista di gruppo quella passata come parametro, mentre se il nodo è un LME, dato che può appartenere a più gruppi contemporaneamente, il metodo imposta la nuova vista ed inizia la gestione del relativo gruppo. (**ME**;**LME**)

- **getAllViews**: Funzione che restituisce l'elenco di tutte le viste gestite da un LME. Può essere utile, sia a livello applicativo che per lo stesso VMS, conoscere gli identificatori di tutte le viste gestite in un determinato istante da un LME. (**LME**)

- **addMember**: Aggiunge un nuovo membro ad una vista, passando come parametri di ingresso quelli relativi alla vista che si intende modificare. Questo metodo viene invocato dal J/LMS di un LME a seguito di un'operazione di join andata a buon fine per aggiungere un nuovo membro ad un gruppo. Anche questo metodo di modifica delle viste è ad utilizzo esclusivo degli LME. (**LME**)

- **getMember**: Restituisce l'entry di una vista. Questo primitiva può essere invocata sia dall'applicazione sia da altri componenti AGAPE, per conoscere le informazioni di un'entità membro. (**ME**;**LME**)

4.2.3 Modello di funzionamento

Il componente che si intende realizzare deve seguire le linee guida di progetto elencate in precedenza e delle quali la più significativa è quella del risparmio di banda. Si vuole infatti realizzare un componente la cui peculiarità chiave è quella di saper scegliere, ad ogni trasmissione, sempre la soluzione che richiede il minor impiego di banda. Per poter fare

questo, vengono introdotti due ulteriori oggetti oltre alla vista, in modo tale da inviare, in ogni situazione, sempre l'oggetto più opportuno.

Con il termine oggetto si intende il tipo di messaggio da inviare, in ogni trasmissione, ai membri del gruppo, per aggiornare le loro viste. Questi tipi di messaggi sono pertanto tre:

- **Vista**: La vista coincide con quella già definita in precedenza e posseduta da ogni membro del gruppo. Alla ricezione di questo tipo di messaggio il membro sostituisce, se la versione è maggiore, la vista memorizzata con quella ricevuta.

- **Update**: Nel caso si abbiano delle piccole modifiche ad una vista, sarebbe superfluo propagare ai membri l'intera vista modificata, ma sarebbe sufficiente inviare un'oggetto che contenesse solamente tali modifiche. Questo oggetto è chiamato Update e contiene delle entry che riportano solamente i membri che si sono aggiunti od eliminati dalla vista a cui si riferisce.

- **Null**: Nel caso infine, che ad una vista, tra un'istante di attivazione ed il successivo, non si sia verificato alcun cambiamento, sarebbe alquanto dispendioso ritrasmettere l'intera vista, ad ogni membro, per comunicare che è rimasto tutto invariato. L'oggetto Null è un messaggio vuoto che indica proprio che il gruppo non ha subito cambiamenti topologici e tutti i membri possono continuare a lavorare con la propria vista memorizzata.

Questa distinzione serve al VMS di un LME per poter differenziare le trasmissioni scegliendo sempre la combinazione migliore di oggetti da inviare. Nel caso in cui, ad esempio, giunga un nuovo membro nella vista di gruppo, occorrerebbe trasmettere la vista completa al nuovo membro, mentre un update, contenente il solo membro aggiunto, a tutti gli altri membri della vista. Grazie alla specializzazione dei messaggi, in questo caso si invierebbe una sola vista ed una serie di update pari al numero dei membri del gruppo, con un notevole risparmio di banda utilizzata.

Per di più il VMS confronta, prima di ogni invio, il numero di byte necessari per la trasmissione che intende effettuare con quello dell'invio di sole viste a tutti i membri, e sceglie, in ogni caso, sempre la soluzione a minor impiego di banda. Il diagramma di Figura 4.3 mostra a grandi linee l'algoritmo per la spedizione dei messaggi.

L'algoritmo sceglie di inviare null in broadcast, se non si sono verificati cambiamenti alla topologia, altrimenti controlla il valore di due funzioni di peso, chiamate U e V, che misurano il numero di byte necessari per la trasmissione di update ed eventuali viste (U) e per la trasmissione di sole viste a tutti i membri (V). L'algoritmo sceglie che cosa propagare sempre in base al valore di queste due funzioni di costo, il cui significato dei simboli verrà spiegato nella Sezione 4.3.5.4:

$$U = (N - Y)weightUpdate + (X)weightView$$

$$V = (N - Y + X)weightView$$

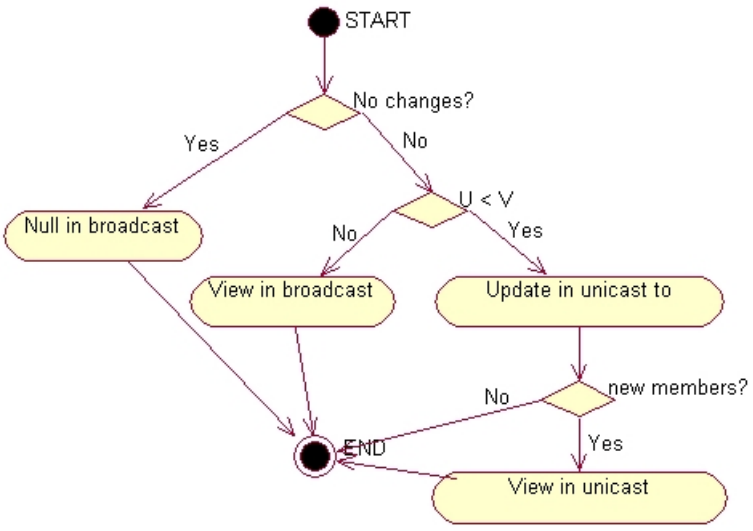

Figura 4.3: Modello della procedura di invio degli aggiornamenti.

Una seconda peculiarità del sistema che si andrà a progettare, sta nella possibilità di richiedere gli aggiornamenti andati perduti. Può accadere infatti, e non raramente, che alcuni messaggi vengano persi durante l'inoltro tra i nodi della MANET, lasciando il destinatario del messaggio con una vista *incoerente*[1]. Al fine di ripristinare lo stato attuale della rete, il nodo quando capisce che è stato smarrito un'aggiornamento, può richiederlo all'LME gestore del gruppo che provvederà alla ritrasmissione.

Si intende sviluppare infatti un'algoritmo di ritrasmissione che ancora una volta preservi la banda, scegliendo sempre la soluzione a minor costo, aggregando gli aggiornamenti dello stesso tipo e persino di scegliere di non ritrasmettere se si considera, in base a precise euristiche, che il nodo non è sufficientemente *"importante"* per il gruppo. L'algoritmo delle ritrasmissioni è mostrato a grandi linee in Figura 4.4.

Quando un LME riceve una richiesta, esegue prima di tutto una fase di controllo, al fine di determinare se tale richiesta può essere soddisfatta o meno. In questa fase controlla se il nodo è ancora presente, il livello di batteria residuo del nodo, la distanza del nodo, il numero di nodi aventi la vista stabile in quel determinato momento, ecc..., e se passa tutta questa serie di controlli prosegue con la fase di aggregazione.

Se una richiesta contiene più aggiornamenti dello stesso tipo, essi possono essere aggregati al fine di risparmiare banda. Se invece la richiesta contiene un solo aggiornamento tale fase viene saltata e si passa direttamente alla trasmissione. A questo punto l'LME controlla se il numero di byte richiesto dalla ritrasmissione è minore del numero di byte necessari per l'invio della vista attuale, e sceglie ancora una volta la soluzione a minor costo. Tale algoritmo infine, esegue anche delle scelte in base alla copertura del gruppo, cioè il numero di nodi con vista coerente, e privilegia ancora una volta il risparmio di banda anche a

[1]Si intende una vista con numero di versione più vecchio di quello attuale e quindi con possibili discordanze rispetto la topologia di rete attuale. Al contrario con il termine *coerente* si intende una vista che rispecchia fedelmente la situazione di rete in quel determinato istante.

scapito di una copertura non completa del gruppo. Tale algoritmo sarà approfondito nella Sezione 4.3.5.3.

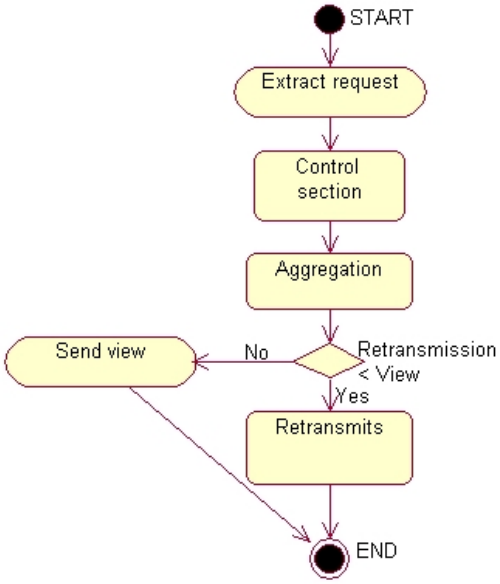

Figura 4.4: Modello della procedura di ritrasmissione degli aggiornamenti.

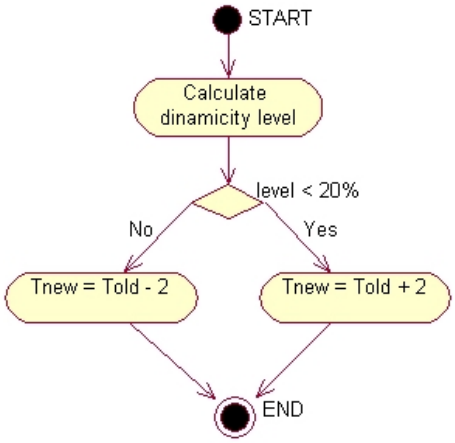

Figura 4.5: Modello della procedura di adattamento degli intervalli di attivazione.

L'ultima peculiarità distintiva del componente è il fatto di essere adattativo rispetto la dinamicità della rete. Come già spiegato infatti, gli istanti di attivazione del VMS variano a seconda della situazione topologica, e si sceglie di farli variare all'interno di un range tra i $2s$ ed i $10s$. L'algoritmo che adatta gli intervalli di attivazione è mostrato in una versione semplificata in Figura 4.5.

In pratica l'algoritmo calcola un livello di dinamicità della rete misurando la percentuale dei cambiamenti rispetto all'attivazione precedente. Se tale livello è al di sotto del 20%, significa che le variazioni sono poche ed è possibile rilassare l'intervallo, se invece il livello è più alto del 20% significa che ci sono state molte variazioni e dunque la rete è più dinamica. E' necessario quindi, in quest'ultimo caso, dover restringere l'intervallo al fine di seguire i cambiamenti della rete in modo più reattivo.

Si rimanda alla Sezione 4.3.5.5 per uno sguardo più approfondito di tale algoritmo.

4.2.4 Architettura logica

In base a quanto detto finora e considerando l'architettura generale del sistema (vedi Figura 3.2) all'interno del quale andrà ad operare il VMS, l'organizzazione logica di riferimento per questo componente risulta essere quella mostrata in Figura 4.6, ed è valida per entrambi i tipi di entità. La differenza sostanziale tra le due release risiede nel componente *VMService*.

L'architettura del VMS si articola di quattro componenti principali: due ascoltatori dei messaggi in arrivo, un gestore dei messaggi per la loro interpretazione ed infine il componente che si interfaccia con il livello applicativo e con gli altri componenti di AGAPE. E' mediante quest'ultimo componente che è possibile utilizzare le primitive ed i servizi forniti dal VMS.

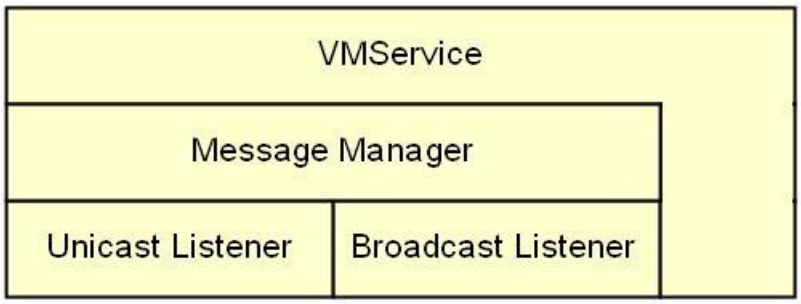

Figura 4.6: Architettura logica del View Manager Service.

Il livello inferiore è costituito di due unità, di cui una parte è soggetta al trattamento dei messaggi unicast (*Unicast Listener*) ed una per quelli di tipo broadcast (*Broadcast Listener*). Infatti il VMS sia per la ricezione che per l'invio dei messaggi fa uso delle primitive fornite dal livello sottostante del middleware e, per quanto riguarda la parte di ricezione, il lavoro viene svolto proprio da parte di questi due blocchi che si interfacciano direttamente con l'NMS presente al livello inferiore di AGAPE. I due ascoltatori realizzano

infatti la funzione di ascolto sfruttando le porte di ingresso unicast e broadcast fornite da questo componente.

Quando un ascoltatore riceve un messaggio diretto al VMS, esso lo passa direttamente al livello superiore per il suo trattamento. Data l'intenzione, come descritto nella sezione precedente, di voler trasmettere non solo viste, ma diversi tipi di messaggi, il gestore dei messaggi (*Message Manager*) è necessario al fine di trattare correttamente ognuno di essi mediante la giusta procedura. Una volta individuato il tipo messaggio, questo componente lo passa alla procedura opportuna per l'interpretazione e fornisce gli eventuali risultati al livello soprastante.

Il componente dello strato più alto (*VMService*) è l'unica unità visibile a livello applicativo e dagli altri componenti di AGAPE, ed è quella che realizza e fornisce i servizi definiti mediante le primitive di sistema. Questa parte sfrutta il servizio di ricezione e di trattamento dei messaggi realizzato dagli ascoltatori e dal gestore dei messaggi e svolge tutte le funzioni dichiarate per il componente. Per l'esecuzione dei suoi compiti anche questa parte necessita dei servizi forniti dal livello inferiore del middleware ed in particolare, parlando di nodi LME, per l'esecuzione del compito fondamentale di diffusione delle viste. Come indicato in Figura 4.6 quindi, anche l'unità VMService deve poggiare sullo strato inferiore al fine di poter utilizzare le primitive di comunicazione fornite dall'NMS, nonchè i servizi forniti dal PENS e dal PS.

Osservando l'architettura di riferimento e tenendo a mente l'architettura generale del middleware ospitante, si può riassumere che il VMS confina al di sotto con lo strato *Basic Service Layer* di AGAPE, sfruttandone i servizi, a destra e sinistra con gli altri componenti del livello *Group Management Layer* quali J/LMS e VCS, con i quali interagisce e fornisce servizi, ed infine al di sopra con l'eventuale applicazione per la quale fornisce il servizio.

4.3 Design

Questa sezione illustra la progettazione dei componenti dell'architettura presentata con particolare enfasi per il VMService, oggetto cardine di questo studio. Si procede in modo *bottom-up* rispetto all'architettura logica e si utilizza il tool di simulazione, esposto in Appendice A, per sviluppare e testare in tempo reale le performance degli algoritmi.

4.3.1 Tipi di oggetti

Come descritto durante la fase di analisi, il tipo di oggetto principalmente trattato dal VMS è la vista. Questa è l'oggetto direttamente fornito al livello applicativo od ai componenti che lo richiedono ma non risulta essere sempre la soluzione migliore durante la fase di invio periodico degli aggiornamenti ai membri di un gruppo. Dato che questa è la funzione cardine svolta dal VMS (lato LME) e quella più dispendiosa in termini di risorse, è bene trattare con cura questa fase già da ora.

Dato che la linea guida principale è quella del risparmio di banda e quindi di energia, si è scelto di non limitare il componente al solo invio di viste, ma anche di altri oggetti che risultano meno dispendiosi in termini di dimensione, in modo tale da utilizzare la soluzione

sempre a minor costo. Questa scelta può non sembrare opportuna se si pensa a delle MANET con pochi nodi e poco dinamiche, ma in reti più estese e più variabili (situazione più vicina a quella reale) questa differenziazione degli oggetti trattati risulta la soluzione vincente.

Durante la fase di invio degli aggiornamenti periodici dunque, si è scelto di dotare il componente di una certa libertà di scelta che privilegia sempre l'oggetto a minore impatto sulla banda. Gli oggetti inviati dal VMS (lato server) e ricevuti dal VMS (lato client) sono i seguenti:

1. **View**: Questo oggetto è la vista vera e propria e viene memorizzata sul lato client così com'è stata ricevuta, rimpiazzando una vista precedente. Alla ricezione, una vista non necessita di particolari trattamenti ma risulta subito disponibile per la lettura.

2. **Update**: L'oggetto update rappresenta l'aggiornamento di una vista già memorizzata sui nodi destinatari di tale messaggio. Quando le modifiche alla topologia sono poche e risulta superfluo l'invio dell'intera vista a tutti i nodi, il VMS sceglie di inviare update preservando notevolmente la banda.

3. **Null**: Quest'ultimo oggetto viene utilizzato dal VMS quando tra due istanti di attivazione consecutivi non rileva alcun cambiamento di topologia, occupando la minor banda possibile dato che non contiene alcuna informazione al suo interno.

In Figura 4.7 vengono mostrati gli attributi di questi tre oggetti.

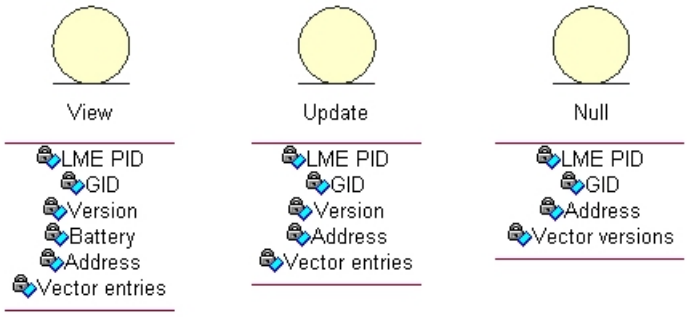

Figura 4.7: Class diagram degli oggetti View, Update e Null.

Gli oggetti vista e update contengono i medesimi attributi ma si differenziano nel tipo di entry contenute in *Vector entries*. Questi due contengono quindi l'identificatore del nodo LME che ha inviato l'oggetto (*LME PID*), l'identificatore del gruppo al quale si riferisce l'aggiornamento (*GID*), il numero di versione dell'aggiornamento per poter essere correttamente trattato (*Version*), un indicatore del livello di batteria residuo dell'LME (*Battery*), l'indirizzo IP dell'LME (*Address*) ed infine un vettore di entry che contiene i dati relativi ai nodi del gruppo (*Vector entries*). La differenza tra i due oggetti sta proprio in questo: mentre la vista contiene tutti i nodi appartenenti ad un gruppo in

un determinato istante, l'update contiene solamente le entry dei nodi che hanno portato modifiche all'overlay di rete tra due istanti consecutivi. Esiste una differenza fondamentale anche tra questo tipo entry: al fine di preservare al massimo l'utilizzo di banda si è scelto di impiegare due tipi diversi di entry per l'oggetto update. Si ha pertanto una *entry positiva* nel caso in cui il nodo si è aggiunto al gruppo ed una *entry negativa* nel caso sia uscito da tale gruppo. La motivazione di questo sta nel fatto che per un'entry positiva è necessario inviare tutte le informazioni relative al nodo al fine di essere aggiunto, mentre per una negativa è sufficiente inviare solamente l'identificatore del nodo in modo tale da essere eliminato dagli altri membri del gruppo.

Per quanto riguarda l'oggetto Null invece è sufficiente che esso riporti l'identificatore del nodo LME che l'ha inviato (*LME PID*), l'identificatore del gruppo al quale si riferisce (*GID*) e il numero di versione (Vector version). Quest'ultimo, a differenza degli altri due casi, non è un singolo numero di versione ma un vettore di numeri, al fine di poter accumulare, dato che non contengono informazioni da gestire, più messaggi dello stesso tipo in uno unico. Questo oggetto serve in pratica per comunicare ai nodi di un gruppo che la situazione non è cambiata rispetto all'istante precedente, utilizzando la minor quantità di banda possibile, quella della serializzazione dell'oggetto.

La Figura 4.8 mostra gli attributi delle entry degli oggetti View e Update.

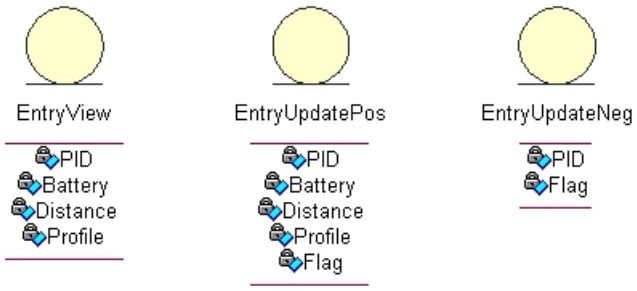

Figura 4.8: Class diagram degli oggetti EntryView ed EntryUpdate.

L'entry di una vista contiene tutte le informazioni necessarie per l'interazione con un nodo e sono quelle direttamente utilizzate a livello applicativo. Vi è pertanto per ogni nodo: un'identificatore del nodo all'interno del gruppo (*PID*), il livello di batteria residuo (*Battery*), la distanza del nodo dall'LME che gestisce il gruppo (*Distance*) ed infine il profilo dell'utente (*Profile*). Anche le entry positive di un update contengono le stesse informazioni dal momento che l'utente deve essere aggiunto ad una vista già esistente. Pertanto sono presenti tutti gli attributi precedenti ed uno ulteriore al fine di poter distinguere, all'interno dello stesso update, un entry positiva da una negativa (*Flag*).

Infine, l'entry negativa di un'update contiene le informazioni essenziali per la sua cancellazione da una vista: il pid del membro da eliminare (*PID*) ed il flag per distinguerlo dall'altro tipo di entry (*Flag*). Il flag assume chiaramente valori positivi per entry positive e negativi nell'altro caso.

Al fine di rendere più pulita l'estrazione delle informazioni riguardanti un membro,

queste verranno raggruppate all'interno di ulteriore oggetto rappresentante il membro vero e proprio. La Figura 4.9 mostra gli attributi di questo oggetto.

Figura 4.9: Class diagram dell'oggetto Member.

Osservando gli attributi di quest'ultimo è evidente come esso vada a sostituire i medesimi attributi all'interno degli oggetti EntryView ed EntryUpdatePos dove il prima risulterà contenere solamente un'oggetto membro ed il secondo un oggetto membro più il flag.

4.3.2 Unicast e Broadcast Listener

I plug-in ascoltatori dei messaggi in arrivo utilizzano le primitive fornite al livello inferiore dal componente NMS. Essi pertanto creeranno una porta di ascolto rispettivamente unicast e broadcast ed ogni volta che giunge un messaggio lo passano al gestore dei messaggi per il trattamento.

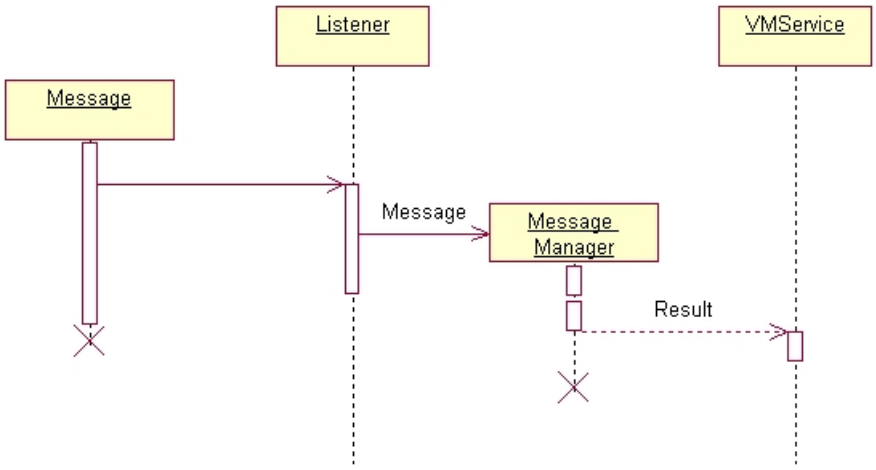

Figura 4.10: Sequence diagram del processo di ricezione e valutazione di messaggi.

La Figura 4.10 mostra lo schema di funzionamento di questo metodo in cui viene indicato un oggetto Listener generico, che può essere sia quello unicast che broadcast, che riceve un messaggio di uno dei due tipi. Alla ricezione di tale messaggio il Listener non

fa altro che inoltrarlo al gestore dei messaggi che sarà proprio generato da quest'ultimo e che terminerà la propria esecuzione alla fine dell'interpretazione ed alla restituzione del risultato al VMService. Una volta passato un messaggio al gestore invece, l'unità listener rimane di nuovo in ascolto sulla porta per il rilevamento di nuovi messaggi.

La classe che realizza il Listener estende direttamente la classe ActiveComponent di AGAPE, al fine di avere un oggetto che esegua un task ad intervalli di tempo regolari. Questa infatti ad intervalli regolari oppure senza alcuna pausa, legge mediante il metodo `readObject()` gli oggetti giunti sulla porta di ascolto e nel caso ci siano, genera un nuovo Message Manager passandogli il messaggio da interpretare come parametro.

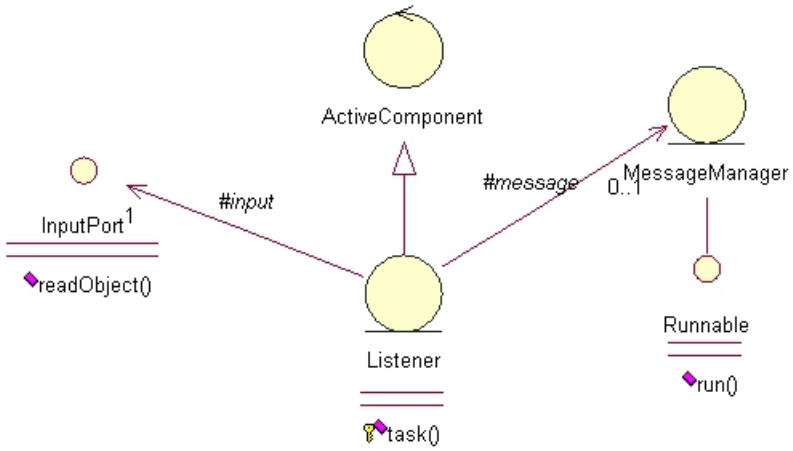

Figura 4.11: Architettura logica del componente Listener.

4.3.3 Message Manager

Questo componente si occupa del delicato compito di interpretare correttamente i messaggi giunti al VMS al fine di apportare le giuste modifiche alla vista attuale. Data la varietà dei messaggi che possono giungere al componente quest'ultimo è stato progettato implementando al suo interno diverse procedure di gestione. Lo schema di funzionamento è questo: una volta che il gestore è stato creato passandogli un oggetto come parametro, quest'ultimo riconosce il tipo di messaggio, lo passa alla giusta procedura, che andrà a modificare le strutture dati del VMService, e termina la propria esecuzione.

Il diagramma rappresentato in Figura 4.12 rappresenta la parte computata dal lato client, cioè i nodi ME, ma dal punto di vista del lato server è bene fare una precisazione, in quanto oltre a questi tre tipi di oggetti essi possono ricevere anche altri due tipi, in particolare il profilo di un nodo che vuole inserirsi in un gruppo (*MyProfile*) oppure una richiesta di ritrasmissione di un aggiornamento perso (*RequestUpdate*). Questi due casi saranno meglio spiegati in seguito, ma per ora occorre solamente tenere in conto della presenza di due ulteriori procedure di gestione nel Manager dei messaggi, una *MyProfile procedure* ed una *RequestUpdate procedure*.

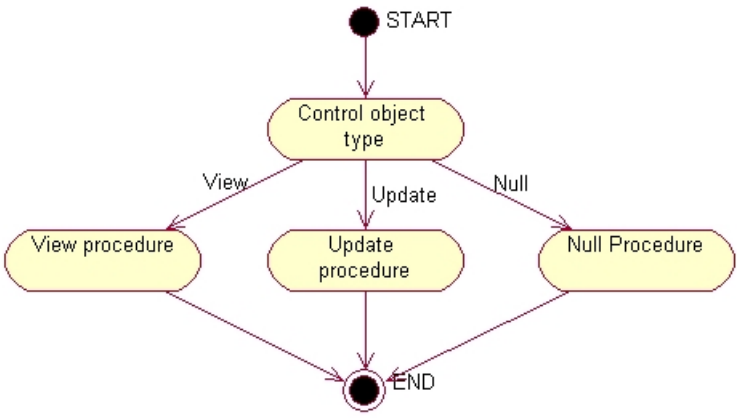

Figura 4.12: State diagram del Manager dei messaggi.

L'architettura logica del Message Manager sarà quindi quella mostrata in Figura 4.13 che riporta il caso più generale dell'LME.

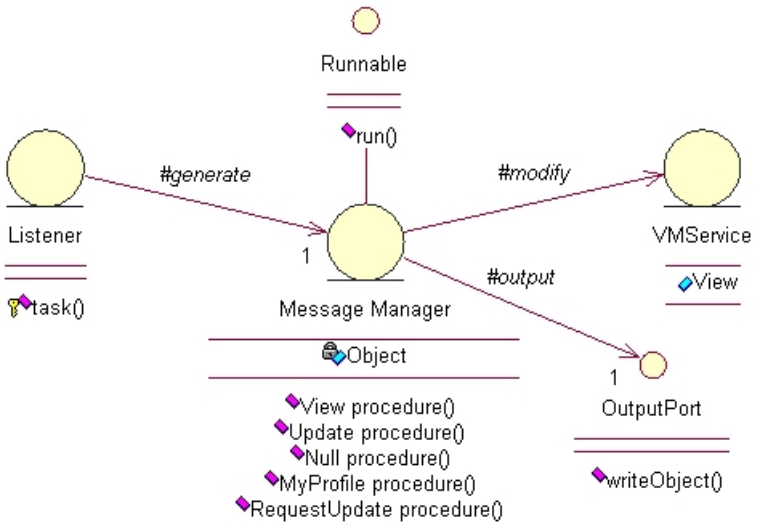

Figura 4.13: State diagram del Manager dei messaggi.

Lo sviluppo del componente procederà ora analizzando caso per caso le varie procedure fornendo per ciascuna uno schema di funzionamento mediante diagrammi di stato UML.

4.3.3.1 View procedure

La procedura di gestione delle viste si differenzia nei due casi di nodo ME ed LME. Il motivo principale di questa divergenza sta nel fatto che mentre un nodo gestito ME può appartenere, in un determinato istante, ad un solo gruppo per volta, un nodo gestore LME

può creare, gestire e quindi appartenere, a più gruppi nel medesimo istante. All'arrivo di una vista pertanto le procedure nei due casi distinti si comporteranno diversamente, ed in particolare il primo ignora la vista se non è quella del gruppo di appartenenza, mentre il secondo ignora la vista se non è tra quelle dei gruppi da esso gestiti.

Il diagramma di Figura 4.14 mostra il caso del nodo ME. Quando un nodo membro di un gruppo riceve una vista, deve eseguire una serie di controlli prima di poter svolgere le azioni di trattamento della stessa. Estrae innanzitutto il gid e il numero di versione della vista ricevuta ed inizia la fase di controllo per verificare se quella ricevuta deve essere trattata o scartata. Controlla quindi se in quel determinato istante esso appartiene ad un gruppo eseguendo una verifica sulla vista memorizzata: se la vista in quel momento è uguale a `null`, il nodo non appartiene a nessun gruppo e può quindi scartare la vista ricevuta, mentre se è diversa da tale valore significa il contrario e prosegue con il controllo.

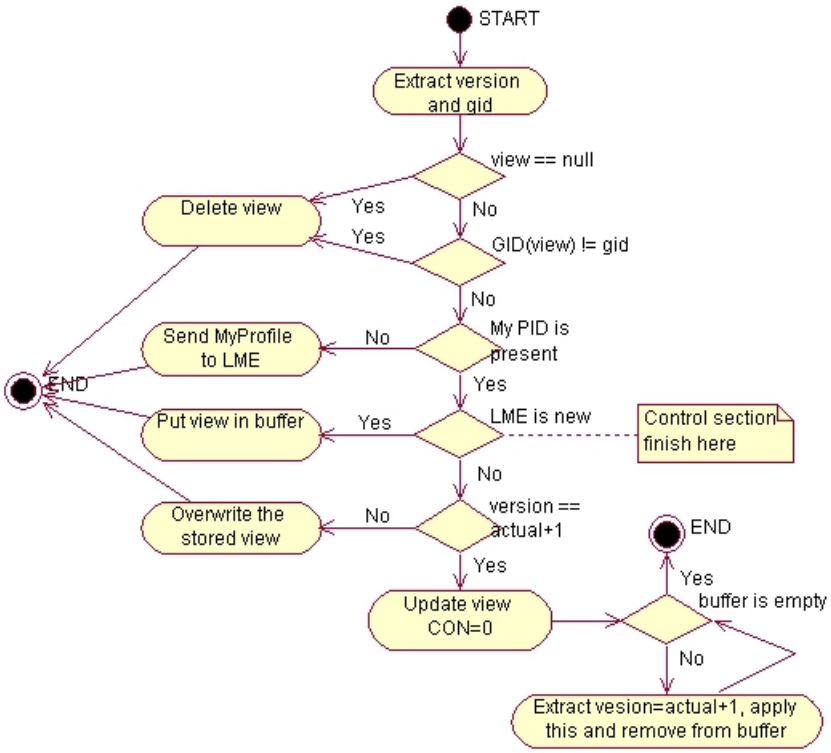

Figura 4.14: State diagram della procedure di gestione delle viste (caso ME).

A questo punto occorre verificare se il GID del gruppo della vista ricevuta coincide con quello del gruppo di appartenenza: se la vista si riferisce ad un'altro gruppo viene scartata mentre se è lo stesso si prosegue. Ora occorre controllare se all'interno dei membri della vista è presente il PID del nodo in questione: se tale identificatore non è presente, significa che il gestore non è stato in grado di inserire il nodo nella vista a causa della mancanza di informazioni sul profilo del membro, ed è quindi necessario inviare il proprio profilo all'LME mittente. Se al contrario l'entry del nodo è presente si prosegue nell'algoritmo.

Dato che uno stesso gruppo può essere gestito da più nodi LME e quindi, che uno

stesso nodo può ricevere viste da più di un gestore contemporaneamente, si mantiene in memoria delle strutture dati separate, una per ciascun nodo LME da cui giungono viste contemporaneamente ottenendo come vista finale utilizzabile al livello superiore quella che risulta dal *merge*[2] di tutte le viste ricevute. In pratica ad intervalli regolari il VMS esegue l'operazione di merge su tutte le viste possedute nelle proprie strutture dati.

Fatta questa premessa, a questo punto dell'algoritmo, è necessario controllare se nelle proprie strutture dati è presente l'entry relativa all'LME mittente in questione: pertanto se tale vista è già contenuta si prosegue, altrimenti si inserisce una nuova entry in memoria relativa al nuovo gestore scoperto.

A questo punto può iniziare il trattamento della vista vero e proprio. Si controlla il numero di versione al fine di verificare la freschezza della vista e nel caso sia successivo a quello della vista memorizzata , si modifica quest'ultima con le nuove informazioni ricevute. Invece nel caso in cui la versione della vista ricevuta è maggiore della successiva, si elimina la vecchia sovrascrivendo la nuova. Il significato del contatore CON verrà spiegato nella prossima sezione.

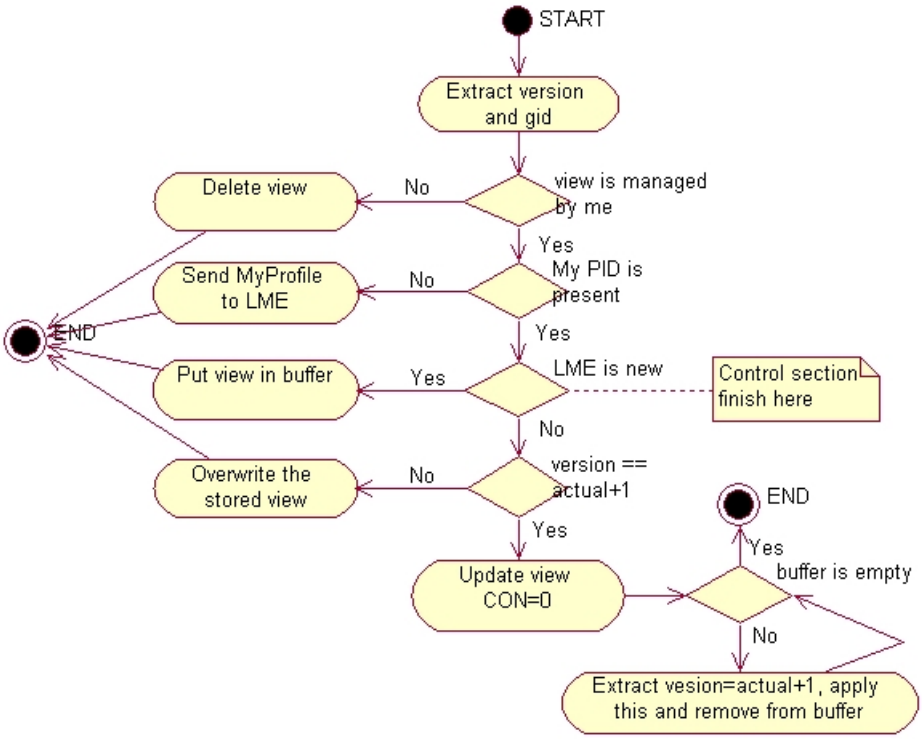

Figura 4.15: State diagram della procedure di gestione delle viste (caso LME).

Vi è infine la presenza di un buffer contenente aggiornamenti pendenti che non è stato possibile applicare direttamente per cui, una volta modificata la vista, è necessario controllare tale buffer al fine di valutare se sono presenti oggetti applicabili. Se vi sono

[2]Operazione che prevede la fusione di tutte le viste memorizzate in una vista globale unica contenente tutti i membri di ciascuna vista senza ripetizioni.

aggiornamenti pendenti utilizzabili (cioè hanno versione successiva a quella memorizzata) questi vengono estratti, applicati e rimossi dal buffer. Quando non ve ne sono più termina l'esecuzione.

La Figura 4.15 mostra lo stesso algoritmo nel caso LME che sostanzialmente varia solamente nella parte di controllo iniziale, pertanto ci soffermeremo soltanto su questa.

A differenza del caso precedente pertanto il nodo gestore controlla prima se la vista ricevuta fa parte dei gruppi gestiti da esso: se la vista si riferisce ad un gruppo estraneo viene scartata altrimenti si prosegue con l'algoritmo che risulta, da questo punto in poi, identico al caso ME.

4.3.3.2 Update procedure

Mantenendo la sezione di controllo identica ai casi precedenti sia per i nodi ME sia per quelli LME, verrà ora affrontata l'analisi solamente della parte di trattamento dell'oggetto ricevuto, e cioè in questa sezione dell'oggetto Update e nella prossima dell'oggetto Null. In Figura 4.16 viene mostrato il diagramma di stato della procedura di gestione degli update valido per entrambi i tipi di nodi, dove viene indicato con un'unico blocco la fase di controllo precedentemente descritta.

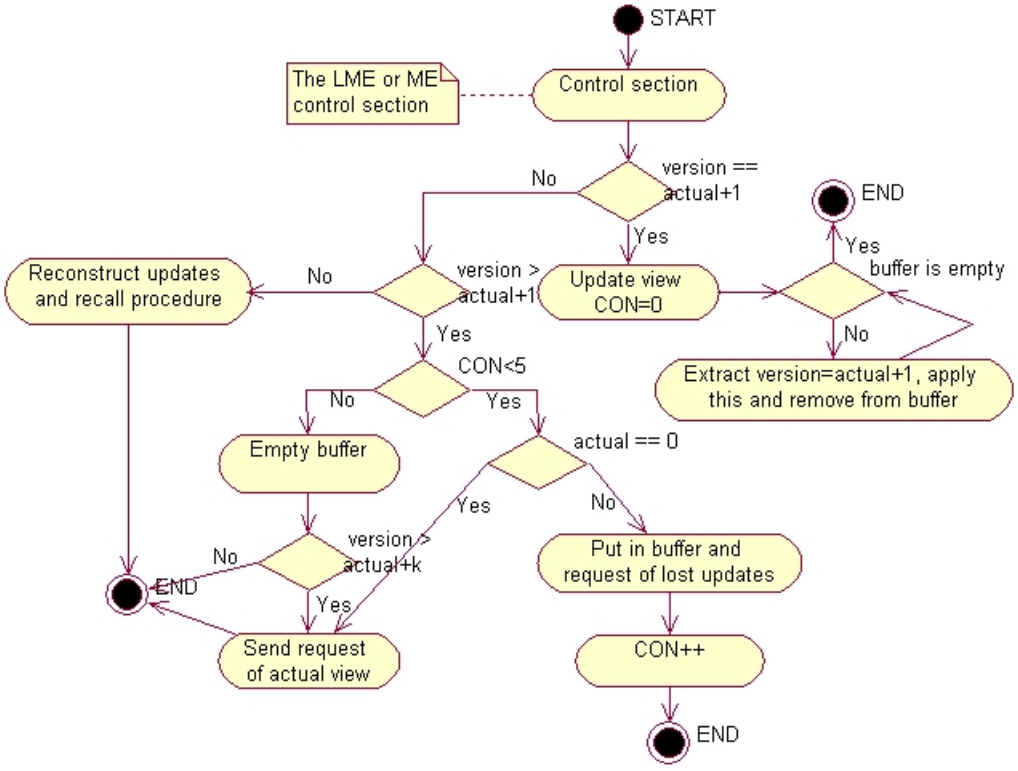

Figura 4.16: State diagram della procedure di gestione degli update.

Dato che il caso di versione successiva all'attuale è già stato discusso nella precedente sezione, ci si occupa ora della parte di sinistra del diagramma. Se la versione non è quella

successiva direttamente trattabile è necessario controllare se si tratta di una versione maggiore di quest'ultima: se non lo è significa che il numero di versione è negativo. In questo caso particolare, il segno negativo alla versione significa che l'update ricevuto è il risultato di un'operazione di merge su più update ed è necessario, al fine del suo trattamento, riassemblare gli update originali. Una volta ricostruiti si invoca nuovamente il Message Manager passando come parametro, uno per volta, gli update singoli. Questo metodo è stato adottato al fine di preservare banda nel caso in cui sia necessario ritrasmettere più update, evitando l'invio di più oggetti e quindi più byte da trasmettere privilegiando l'invio di un oggetto singolo che li contenga tutti.

Nel caso in cui invece la versione non è negativa è necessario controllare il valore del contatore CON al fine di valutare se richiedere gli aggiornamenti pendenti o meno. Questo contatore infatti è stato introdotto solamente per limitare l'utilizzo di importanti risorse sui dispositivi mobili. Il ragionamento che sussiste alla base di questo fatto è questo: se, dopo alcune richieste di ritrasmissione, ancora non è stata ricevuta alcuna risposta, significa che il gestore non è più raggiungibile oppure non intende rispondere e dunque è plausibile presupporre che anche nell'immediato futuro esso possa non soddisfare la richiesta. Pertanto dopo alcuni tentativi (pari al valore del contatore CON) il nodo cessa l'invio di richieste, risparmiando banda e batteria, e continua a lavorare con una vista *incoerente*. Questo valore è stato settato pari a 5 tentativi ma può essere modificato a seconda degli scenari e delle esigenze dell'applicazione.

Pertanto quando il numero di tentativi massimo è stato raggiunto, si prosegue svuotando il buffer degli aggiornamenti pendenti e si richiede, trascorso un certo intervallo di tempo pari a k numeri di versione di oggetti ricevuti, richiedendo la vista attuale, che una volta giunta sbloccherà tale situazione di incoerenza. Nel caso in cui invece è possibile fare ancora tentativi di richieste di aggiornamenti persi, si prosegue richiedendo la vista attuale, nel caso in cui la vista memorizzata ha versione 0, oppure tutte le versioni mancanti nel caso in cui la versione memorizzata è maggiore di 0. In quest'ultimo caso è necessario anche inserire l'update ricevuto che non può essere applicato nel buffer ed incrementare di un'unità il contatore CON.

4.3.3.3 Null procedure

La procedura di gestione degli oggetti Null è in pratica identica alla precedente con l'unica differenza che ora l'unica informazione utile alla modifica della vista memorizzata è il numero di versione del Null. Sempre per lo stesso fine di risparmiare l'utilizzo di banda, nel caso di richieste di ritrasmissione di più Null, è stato ancora una volta adottato il metodo della fusione di più oggetti in uno globale, che nel caso dei Null significa raggruppare tutti i numeri di versione in un vettore. In questo modo tutta la procedura vista per il trattamento degli update è la stessa di ora e viene reiterata per ogni numero di versione estratto contenuto nel vettore. Prima di uscire pertanto è necessario controllare ad ogni ciclo se il vettore contiene altri numeri di versione ed in caso affermativo tornare al punto iniziale e ripercorrere la procedura.

L'unica differenza dal caso precedente sta quindi nell'introduzione di un nuovo blocco

prima dell'uscita dalla procedura per il controllo sul vettore delle versioni e da un blocco all'inizio del ciclo per l'estrazione dei singoli elementi dal vettore.

La Figura 4.17 mostra questo comportamento.

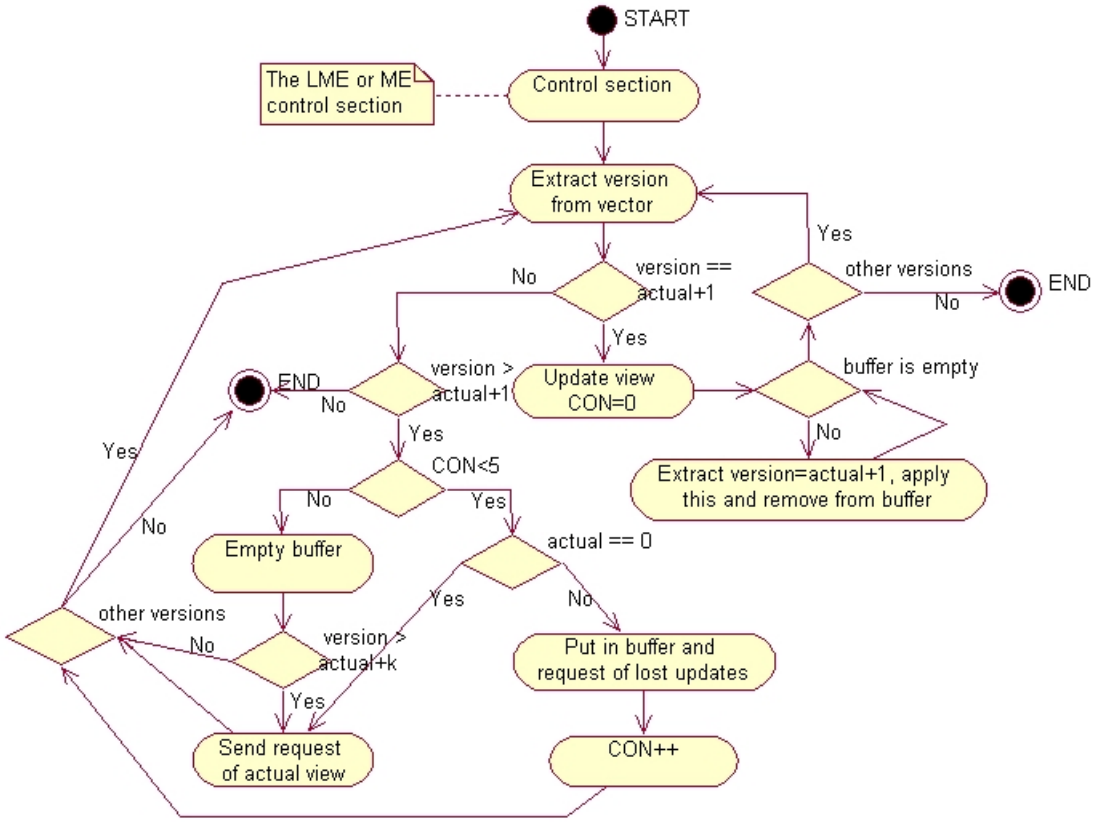

Figura 4.17: State diagram della procedure di gestione dei Null.

4.3.3.4 MyProfile procedure

Questa procedura e la successiva riguardano esclusivamente il caso LME, in quanto sono tipi di oggetti che possono essere ricevuti solamente dai nodi gestori. Nel caso in cui l'oggetto ricevuto sia un *MyProfile*, significa che il mittente di tale messaggio, sia ME che LME, richiede il proprio inserimento in una determinata vista. Questo può succedere ad esempio quando si riceve una vista di un gruppo al quale si appartiene ma in cui non è presente la propria entry. Al fine di effettuare l'inserimento della propria entry in tale vista è dunque necessario inviare il proprio profilo all'LME gestore. Tutte le altre informazioni necessarie per la costruzione dell'entry sono già in possesso del gestore e sono ottenute invocando le opportune API del PENS.

Quando un nodo gestore riceve un oggetto MyProfile quindi, estrae prima il gid della vista alla quale si riferisce, costruisce l'entry ed aggiunge l'entry a tale vista.

4.3.3.5 RequestUpdate procedure

Anche questo tipo di oggetto può essere ricevuto solamente da un nodo LME e significa la richiesta, da parte del nodo mittente, di aggiornamenti persi. Come mostrato dai diagrammi precedenti infatti quando un nodo si accorge della mancanza di alcuni aggiornamenti, esso può richiederli al nodo gestore presente in località proprio mediante questo tipo di messaggio. Alla ricezione di tale oggetto quindi il nodo LME inserisce la richiesta in un buffer che sarà trattato e svuotato ad ogni istante di attivazione del VMService da una procedura dedicata per le ritrasmissioni.

4.3.4 VMService: ME

Per la trattazione di questo componente del VMS è necessario separare le due tipologie di host in quanto profondamente diverse. Mentre il VMS di un ME deve semplicemente ricevere le viste e gli aggiornamenti da parte degli LME in località e quindi fornire una vista coerente a tutti i richiedenti, il VMS di un LME deve svolgere tutte le funzioni di gestione realizzandole al meglio secondo le linee guida stilate nei paragrafi precedenti. In questa prima sezione ci si occupa del lato ME.

Come tutti i componenti attivi di AGAPE, anche il VMS deve realizzare un servizio attivo che ad intervalli di tempo regolari svolga i propri compiti. Sul lato server questo si concretizza con la fruizione di tutti i servizi gestionali riguardanti i gruppi gestiti, mentre sul lato client, dato che la funzione di ricezione e modifica delle viste è realizzata dal Message Manager, questo viene realizzato svolgendo ad intervalli di tempo regolari due particolari funzioni interne al componente. Il VMService di un nodo ME pertanto oltre a fornire le API fondamentali del componente VMS, svolge ad intervalli di tempo regolari, queste due operazioni di controllo invocate mediante l'attivazione di un metodo `task`. La prima riguarda il refresh sulla località per verificare la costante presenza degli LME scoperti e la seconda per eseguire la funzione di merge sulle viste ricevute da tutti gli LME in località.

La procedura `controlLME` svolge la prima delle funzioni citate. Qualora un nodo ME appartenga ad un gruppo per il quale non riceve più aggiornamenti a causa della sua uscita dalla località dell'LME gestore, è necessario che esso elimini dalle proprie strutture dati l'entry relativa a tale LME in quanto non più raggiungibile. Se vi sono altri LME nel suo range di comunicazione esso continuerà a ricevere le viste e gli aggiornamenti da parte di quest'ultimo mentre se non vi sono altri gestori è costretto a continuare a lavorare con l'ultima vista memorizzata. che diverrà presto incoerente, in attesa dell'arrivo di un qualunque LME gestore del proprio gruppo di appartenenza.

La procedura `mergeViews` invece è necessaria per la fusione delle viste ricevute da parte di tutti gli LME raggiungibili, al fine di costruire la vista finale coerente da fornire a chiunque la richieda mediante l'apposita primitiva. Chiaramente, nel caso della presenza di un solo LME gestore in località tale procedura fornirà semplicemente come vista coerente l'unica vista memorizzata senza alcuna computazione.

In Figura 4.18 viene raffigurata l'integrazione del componente all'interno dell'architettura logica di riferimento. Come si può vedere, i metodi indicati dalla classe VMService corrispondono, ad esclusione delle due procedure sopraindicate, ai casi d'uso mostrati in

Figura 4.2. Tra gli attributi del componente VMService invece si nota la presenza della vista del gruppo di appartenenza costruita mediante la procedura `mergeViews` e tre oggetti necessari per realizzare le strutture dati: un oggetto `cache` che memorizza gli aggiornamenti pendenti non applicabili, un oggetto `versions` dove vengono memorizzati i numeri di versione degli aggiornamenti mancanti ed un oggetto `groups` dove vengono memorizzate tutte le viste ricevute da parte di tutti gli LME e sulle quali opera la `mergeViews`.

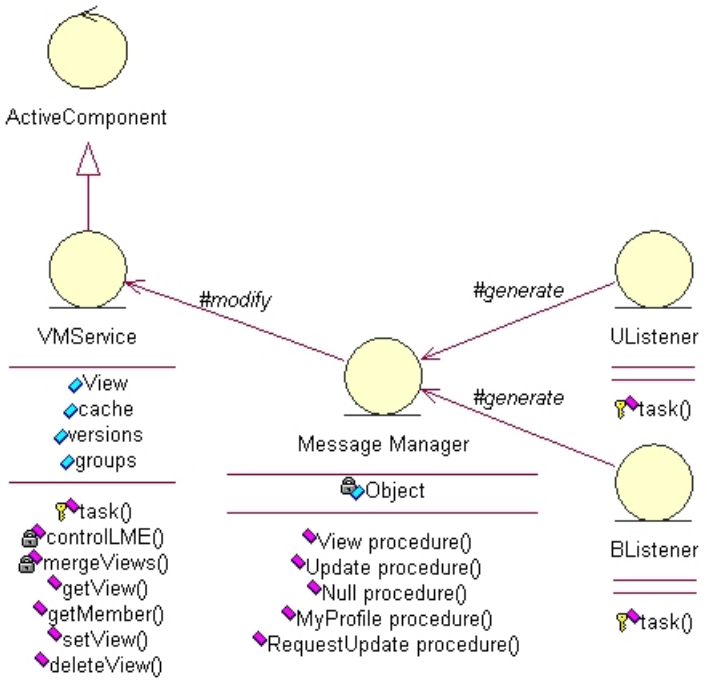

Figura 4.18: Class diagram dell'architettura logica finale del VMS nel caso ME.

4.3.5 VMService: LME

Il componente VMService sul lato server deve realizzare invece tutte le politiche di gestione dei gruppi ed è il responsabile di tutte le operazioni di invio e ritrasmissione di viste ed aggiornamenti. Per la realizzazione di queste funzioni è necessaria l'implementazione di procedure specifiche, che verranno invocate ad intervalli di tempo, ora non più regolari, dal solito metodo `task`.

La Figura 4.19 mostra l'integrazione del componente all'interno dell'architettura logica di riferimento, il quale, a differenza del caso precedente, mostra più metodi ed utilizza le primitive del livello inferiore del middleware. Vengono pertanto indicati come attributi tutte le strutture dati necessarie per la corretta manutenzione delle informazioni e per la corretta esecuzione delle procedure. In particolare la `stableViews` contiene le viste finali offerte dal servizio a tutti i richiedenti, la `groupsComponent` contiene tutte le informazioni relative a ciascun gruppo gestito, la `cronologyBuffer` contiene la storia di tutte le trasmissioni fatte in precedenza al fine di soddisfare richieste di ritrasmissione, la `requestUpdate`

è un buffer per la memorizzazione di tutte le richieste di ritrasmissione che giungono al componente ed infine la `time` è un struttura contenente, per ciascuna vista, gli istanti di attivazione successivi.

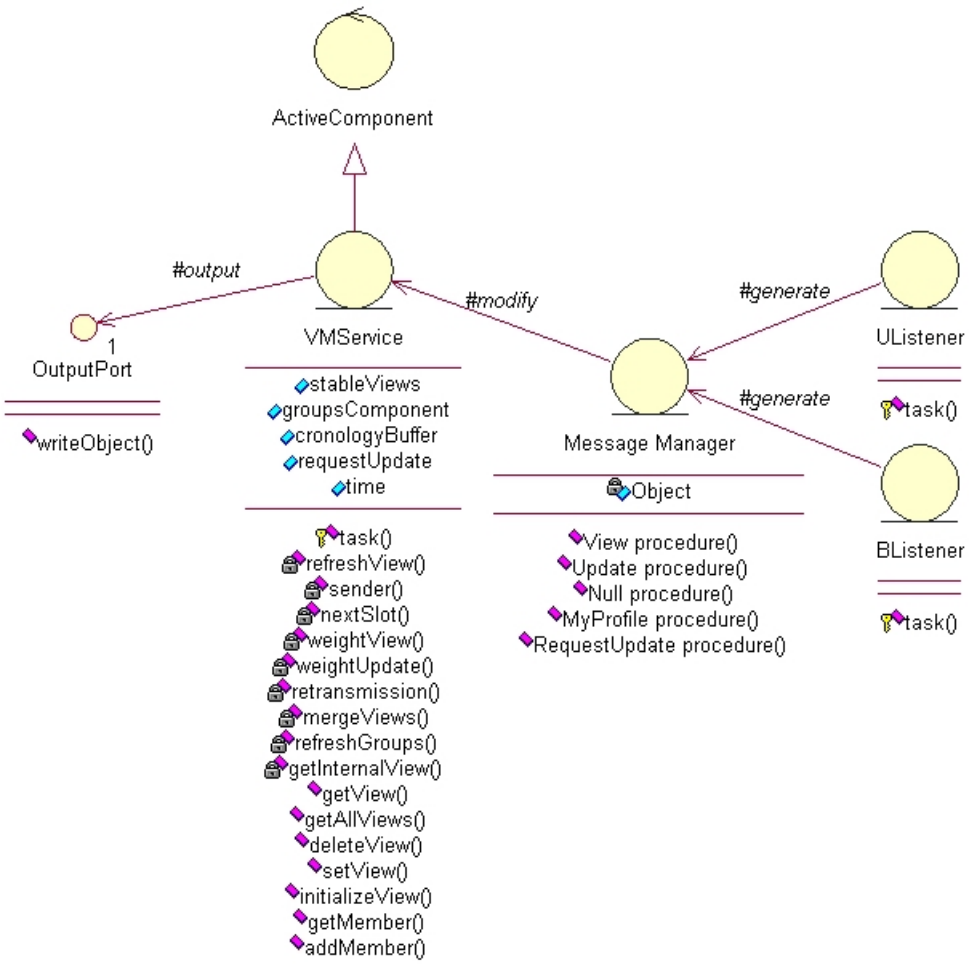

Figura 4.19: Class diagram dell'architettura logica finale del VMS nel caso LME.

Le caratteristiche fondamentali del modulo VMService sono le seguenti:

- Il servizio risulta adattativo rispetto alla dinamicità della topologia di rete. Maggiore è la variabilità della rete più stretto sarà l'intervallo di tempo tra due attivazione consecutive del task, e viceversa.

- Il servizio, prima di ogni invio periodico, misura la dimensione in byte di tutti gli oggetti inviati mediante due funzioni apposite, e sceglie per l'invio sempre la soluzione a minor impatto sulla banda.

- Il servizio implementa un'euristica decisionale che decide, in base a particolari criteri,

se soddisfare o meno le richieste di ritrasmissione, privilegiando un risparmio di banda piuttosto che una copertura[3] completa dei nodi.

Per quanto riguarda invece i metodi implementati nel componente si individuano due gruppi: un primo gruppo rappresentato dai metodi pubblici che rispecchiano e realizzano le primitive di sistema dichiarate in precedenza, ed un secondo gruppo rappresentato da quelli privati necessari al corretto funzionamento interno del modulo VMService. Tra questi ultimi i principali sono il `refreshView`, che ad ogni invocazione controlla lo stato dei gruppi mediante il PENS, il `sender` che si occupa dell'invio degli aggiornamenti periodici, il `nextSlot` che determina in base alla situazione topologica l'istante di attivazione successivo per ogni gruppo ed il `retransmission` che si occupa delle ritrasmissioni di aggiornamenti persi. Sono presenti anche ora, come nel caso ME, anche un metodo per il merge delle viste che si riferiscono allo stesso gruppo (`mergeViews`) ed un metodo per l'eliminazione dalle proprie strutture dati delle entry relative ad LME non più raggiungibili (`refreshGroups`).

4.3.5.1 Il metodo task

Questo metodo realizza il servizio vero e proprio invocando in sequenza, ad ogni sua attivazione, i metodi per la gestione dei gruppi e per l'invio degli aggiornamenti. Dato che un LME può gestire contemporaneamente più gruppi, ad ogni sua attivazione il task deve svolgere le funzioni di management per ogni vista gestita e deve farlo secondo l'istante di attivazione opportuno, scelto in base ad un algoritmo realizzato dal metodo `nextSlot`. Pertanto il task ad ogni attivazione deve verificare quale gruppo deve essere gestito in quell'istante controllando la struttura dati time, ed in seguito invocare tutta la serie di metodi specifici per il trattamento.

Una volta individuato il gruppo od i gruppi che devono essere mandati in esecuzione, il task invoca la procedura `refreshView` passando come parametro di ingresso il GID del gruppo in questione. In seguito, al termine di tale procedura, il VMService non invia immediatamente gli aggiornamenti individuati, ma, nel caso ci siano richieste di ritrasmissione pendenti, passa il controllo al metodo `retransmission` per il loro trattamento. A questo punto, sempre prima dell'invio, invoca il metodo di controllo `refreshGroups` per la verifica della presenza di tutti gli LME conosciuti in precedenza con la relativa eliminazione nel caso non siano più visibili.

Giunti a questo livello di computazione sembrerebbe di essere in possesso di tutte le informazioni necessarie per il corretto invio degli aggiornamenti periodici, ma manca ancora un passo fondamentale prima dell'invocazione del metodo `sender`. In particolare è necessario interrogare un ulteriore componente di AGAPE, al fine di ottenere l'autorizzazione alla trasmissione. Questo componente è il *View Controlling Service* (VCS) che, mediante l'esecuzione di un'euristica decisionale, conferma o meno al VMS l'invio degli aggiornamenti. Il ragionamento alla base del VCS è questo:

[3]Il termine è utilizzato per indicare il numero di nodi del gruppo che possiedono la vista coerente in un determinato istante. Pertanto il modulo in particolari casi sceglie di non ritrasmettere lasciando il nodo richiedente con vista incoerente a vantaggio di un maggiore risparmio di banda.

In una località, se sono presenti più nodi LME, non è necessario che tutti trasmettano le proprie viste, ma solamente quelli attraverso i quali si ottiene una copertura minima dei nodi.

Si rinuncia ancora una volta ad avere una copertura totale della topologia in favore di un risparmio di banda.

Se il VCS da l'autorizzazione alla trasmissione, il task invoca il metodo sender per l'invio periodico, altrimenti salta tale fase limitandosi a mantenere aggiornate le proprie strutture dati. Al termine di questa computazione il task invoca il metodo `nextSlot` per il settaggio del successivo istante di attivazione per quel determinato gruppo ed infine esegue la procedura `mergeViews` per la costruzione della vista finale fusa di tutte le viste possedute per quel gruppo.

La Figura 4.20 mostra l'esecuzione del metodo task.

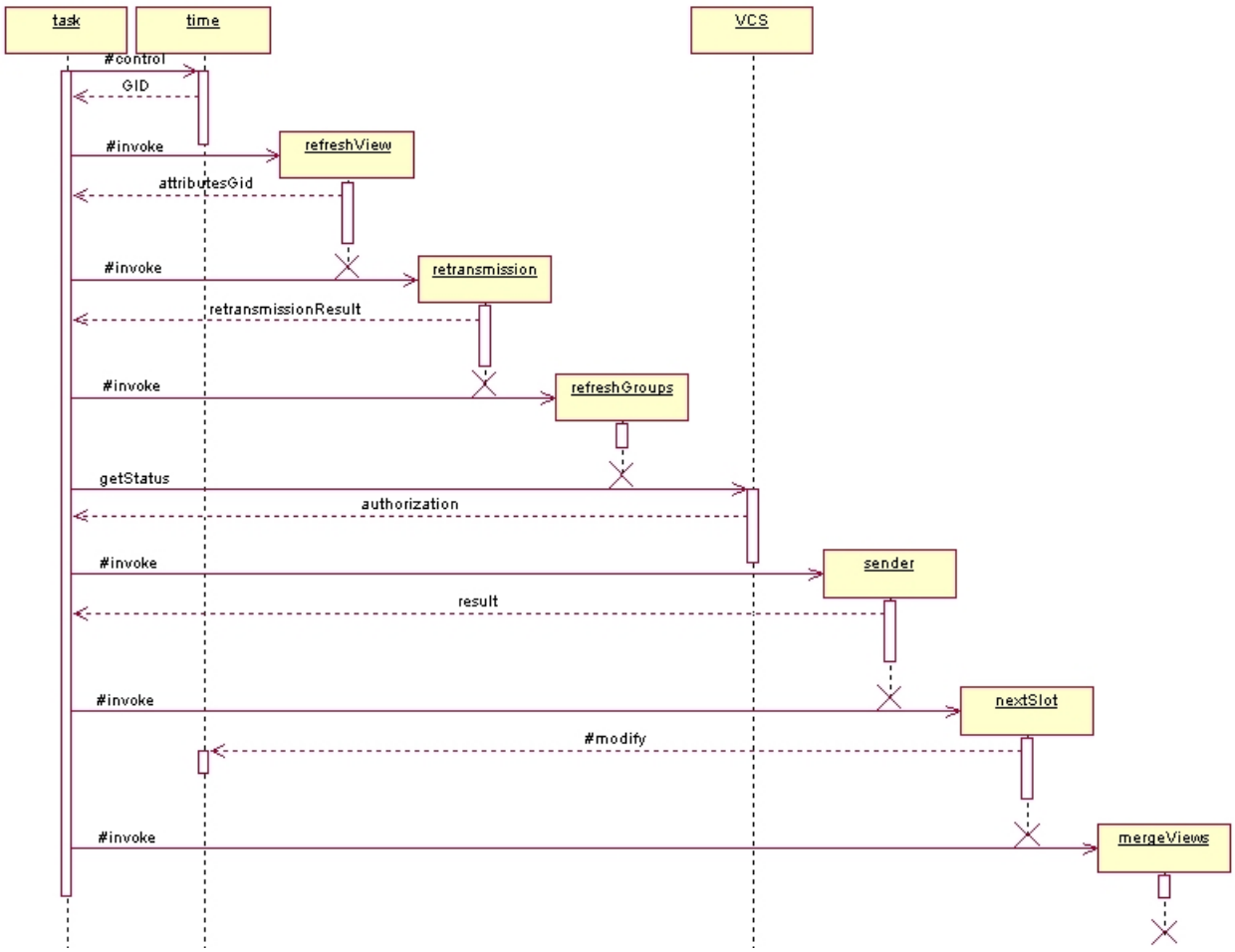

Figura 4.20: Interaction diagram del metodo task.

4.3.5.2 Il metodo refreshView

Tale procedura svolge l'importante funzione di aggiornare la vista di un gruppo gestito mediante il controllo della situazione topologica. Essa infatti interroga il PENS del livello sottostante del middleware richiedendo gli identificatori dei membri presenti in quel determinato istante in località. A questo punto, mediante questo elenco di membri, provvede ad aggiornare la vista aggiungendo i nuovi membri appena scoperti ed eliminando quelli non più presenti.

Questo metodo svolge anche l'importante funzione di creazione degli oggetti `View`, `Update` e `Null` che saranno poi successivamente inviati dalla procedura `sender`. Pertanto a seconda dei casi costruisce gli oggetti `EntryView` ed `EntryUpdate` e li inserisce negli oggetti opportuni. Inoltre per conoscere i destinatari di ogni tipo di oggetto, esso costruisce anche delle liste con le informazioni dei nodi riceventi suddivise in base al tipo di oggetto, le quali saranno poi utilizzate dal metodo sender per il corretto invio dei tre tipi di oggetti.

Dato che tale metodo deve restituire tutta una serie di oggetti di utilizzo delle procedure successivamente invocate dal task, esso costruirà anche un nuovo oggetto contenente tutte le informazioni ricavate e lo restituirà al metodo task. Questo oggetto contiene tutti gli oggetti create durante l'esecuzione più altre informazioni di controllo per la corretta esecuzione delle procedure `sender` e `retransmission`.

La Figura 4.21 mostra l'esecuzione del metodo nel dettaglio.

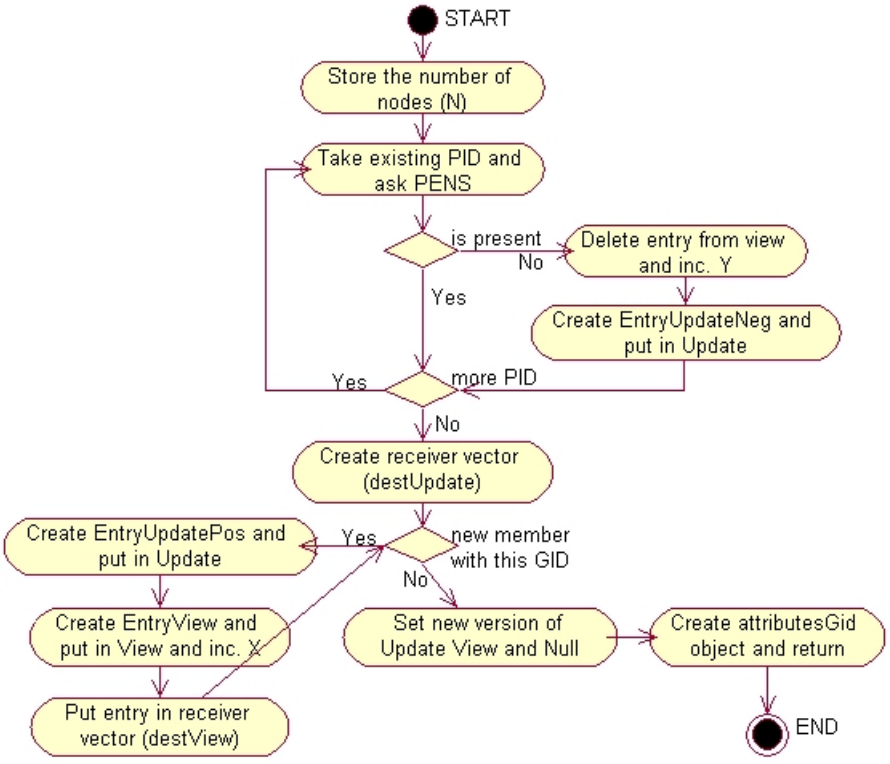

Figura 4.21: State diagram del metodo refreshView.

L'oggetto `attributesGid` rappresenta l'oggetto restituito al task contenente tutte le informazione sull'esecuzione. In particolare esso contiene: il GID del gruppo al quale si riferisce (`GID`), la versione della vista (`version`), gli oggetti View ed Update appena costruiti (`View ed Update`), i tre vettori contenenti l'elenco dei destinatari dei tre oggetti (`destView`, `destUpdate` e `destNull`), tre contatori che rappresentano rispettivamente il numero di nuovi nodi, il numero di quelli usciti ed il numero totale dei nodi attualmente presenti (`X`, `Y` ed `N`).

La Figura 4.22 mostra gli attributi di questo oggetto.

Figura 4.22: Class diagram dell'oggetto attributesGid.

4.3.5.3 Il metodo retransmission

Una volta individuati i cambiamenti della topologia mediante l'invocazione del metodo appena descritto, non si passa immediatamente alla trasmissione degli aggiornamenti ma è necessario controllare se sono state ricevute richieste di ritrasmissione. Questo tipo di controllo deve essere effettuato prima dell'invio, in quanto l'algoritmo di ritrasmissione deve conoscere, nel caso debba ritrasmettere, quale tipo di messaggio tale nodo deve ricevere. Infatti nel caso in cui l'host debba ricevere una vista come aggiornamento periodico, l'algoritmo rifiuterebbe la richiesta in quanto risulterebbe superflua.

Questo algoritmo svolge un ruolo cruciale nel mantenimento di un buon utilizzo di banda, ai fini del risparmio di batteria sia del nodo gestore sia dei membri del gruppo. La linea guida adottata pertanto nella progettazione di tale algoritmo è sempre quella del risparmio di banda, ritrasmettendo solo quando strettamente necessario ed utilizzando ancora una volta sempre il minor numero di byte possibile.

Il punto di forza dell'algoritmo sta nell'implementazione di euristiche decisionali per la valutazione del soddisfacimento o meno della richiesta. Attraverso queste scelte, il metodo può decidere di rispondere alla richiesta solamente nel caso in cui un nodo è ritenuto *"sufficientemente rilevante"* per il bene di tutta la rete. Ad esempio nel caso di un nodo periferico oppure di un nodo con basso livello di batteria residua, l'algoritmo può scegliere di non soddisfare la richiesta in quanto il nodo è poco interessato al gruppo o potrebbe

scomparire in breve tempo. In questi casi infatti utilizzare risorse quali banda e batteria per riportare il nodo ad uno stato di coerenza, risulterebbe dispendioso senza grossi vantaggi per il gruppo. Quello che esegue il metodo `retransmission` pertanto è un *trade-off* tra il numero di nodi con vista coerente e l'utilizzo del minor impiego di banda possibile, rinunciando alla completa consistenza dell'intera rete.

Le richieste che giungono all'LME vengono mantenute in un buffer di memoria che viene svuotato ad ogni istante di attivazione del task, pertanto non necessita di particolari gestioni per limitare la sua dimensione. Ogni richiesta presente in tale buffer rappresenta un nodo, in quanto nel caso in cui un membro debba richiedere più aggiornamenti persi raggruppa i numeri di versione includendoli nello stesso messaggio `RequestUpdate`. Alla sua invocazione il metodo retransmission riceve come parametri di ingresso il vettore delle richieste relative ad un determinato gruppo, la struttura dati `cronology` contenente la cronologia delle trasmissioni e l'oggetto `attributesGid` che contiene il risultato della procedura `refreshView`. In uscita invece restituirà un'oggetto contenente tutti i risultati della computazione, ed è chiamato `RetransmissionResult`.

In Figura 4.23 viene mostrato il diagramma degli stati dell'esecuzione di tale algoritmo. Come per le procedure viste in precedenza anche ora si suddivide la sequenza di stati in una prima parte di *controllo*, per verificare se la richiesta può essere soddisfatta, ed una parte di *trattamento* vero e proprio della richiesta.

Come primo controllo si effettua il conteggio del numero di richieste in confronto con il numero totale dei nodi. Dato che ogni richiesta in buffer rappresenta un nodo, se la percentuale delle domande è maggiore di un livello percentuale minimo `Z` del numero totale dei nodi, la richiesta può essere scartata in quanto l'LME si accontenta di una copertura dei nodi parziale. Al di sopra di questo livello di percentuale di riferimento, il nodo ritiene essere in una situazione di rete soddisfacente e non intende sprecare risorse per aumentare il livello di copertura. Come mostrato nel diagramma quindi, se ci si trova in questo caso, la richiesta viene scartata e, se non ci sono altre richieste pendenti, si esce dall'algoritmo.

Se ci si trova invece al di sotto del livello minimo di copertura `Z`, si prosegue con la parte di controllo. Si verifica pertanto se il nodo mittente della richiesta è ancora presente, controllando la sua presenza nella vista appena modificata dal metodo `refreshView`. Se è ancora presente in topologia si prosegue controllando il suo livello di batteria residuo in quanto, se troppo basso, rappresenterebbe per la ritrasmissione un invio poco utile a causa della sua imminente scomparsa dalla rete. Pertanto si sceglie un livello di batteria minimo `B` al di sopra del quale l'algoritmo soddisfa sempre la richiesta.

A questo punto la fase di controllo prosegue con la verifica della distanza di tale nodo dall'LME. Si ritiene infatti che i nodi periferici siano poco interessati alle attività del gruppo e pertanto, i nodi al di sopra di un certa distanza, intesa come numero di hop, non ricevono risposta alla loro richiesta di ritrasmissione. Si sceglie anche in questo caso un livello minimo di distanza `K` al di sotto del quale avviene la ritrasmissione.

Ora, come ultima operazione di controllo, si verifica che tipo di aggiornamento periodico debba ricevere il nodo richiedente, in quanto, se si tratta di una vista, risulterebbe alquanto inutile l'invio di precedenti aggiornamenti dato che una vista sovrascrive quella precedentemente memorizzata. Pertanto se l'aggiornamento è una vista si scarta la richiesta mentre se non lo è si passa alla fase di trattamento.

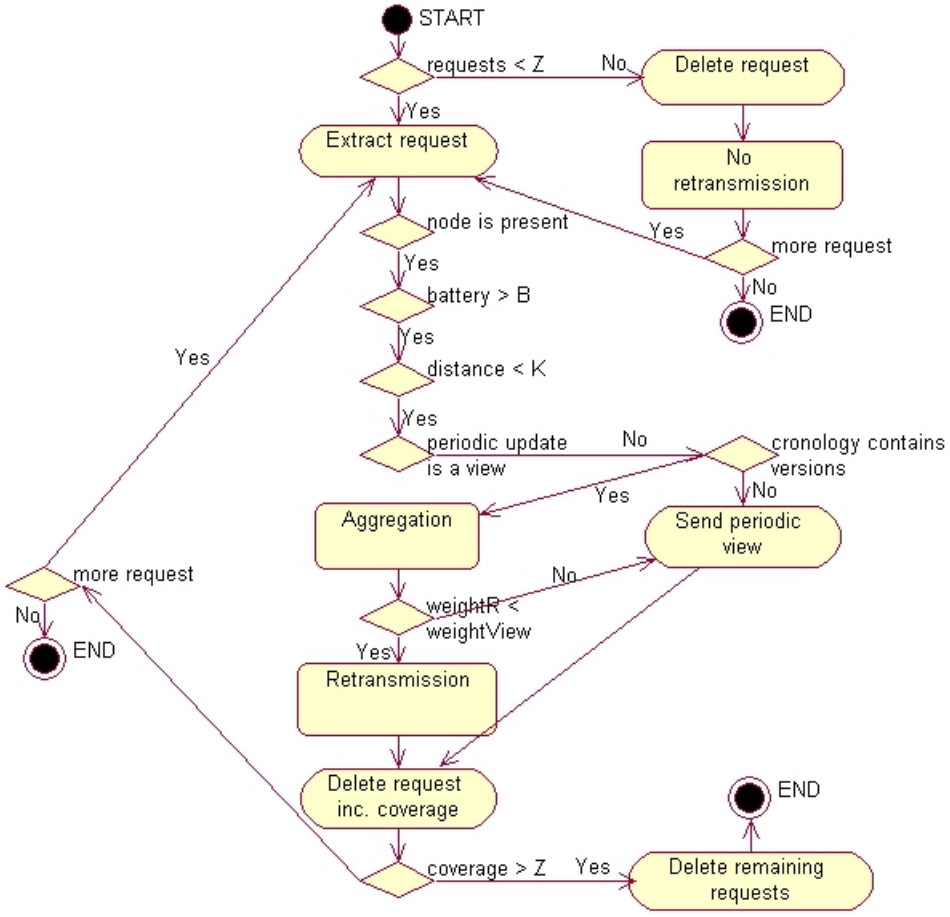

Figura 4.23: State diagram del metodo retransmission.

Si controlla quindi se tutte le versioni richieste sono presenti nel buffer `cronology` ed in caso positivo si passa il controllo ad una fase di *aggregazione*. Questa fase è necessaria sempre ai fini di un risparmio di banda in quanto, nel caso di ritrasmissione di più oggetti, l'aggregazione di oggetti dello stesso tipo in uno unico risulta meno dispendioso in termini di byte inviati. Una volta aggregati gli oggetti, si effettua un ulteriore controllo sul numero di byte necessari per la trasmissione dell'oggetto costruito e quelli necessari per l'invio della vista attuale, in quanto anche questo algoritmo, sceglie sempre la soluzione a minor consumo di banda. Pertanto se il peso della ritrasmissione `weightR` è minore del peso della vista attuale `weightView` si ritrasmette altrimenti si invia la vista attuale. Giunti a questo livello dell'algoritmo quindi si incrementa la copertura di un'unità e, nel caso è stato raggiunto il livello minimo si esce eliminando le eventuali richieste ancora pendenti, altrimenti si ritorna alo stato iniziale e si reitera il procedimento.

Nel caso in cui invece le versioni degli aggiornamenti non sono tutte presenti nel buffer `cronology`, si invia come aggiornamento periodico la vista attuale, semplicemente aggiungendo l'entry del nodo tra i destinatari della vista `destView`.

Al termine dell'esecuzione il metodo costruisce l'oggetto `RetransmissionResult` con

tutti i risultati della computazione e lo restituisce al task di esecuzione per l'invocazione del metodo `sender`.

E' bene soffermare l'attenzione sulla fase di aggregazione. Se l'oggetto da ritrasmettere è unico oppure sono più di uno ma di tipo diverso, tale fase viene omessa proseguendo nella computazione dell'algoritmo. Nel caso in cui invece, ci siano più aggiornamenti dello stesso tipo, si sceglie di aggregare inviando il tutto come un unico aggiornamento e demandando poi al lato client la ricostruzione corretta degli oggetti originali. La Figura 4.24 mostra l'esecuzione di questo procedimento.

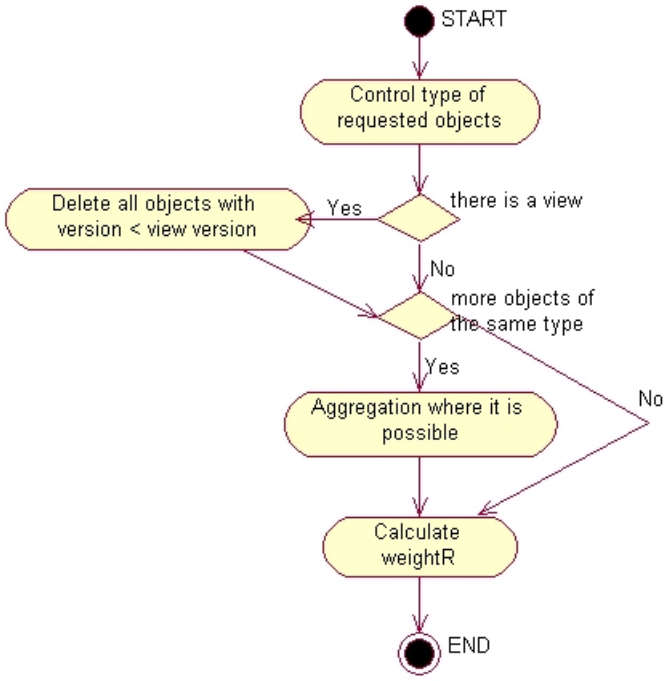

Figura 4.24: State diagram della fase di aggregazione.

L'algoritmo analizza prima di tutto il tipo di aggiornamenti richiesti dopo di che, nel caso ci siano delle viste da rinviare, esegue un filtraggior di tali richieste. Questa operazione viene eseguita al fine di preservare ancora una volta il numero di byte da trasmettere. Nel caso della presenza infatti di una vista con numero di versione superiore a quello degli altri aggiornamenti da ritrasmettere (di qualunque tipo), risulterebbe superfluo l'invio di questi ultimi in quanto l'arrivo sul lato client della vista riporterebbe immediatamente il nodo ad uno stato di coerenza sovrascrivendo gli aggiornamenti precedenti.

A questo punto quindi controlla se ci sono oggetti dello stesso tipo e nel caso ci siano li aggrega dove possibile. E' da notare che l'aggregazione coinvolge solamente Update e Null e non viste in quanto, per i motivi appena descritti, verrebbe inviata solamente la vista con numero di versione maggiore. Nel caso dei Null si aggrega semplicemente inserendo nel vettore delle versioni i numeri di versione degli oggetti raggruppati. Un caso particolare è invece rappresentato dagli Update. L'unione di questi infatti necessita maggior computazione sul lato client in quanto è necessario identificare a che numero di

versione corrisponde ogni entry al fine di mantenere la cronologia degli avvenimenti. Per poter fare questo è necessario appendere ad ogni entry dell'Update, sia negativa sia positiva, la versione alla quale si riferisce in modo che il ricevente possa ricostruire l'Update originale. Viene introdotto pertanto un nuovo tipo di entry, chiamata `EntryUpdateVersion` (vedi Figura 4.25), che possiede come attributi l'entry standard di un update ed il numero di versione.

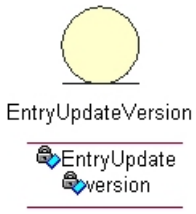

Figura 4.25: Class diagram dell'oggetto EntryUpdateVersion.

Al termine della fase di aggregazione viene calcolato il peso dell'invio `weightR` e si riprende l'esecuzione dell'algoritmo di ritrasmissione.

4.3.5.4 Il metodo sender

Questo metodo è il responsabile dell'invio degli aggiornamenti costruiti durante la fase di refresh della topologia e modificati durante la fase di ritrasmissione. L'unico risultato dell'esecuzione del metodo precedente che si ripercuote su questo metodo è infatti l'eventuale modifica dei vettori dei destinatari, pertanto, in ingresso al `sender` avremo, il risultato della `refreshView` (`attributesGid`), il risultato della `retransmission` (`RetransmissionResult`) ed il buffer della cronologia per l'inserimento (`cronology`). Il risultato di questa operazione sarà la restituzione della `cronology` modificata.

Anche questo metodo è in grado di valutare sempre il miglior tipo di invio eseguendo un controllo sul numero di byte da inviare nei vari casi. Per l'invio degli aggiornamenti fa uso, come indicato dall'architettura logica, delle primitive di livello inferiore ed in particolare delle porte di uscita unicast e broadcast. Gli invii infatti sono differenziati a seconda delle variazioni topologiche, e può accadere che in una stessa trasmissione alcuni membri necessitino di un tipo di oggetto ed altri di un tipo diverso, nel qual caso si utilizzerebbe un invio unicast, ed anche casi in cui tutti i nodi del gruppo siano destinatari di uno stesso oggetto e quindi in tal caso si utilizzerebbe un invio di tipo broadcast. Questa discriminazione delle trasmissioni aiuta maggiormente la preservazione della banda.

La Figura 4.26 mostra il diagramma degli stati di tale metodo. Si preleva pertanto l'`attributesGid` passato in ingresso e si estraggono gli oggetti da inviare e i destinatari per ogni tipo di oggetto. Come mostrato dal diagramma di classe di tale oggetto esso contiene anche tre contatori necessari a questo metodo per valutare gli avvenimenti verificatisi in topologia, e sono `X`, `Y` ed `N`. Se il sender rileva che non vi sono stati cambiamenti in topologia ($X == Y == 0$), esso decide di inviare attraverso la porta broadcast un oggetto Null a tutte le entry contenute in `destNull`. Nel caso in cui invece tale uguaglianza non

è verificata, il metodo deve calcolare e valutare il peso di due funzioni `U` e `V` al fine di constatare il tipo di invio migliore. Queste due funzioni calcolano il peso in termini di byte da inviare rispettivamente di un invio di Update o Update più View a seconda delle modifiche della topologia, e di un invio di View a tutti i membri della vista.

$$U = (N - Y)weightUpdate + (X)weightView \tag{4.1}$$

$$V = (N - Y + X)weightView \tag{4.2}$$

Queste due funzioni rappresentano il numero totale complessivo di byte da inviare nei due casi. Infatti la 4.1 somma il numero di byte dell'invio di update a tutti i destinatari e l'eventuale invio di viste ad eventuali nodi destinatari. Nel caso ci siano solo partenze la X risulta nulla e si hanno solo update mentre nel caso di partenze e arrivi al gruppo la X e la Y saranno maggiori di 0 e la U sommerà entrambi i fattori. La 4.2 somma sempre ed in ogni caso i byte da inviare nel caso di vista a tutti i membri in broadcast. Pertanto moltiplica il peso attuale della vista per i destinatari di tale vista, che sono i nodi presenti (N) meno quelli partiti (Y) più quelli arrivati (X).

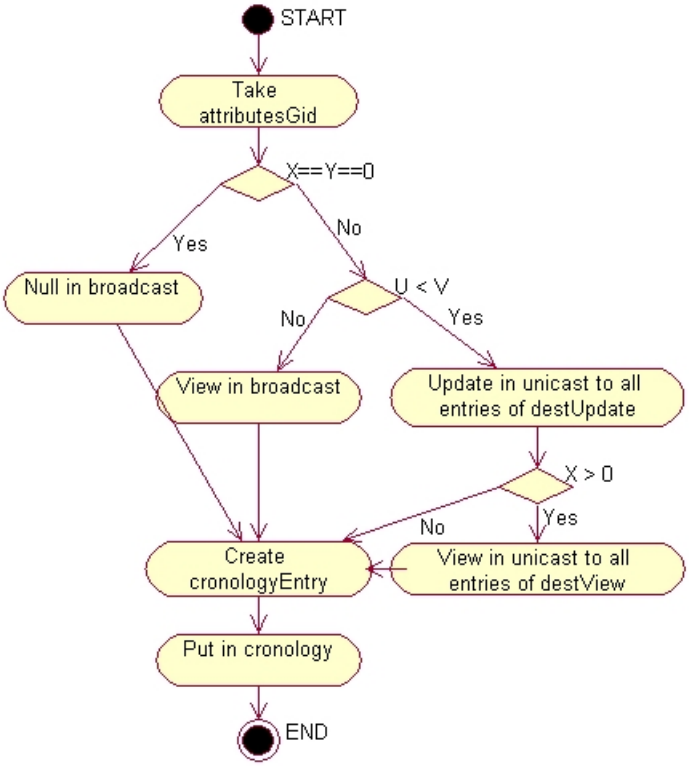

Figura 4.26: State diagram del metodo sender.

A questo punto, una volta effettuato l'invio, è necessario memorizzare l'avvenimento nel buffer cronology al fine di poter rinviare l'aggiornamento perso nel caso di richiesta da parte di qualche nodo del gruppo. Per fare questo è necessario salvare in memoria il tipo

di oggetto inviato ad ogni destinatario della trasmissione, pertanto il `sender` costruisce un oggetto `cronologyEntry` contenente tutte queste informazioni. In particolare esso conterrà i tre oggetti `View`, `Update` e `Null` e gli eventuali destinatari di ogni oggetto, cioè `destView`, `destUpdate` e `destNull`. Una volta costruito tale oggetto viene inserito all'interno del buffer `cronology` e tale buffer restituito al metodo task per poter proseguire con l'esecuzione.

Un cenno particolare va fatto riguardo il metodo di gestione del buffer `cronology` in quanto, in ambito mobile, risulta significativo utilizzare al meglio la risorsa memoria data la limitata disponibilità di questi dispositivi. In Appendice A viene svolta un'accurata analisi della soluzione adottata e di altre politiche di *replacement* per la gestione della cache. La politica di sostituzione implementata nel task è la standard FIFO con dimensione pari a 15 entry.

4.3.5.5 Il metodo nextSlot

Come abbiamo accennato in precedenza, una caratteristica importante del componente VMS è la sua capacità di adattarsi alla dinamicità della topologia. Dato che il task viene attivato secondo certi intervalli di tempo, il metodo adottato per reagire alle variazioni della rete è quello di determinare ad ogni esecuzione del task, l'intervallo di attivazione successivo osservando i cambiamenti verificatisi in quell'istante di attivazione. Il task esegue questo compito invocando il metodo `nextSlot`.

L'idea alla base del metodo è la seguente:

> Maggiori sono i cambiamenti topologici più elevata dovrà essere la velocità di esecuzione del task, cioè più stretto dovrà essere l'intervallo di attivazione. Se invece i cambiamenti alla topologia sono pochi, la velocità di esecuzione può essere moderata e quindi l'intervallo di attivazione può essere dilatato.

In pratica campionando la topologia più frequentemente, cioè attivare il task a frequenza maggiore, i cambiamenti della stessa saranno minori tra due istanti consecutivi e gli aggiornamenti da trasmettere avranno meno informazioni al loro interno con conseguenti benefici alla banda. Si hanno pertanto meno byte in trasmissione con l'unico contro di avere una maggiore computazione sul lato server, cosa trascurabile dato che di solito il caso LME viene realizzato da nodi a maggiore potenza computazionale. Il metodo nextSlot realizza proprio questa funzione, cioè quella di capire quando è il caso di dilatare o restringere l'intervallo in base alla dinamicità del gruppo. Se siamo in una situazione ad alta dinamicità infatti, al fine di avere piccoli cambiamenti da un'istante all'altro occorre che il VMS si attivi più frequentemente e quindi l'intervallo sia più stretto. Al contrario nel caso in cui la situazione sia più statica, anche con intervalli di tempo maggiori si registreranno piccoli cambiamenti ottenendo quindi poca computazione e pochi byte in trasmissione.

Gli intervalli di tempo variano all'interno di un range limitato superiormente al fine di mantenere sempre una certa reattività della rete rispetto ai cambiamenti. Il range varia da un valore minimo pari a 2 secondi fino ad un valore massimo di 10, dove quest'ultimo rappresenta anche l'intervallo di partenza. In seguito l'algoritmo, per eseguire la scelta dei nuovi intervalli, introduce dei *livelli di dinamicità* attraverso i quali giudicare il livello di

variazione da un'istante al successivo. I livelli di dinamicità vengono calcolati in base alla percentuale dei nodi cambiati nell'overlay di rete in quell'istante e sono valutati in questo modo:

- **Level 1**: cambiamenti minori uguali del 10%.
- **Level 2**: cambiamenti compresi tra il 10% ed il 20%.

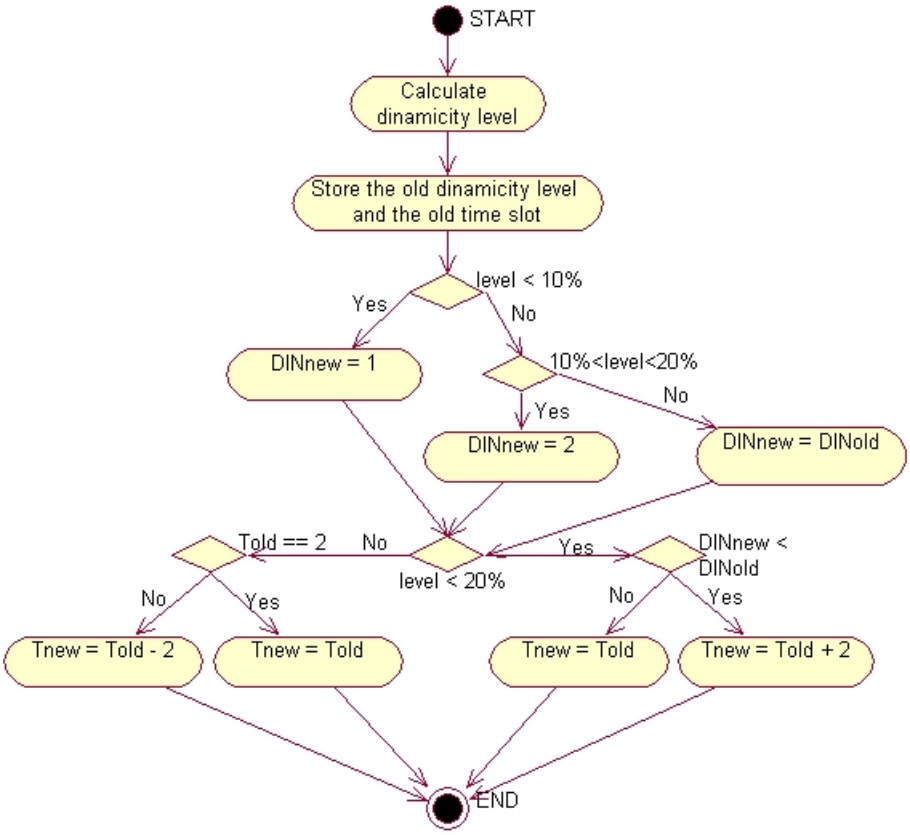

Figura 4.27: State diagram del metodo nextSlot.

Quando i cambiamenti della rete superano il valore massimo del 20% è necessario restringere l'intervallo mentre se si è al di sotto, è necessario eseguire un confronto con il precedente livello di dinamicità:

- **Dinamicità uguale o crescente**: si mantiene l'intervallo uguale al precedente.
- **Dinamicità decrescente**: si dilata l'intervallo precedente.

Le variazioni degli intervalli avvengono sempre nella misura di 2 secondi per volta e non possono mai oltrepassare i limiti del range. La Figura 4.27 mostra l'esecuzione dell'algoritmo. E' da notare che i valori scelti per gli istanti limite e per l'ammontare della

variazione sono stati scelti in base a prove di simulazione e che possono essere cambiati in base ai requisiti applicativi.

Il primo passo dell'algoritmo è quello di memorizzare il precedente livello di dinamicità (`DINold`), il precedente slot time (`Told`) ed il nuovo livello attuale di dinamicità (`level`). A questo punto controlla `level` e nel caso in cui sia minore del 10% setta il nuovo valore di `DINnew` a 1, se compreso tra il 10% ed il 20% a 2 ed invariato negli altri casi. Ora se il livello attuale `level` è maggiore del 20% verifica se è stato raggiunto il *lower bound* del range, cioè 2 secondi. Nel caso in cui quindi `Told` era di 2 secondi non è possibile restringere l'intervallo e lo si lascia immutato, mentre se non si è in questo caso è possibile restringere di 2 secondi. Se invece il nuovo `level` è inferiore al 20% è necessario comparare i due livelli di dinamicità, `DINold` e `DINnew`, per capire se è possibile rilassare l'intervallo oppure se debba essere lasciato invariato rispetto al caso precedente. In pratica i due livelli di dinamicità sono necessari nel caso in cui, dopo aver ristretto l'intervallo, è possibile riprendere un'esecuzione più moderata a causa di un cambiamento di dinamicità di tipo decrescente.

4.4 Implementazione

Il linguaggio di programmazione di riferimento del middleware è Java, pertanto anche l'implementazione del componente VMS avverrà in questo linguaggio, prendendo come riferimento la versione j2sdk1.4.2. In questa sezione verranno esposti i dettagli implementativi degli oggetti e delle classi delineate nella precedente fase di design.

4.4.1 Oggetti serializzabili

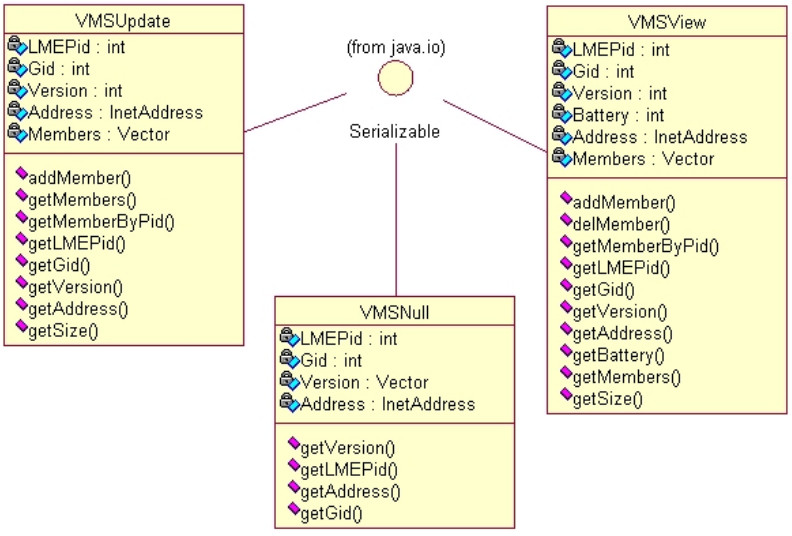

Figura 4.28: Class diagram degli oggetti VMSView, VMSUpdate e VMSNull.

Gli oggetti appartenenti a questo gruppo sono tutti quegli oggetti coinvolti nella trasmissione e pertanto sia loro, sia gli oggetti in esso contenuti, devono essere serializzabili, al fine di poterli trasformare in una sequenza finita di byte per l'invio. Si hanno pertanto i cinque oggetti visti in fase di progetto che rappresentano i cinque tipi di messaggio scambiati tra le entità della rete, più tutti gli oggetti in essi contenuti.

La Figura 4.28 mostra il diagramma di classe dei tre tipi di oggetti primitivi rappresentanti i messaggi per gli aggiornamenti delle viste.

Come si può notare dagli attributi dei tre oggetti, l'unico che necessita di ulteriori trattamenti è il vettore dei membri, in quanto i tipi dei restanti oggetti sono nativi e già serializzabili. Il vettore `members` infatti contiene a seconda dei due casi o delle `EntryView` oppure delle `EntryUpdate`, le quali a loro volta contengono degli oggetti rappresentanti i nodi chiamati `VMSMember`. La Figura 4.29 mostra le relazioni tra questi tipi di oggetti.

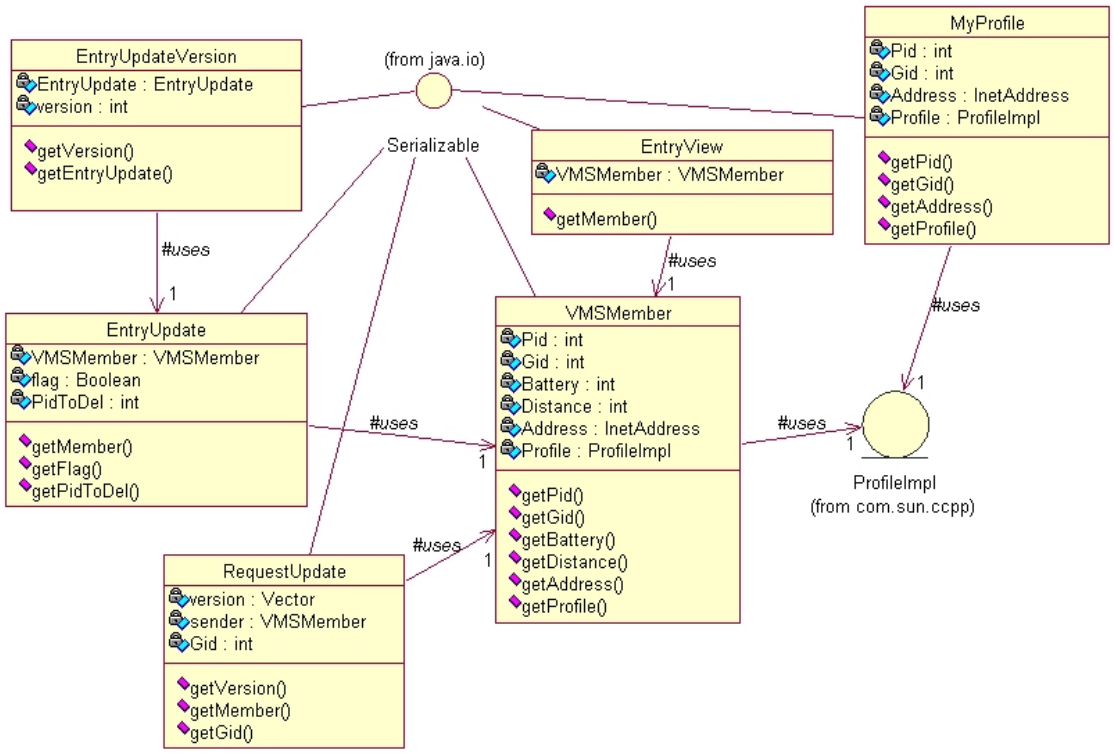

Figura 4.29: Class diagram degli oggetti EntryUpdate, EntryView, EntryUpdateVersion, VMSMember e MyProfile.

L'oggetto EntryView contiene solamente l'oggetto contenente gli attributi di un membro e cioè `VMSMember`. L'oggetto `EntryUpdate` invece contiene, oltre a quest'ultimo, anche un `flag` di identificazione per discriminare gli Update negativi da quelli positivi, e nel caso di Update negativo, l'identificatore `pidToDel` per l'eliminazione di un membro da una vista. L'oggetto EntryUpdateVersion risulta essere un oggetto contenitore in quanto ingloba al suo interno un `EntryUpdate` con il relativo numero di versione al quale si riferisce.

L'oggetto che riporta tutte le informazioni riguardanti un membro del gruppo è l'oggetto

`VMSMember` che contiene in particolare anche il profilo dell'utente. Questo profilo viene creato staticamente a livello applicativo all'inizio della computazione ed è rappresentato dalla classe `ProfileImpl` contenuta nel package `com.sun.ccpp`. Questo oggetto `ProfileImpl` viene utilizzato anche dal messaggio `MyProfile` attraverso il quale un membro può inviare all'LME il proprio profilo.

Infine l'oggetto `RequestUpdate` serve per segnalare all'LME i numeri di versione di aggiornamenti persi, pertanto conterrà al suo interno un vettore contenente tali numeri e l'indirizzo IP del mittente di tale messaggio al fine di poter ricevere una risposta unicast.

4.4.2 Listener e Message Manager

Per la realizzazione degli ascoltatori si utilizzano delle classi che estendono la classe `activeComponent` del package `agape`, in quanto fornisce un metodo task, derivato dall'interfaccia di implementazione `Runnable`, che si attiva ad intervalli di tempo regolari mediante un ciclo while. Dato che gli ascoltatori dovranno essere in grado di rilevare messaggi in arrivo in qualunque momento, si sceglie di non mettere mai in stato sleep questi thread, ma di lasciarli continuamente attivi per tutto il tempo della computazione. Questi due oggetti pertanto costruiscono staticamente al momento della loro attivazione, eseguita mediante il metodo `startListener`, una porta di ingresso di ascolto mediante le primitive fornite dalla classe `NetManager` contenuta nel package `nms.net`.

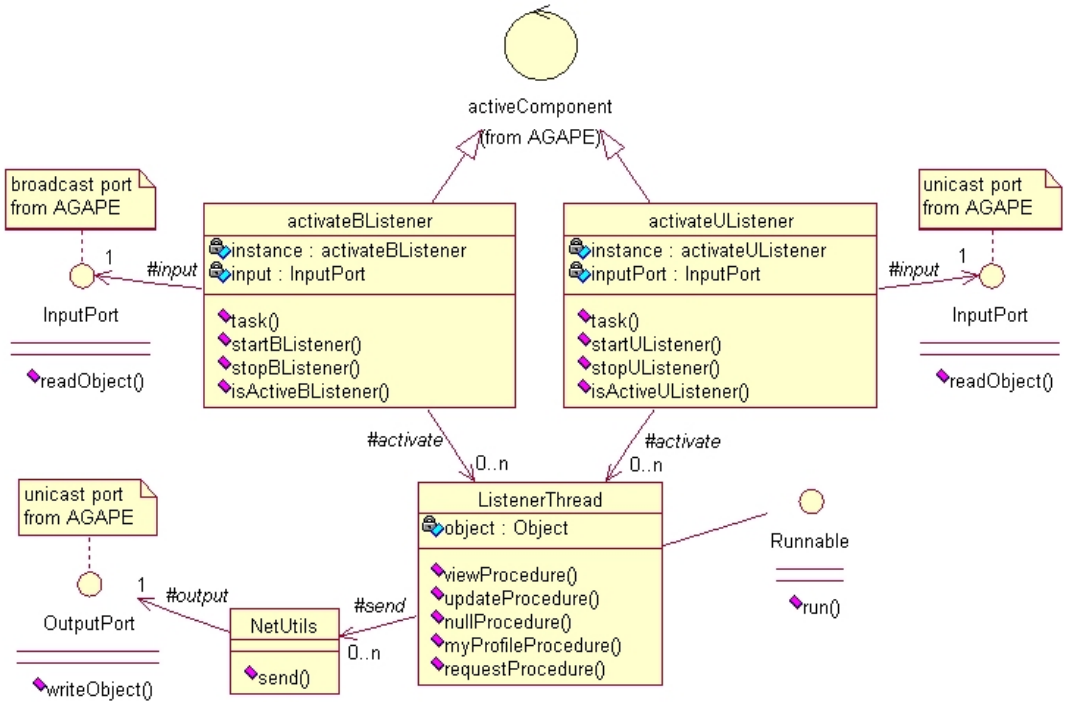

Figura 4.30: Class diagram degli oggetti per il trattamento dei messaggi.

Il primo quindi invoca il metodo `getBCastInputPort` ed il secondo il metodo `getUCastI-`

`nputPort` con numeri di porta differenti, che restituiscono una porta di ascolto rispettivamente broadcast ed unicast.

Ogni volta che ricevono un messaggio i listener attivano il Message Manager che viene implementato da un thread chiamato `ListenerThread` e che ha il compito del trattamento del messaggio. Il messaggio viene passato dai listener nella stessa forma in cui è stato ricevuto demandando al thread di gestione il compito di eseguire un `instanceof` per determinarne il tipo, ricostruire l'oggetto mediante cast esplicito ed invocare la giusta procedura di trattamento. Una volta terminata tale procedura il thread termina l'esecuzione deallocandosi dalla memoria.

Nel caso in cui una delle procedure necessiti di inviare il proprio profilo all'LME mittente del messaggio, il thread utilizza una classe utility chiamata `NetUtils`, che fornisce un metodo `send` per l'invio di messaggi unicast. Questo metodo costruisce una porta di uscita mediante le informazioni passate come parametri, invia l'oggetto invocando il metodo `writeObject` ed, al termine della trasmissione, chiude la porta appena creata. Le informazioni necessarie per una trasmissione sono l'oggetto da inviare, l'indirizzo IP del nodo ricevente e la porta da utilizzare. Naturalmente la porta adottata per l'invio unicast sarà la stessa utilizzata per l'ascolto dal listener unicast.

4.4.3 VMService: LME

L'ostacolo maggiore per l'implementazione del metodo sul lato server è stato quello di mantenere correttamente in memoria le informazioni relative a ciascun gruppo gestito mediante una serie di strutture dati dedicate suddivise per gruppo e per gestore. Risultano molto frequenti infatti operazioni sui gruppi ed è quindi necessario poter estrarre correttamente le informazioni necessarie, modificarle e reinserirle all'interno degli oggetti dedicati per la memorizzazione.

Un LME può gestire più gruppi contemporaneamente, quindi ad ogni attivazione periodica del metodo task, può succedere di dover eseguire tutte le procedure da esso invocate per più gruppi contemporaneamente pertanto sono necessarie delle strutture dati che suddividano i gruppi in base al GID. Inoltre, a causa dell'adattatività del servizio rispetto alla topologia, la cosa viene ulteriormente complicata. Infatti nel caso della gestione di due gruppi aventi dinamicità molto diversa, sarebbe necessario attivare il task per ciascuno a momenti diversi, pertanto è necessaria anche una struttura dati per la memorizzazione dei tempi di attivazione ed un metodo per capire in ogni istante quale sia il gruppo da eseguire ed a quale intervallo di attivazione.

Per quanto riguarda le strutture dati, si è facilmente risolto il problema mediante l'utilizzo di oggetti di tipo `TreeMap` o `Hashtable`, che permettono l'estrazione e l'inserimento immediati di valori mediante il passaggio di una chiave. Inoltre il primo permette anche di mantenere i dati al suo interno ordinati in ordine crescente e pertanto, dove necessaria quest'ultima caratteristica, è stato privilegiato il tipo `TreeMap`.

Per quanto riguarda invece il problema della gestione multipla di gruppi mediante lo stesso task, è stata utilizzata come struttura dati una TreeMap per la memorizzazione di tutti gli intervalli di attivazione ed un metodo di scansione del minimo intervallo per

la determinazione degli slot di attivazione successivi. Tale metodo viene mostrato nel diagramma di Figura 4.31.

Viene indicato con `Time` il `TreeMap` contenente gli intervalli di attivazione. Si parte con un valore predefinito sia per l'intervallo del gruppo `VAL` sia per quello `PERIOD` che indica i secondi di pausa tra due attivazioni successive del task. Vengono settati pertanto all'interno di `Time` i valori del nuovo intervallo per ogni gruppo sottraendo al valore `VAL` impostato dal metodo `nextSlot` il valore di `PERIOD`, che rappresenta il tempo già trascorso. Il valore ottenuto rappresenta il tempo che deve ancora trascorrere prima della sua attivazione.

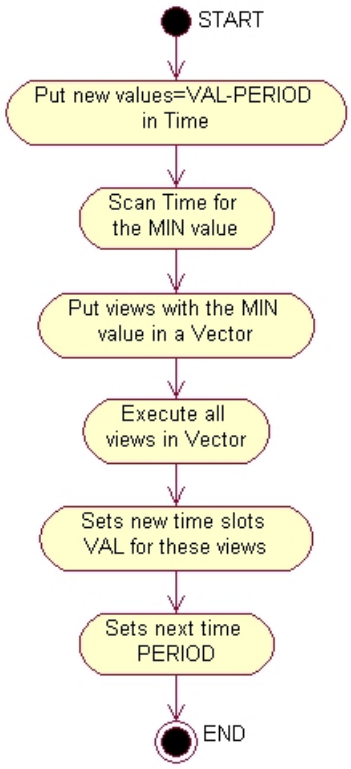

Figura 4.31: State diagram dell'algoritmo per la determinazione del minimo intervallo.

A questo punto viene scansionato `Time` alla ricerca di gruppi aventi `VALUES` a 0 e quindi messi in un vettore per essere eseguiti dal metodo task. Il metodo pertanto esegue tutte i gruppi contenuti in tale vettore, calcola il successivo intervallo di attivazione mediante `nextSlot` e quindi deve impostare il nuovo intervallo di attesa `PERIOD`. Per fare questo determina il minimo valore contenuto in `Time` ed imposta lo *sleep* del Thread pari a questo valore.

Per il buon mantenimento delle informazioni in memoria, oltre alle strutture dati fornite dal linguaggio Java, sono stati introdotti anche ulteriori oggetti che si adeguano meglio alle esigenze del progetto. In particolare è stato utilizzato un oggetto `CronologyEntry` contenente gli oggetti ed i destinatari di un invio periodico, che viene mantenuto in memoria in un buffer gestito con politica FIFO, e richiamato ogni qualvolta un membro lo richieda

come ritrasmissione. E' stato inoltre introdotto un oggetto `ViewComponent` per la memorizzazione di tutte le strutture dati primitive relative a ciascun gruppo e mantenuto in memoria mediante un buffer chiamato `groupsComponent`.

La Figura 4.32 mostra il diagramma di classe di questi due oggetti relazionati con la classe VMService.

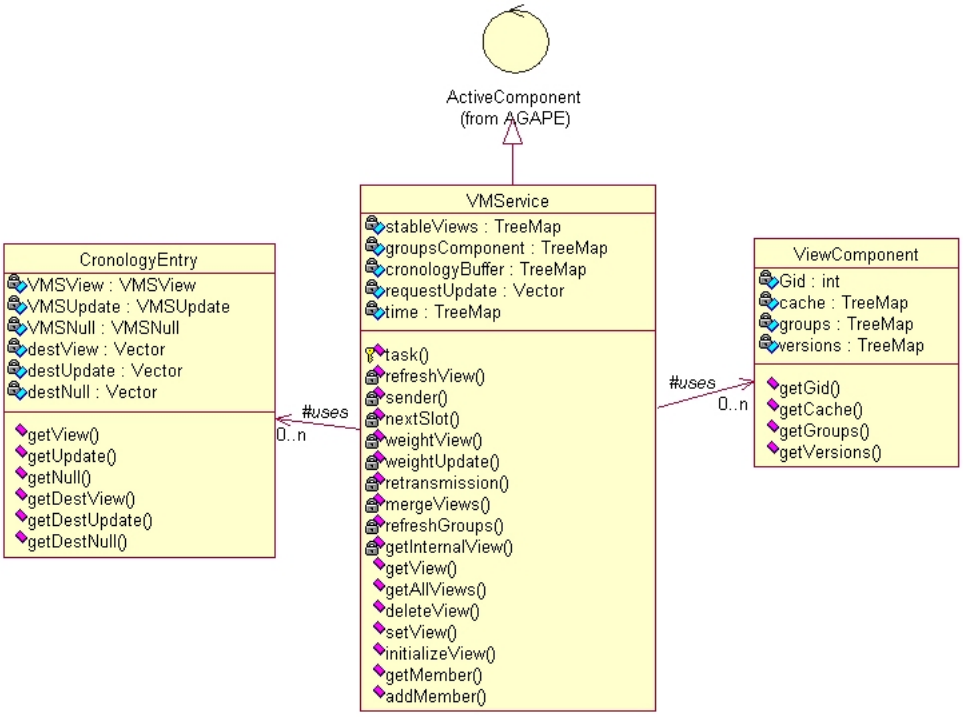

Figura 4.32: Class diagram delle classi ViewComponent e CronologyEntry.

4.4.4 VMService: ME

Per quanto riguarda il `VMService` lato ME, l'implementazione non incontra particolari ostacoli soprattutto per la gestione delle viste, in quanto un nodo di questo tipo può appartenere solamente ad un gruppo per volta. Ora l'unico compito svolto dal `VMService` è quello di mantenere separate le viste e gli aggiornamenti ricevuti da ogni LME presente nel suo raggio d'azione, in modo tale da fornire la vista fusa mediante l'operazione di merge. Al fine di mantenere separate queste informazioni la classe utilizza ancora una volta le strutture dati fornite dal linguaggio ed in particolare la `TreeMap`. All'interno di essa vengono memorizzate le viste in base al pid dell'LME mittente in modo tale da recuperarle immediatamente all'occorrenza utilizzando quest'ultimo come chiave per l'estrazione.

Capitolo 5

Testing

Lo scopo di questa sezione è quello di determinare ed analizzare le performance del VMS, in termini di banda occupata, intervallo di attivazione, copertura, gestione della cache e latenza ai cambiamenti della rete.

Il testing di applicazioni collaborative in ambito ad hoc, risulta essere una situazione molto ostica in quanto, per verificare il reale comportamento su macchine reali, è necessario disporre di un'adeguato numero di dispositivi, cosa non sempre realizzabile. A questo fine viene in aiuto una tecnica di testing diversa dallo scenario reale, ma che mantiene un livello di accuratezza della misura molto prossimo ad esso. Questa è la cosiddetta tecnica dell'*emulazione*.

La ragione per cui questo metodo viene definito con il termine di emulazione, sta nel fatto che il *testbed* di collaudo è una rete LAN non wireless, in cui viene emulato appunto il movimento dei nodi attraverso dei tool specifici. Infatti, dato che il target del progetto è l'ambito ad hoc, non è sufficiente testare il middleware su macchine fisse ma è necessario anche che queste siano in movimento provocando situazioni tipiche del mondo wireless come il roaming e le continue disconnessioni. Lo scenario finale risultante quindi è si un'emulazione, ma non troppo discostante dalla situazione reale. La ragione per cui si effettua un'emulazione piuttosto che una prova vera e propria con dispositivi mobili è semplicemente logistica, a causa dell'indisponibilità di un consistente numero di dispositivi wireless impiegabili.

L'emulazione si pone pertanto in un livello intermedio tra uno scenario simulato ed il testbed reale raggiungendo un buon compromesso tra il costo del test e l'accuratezza dei risultati ottenuti.

5.1 Applicazione di test

Al fine di valutare e testare il corretto funzionamento del middleware ed in particolare del VMS, è stato implementata un'applicazione di test che attiva la collaborazione improvvisa di vigili del fuoco co-locati in uno scenario semplificato di soccorso.

L'applicazione crea un unico gruppo di soccorso al quale si uniscono poi tutti i vigili del fuoco presenti nell'area emulata. Tale area verrà creata in seguito, durante la configurazione del testbed, mediante il tool `setdest` ed ha una superficie di $1Km^2$. L'applicazione aggrega

i vigili del fuoco nell'unico gruppo presente nell'area e fornisce loro la visibilità dei colleghi, permettendo la comunicazione di istruzioni o immagini come mappe o planimetrie di edifici mediante un semplice scambio di messaggi.

Il vigile del fuoco tipicamente deve prendere importanti decisioni in breve tempo basandosi su informazioni incomplete. Mediante lo scambio di informazioni con i colleghi colocati, il vigile del fuoco può facilitare e migliorare le sue scelte. Il vigile del fuoco infatti può, mediante l'applicazione basata su AGAPE, improvvisamente chiedere ai colleghi tutti i dettagli necessari per avere una visione chiara dello scenario di intervento. La possibilità di scambiarsi messaggi scritti è un vantaggio notevole per i soccorritori, che si trovano spesso ad operare in situazioni molto rumorose e con l'esigenza di essere in visibilità reciproca.

L'applicazione utilizza dei profili compatibili con il modello CC/PP per descrivere i vigili del fuoco, i dispositivi utente e le proprietà del gruppo. Il profilo di un vigile del fuoco include il suo nome ed il grado, il profilo del dispositivo rappresenta le caratteristiche del dispositivo di accesso, mentre il profilo del gruppo identifica l'obiettivo della squadra di vigili. I vigili del fuoco di alto grado come i capitani ed i sottotenenti, operano mediante laptop equipaggiati con piattaforma Linux e con la release di AGAPE adatta per dispositivi ricchi. Questi rappresenteranno gli LME gestori del gruppo. Tutti gli altri vigili di grado inferiore, operano sul campo mediante PDA con sistema operativo Linux e la release di AGAPE per i dispositivi con vincoli sulle risorse.

Si è scelto dunque di inserire nello scenario di soccorso 3 ufficiali di alto rango che realizzano gli LME per la gestione del gruppo, e 27 vigili del fuoco pronti ad intervenire che rappresentano gli ME. In particolare degli LME solamente uno (ad esempio può essere il capitano) crea il gruppo di lavoro, gli altri, una volta individuato il gruppo ed essersi uniti, iniziano a loro volta a gestirlo. All'inizio dell'emulazione il capitano crea immediatamente il gruppo di soccorso ed inizia a propagare le viste. Gli altri due ufficiali di alto grado che realizzano gli LME, non appena entrano nella località del capitano, si uniscono al gruppo mediante l'operazione di join ed iniziano a gestirlo a loro volta. Con il trascorrere del tempo si uniscono anche tutti gli altri vigili e continuano a collaborare per tutta la durata dell'emulazione, cioè dell'operazione di soccorso.

La prima cosa da effettuare durante l'esecuzione dell'applicazione è quella di attivare tutti i componenti attivi del middleware. Questi vengono attivati dalla classe `Test` mediante il suo metodo di attivazione `startTest`, che è l'unico metodo invocato dal main.

Listing 5.1: Test/Main.java

```
25 public class Main{

27         public static void main(String[] args) throws IOException{

29                 Test.startTest(); //Attiva il test e tutti i servizi di AGAPE
                   System.out.println("\nTEST: Test started.");
31
           }//main
33
}//Main
```

Il metodo `startTest` attiva rispettivamente il PENS, il PS, il VMS ed il J/LMS, nonchè il componente attivo della classe stessa.

Listing 5.2: Test/Test.java

```
/**
 ** Starts the active component.
 */
public static void startTest(){

        if(!isActiveTest()){
                PENService.startPENS();
                PService.startPS();
                VMService.startVMS();
                JLMService.startJLMS();
                instance.start();
        }//if

}//startTest
```

Questi due metodi vengono invocati su tutti i nodi indipendentemente dal grado del vigile del fuoco. La differenza sostanziale invece sta nel task di esecuzione. Il capitano esegue la creazione e quindi termina immediatamente il proprio task, ma continua a funzionare tutto il middleware sottostante per la gestione del gruppo e per lo scambio di messaggi. Per creare il gruppo, inizializza un nuovo profilo, un nuovo vincolo ed invoca il metodo `CreateNewGroup` del `JLMService`.

Listing 5.3: Test/Test.java

```
/**
* The task of the component.
*/
protected void task() throws Throwable {

        ProfileFactory pf = new ProfileFactory();
        ProfileImpl groupProfile = pf.createTestGroupProfile();//Crea il profilo
        Vector v = ConstraintFactory.createConstraint();//Crea i vincoli
        String groupgid = JLMService.CreateNewGroup(groupProfile, v);//Crea il gruppo
        System.out.println("\nTEST: You have created a new group with GID: "+groupgid);

}//task
```

Il task degli LME che si uniscono al gruppo e di tutti gli ME presenta invece un ciclo `while` che viene ripetuto ad intervalli regolari (ogni 90 secondi) fino alla scoperta del gruppo ed alla avvenuta unione. Una volta unito ad un gruppo infatti il polling termina in quanto viene cambiato il valore della variabile che controlla il ciclo `while`.

Questa procedura di Join computa in questo modo: viene invocato il metodo `discoverAllGroups` del `JLMService` che restituisce tutti i gruppi rilevati in località, dopodichè, se i gruppi trovati sono almeno uno, viene invocato il metodo `Join` per il primo gruppo della lista restituita da `discoverAllGroups`.

Listing 5.4: Test/Test.java

```
/**
 * The task of the component.
 */
protected void task() throws Throwable {

        while(k==0){

                JLMService.discoverAllGroups();//Cerca i gruppi

            try{Thread.sleep(5000);}catch(Exception ex){}
```

```
71              Vector vec = JLMService.getDiscoveredGroups();//Preleva i gruppi trovati
73              if(vec.size()>0){
75                  JLMSDiscoverResponse gid = (JLMSDiscoverResponse)vec.get(0);
                        int i = gid.getGid();//GID del primo gruppo trovato
77                      JLMService.Join(i);//Join a tale gruppo
                        System.out.println("\nTEST: You have joined the group: "+i);
79                      k++;//Incrementa per uscire dal while
81              }//if
                else System.out.println("\nTEST: No groups found.");
83
                worker.sleep(period);//Pausa tra un tentativo ed un'altro
85
            }//while
87
}//task
```

Quando tutti i nodi sono usciti dal ciclo while, cioè si sono tutti uniti al gruppo, il task della classe `Test` cessa l'esecuzione ma il middleware sottostante continua a fornire la visibilità degli utenti co-locati e permette lo scambio dei messaggi dei soccorritori. A questo punto il middleware continua a gestire il gruppo autonomamente per tutta la durata dell'emulazione, raccogliendo i risultati dell'emulazione in file di testo per permettere un'analisi approfondita a posteriori.

Un esempio di quello che potrebbe accadere in realtà nel caso di un'operazione di soccorso è il seguente.

I vigili del fuoco ricevono la richiesta di intervento e si dirigono sullo scenario dell'incendio. Mentre si avvicinano alla zona del disastro, il capitano promuove la formazione dinamica del gruppo di soccorso. Il dispositivo del capitano gli permette di specificare il profilo del gruppo che intende creare, che in questo caso sarà *"Vigili del fuoco"*, insieme con gli attributi di profilo dell'utente, ad esempio *"Tom, Capitano"*. Il suo laptop agisce come LME e sfrutta il PENS installato localmente per generare la coppia GID/PID, il VMS per inizializzare la vista context-dependent ed il PS per pubblicare la disponibilità on-line del nuovo gruppo. Dato che per ora il dispositivo del capitano è l'unico LME presente, il VCS di istanza sul suo dispositivo concede il permesso al VMS di trasmettere.

Quando il capitano e tutta la squadra arriva sul luogo dell'incendio, il gruppo conterrà solamente l'entry del capitano, la stessa cosa che accade nella situazione emulata. Quando i vigili del fuoco, una volta giunti sullo scenario di intervento, accendono i propri PDA, iniziano la ricerca del gruppo e, a seconda se si trovano o meno nella località del capitano, si uniscono al gruppo trovato.

Tutti i vigili del fuoco che si vogliono unire al gruppo specificano, nel loro profilo, le caratteristiche richieste, ad esempio *"Vigili del fuoco"*, e tentano di accedere al gruppo promosso dal capitano. Il J/LMS installato sui PDA scopre l'LME coordinandosi col PENS. A questo punto il vigile del fuoco chiede all'LME di potersi unire al gruppo ed il capitano concede il permesso all'entrata.

L'unica limitazione di questa emulazione riguarda la mobilità degli ufficiali di alto grado. Questi infatti, in uno scenario reale, opererebbero per la maggioranza del tempo nella stessa località ed avrebbero piccoli spostamenti sullo scenario di emergenza. Nella situazione

emulata invece tutti i nodi godono della stessa mobilità, in quanto non è possibile, mediante il tool utilizzato per emulare il movimento, intervenire sulla mobilità di ogni singolo nodo.

Il testbed dell'emulazione è costituito da una serie di macchine che rappresentano i nodi della rete ad hoc, e da un tool software che emula il movimento dei nodi. Le macchine utilizzate possono essere sia reali sia emulate su altre macchine: nel primo caso, dato il bisogno di avere un elevato numero di nodi, è necessario disporre di un laboratorio, mentre nel secondo è possibile mediante un piccolo numero di macchine realizzare una una rete ad hoc di medie dimensioni, moltiplicando il numero di nodi emulati su ciascuna macchina. Data la disponibilità di un laboratorio[1] con un sufficiente numero di macchine, si è scelto di intraprendere questa strada emulando il movimento dei nodi mediante uno dei numerosi tool presenti in rete adatti a questo scopo.

In letteratura sono presenti numerose soluzioni ed applicazioni che permettono di eseguire il testing di reti ad hoc, ma quello scelto ed utilizzato per testare il caso in esame è l'emulatore *MobiEmu*. Mediante MobiEmu (vedi Appendice C) è possibile impiegare come oggetto del test l'intero middleware AGAPE e non un suo modello.

5.2 Parametri osservati

Al termine dell'implementazione del VMS all'interno del framework AGAPE, è stato possibile testare il funzionamento non solo del componente di interesse ma anche dell'intero middleware. Infatti ciò che viene eseguito sulle macchine del testbed di emulazione non è più solamente il VMS, come per la fase di simulazione, ma l'intero framework AGAPE.

L'applicazione descritta nella sezione precedente attiva tutti i servizi del middleware e raccoglie i dati dell'emulazione in file di testo, nonchè fornisce il supporto alla comunicazione degli utilizzatori.

I parametri osservati in questa fase di testing riprendono quelli già analizzati in sede di simulazione (vedi Appendice A) e sono i seguenti:

- **Hit ratio**: E' bene osservare se, anche ora, la scelta di una gestione FIFO produce ottimi risultati in termini di hit ratio.
- **Dimensione buffer**: Si misura in byte l'occupazione del buffer `cronologyBuffer`.
- **Intervallo di attivazione**: Viene analizzata l'adattatività del sistema ai cambiamenti di topologia. Ora, a differenza della simulazione, subentra un'ulteriore componente ad influenzare questo comportamento ed è il VCS. Infatti ora non è detto che tutti gli LME presenti in località lavorino, e nel caso non lo facciano, subiscono forti contrazioni di tale intervallo al fine di determinare in modo reattivo quando vi sono le condizioni per ripristinare l'esecuzione.
- **Copertura**: Questo parametro misura il numero di nodi, in un certo istante, che possiede una vista coerente. Anche in questo caso la presenza del VCS influenza pesantemente la misura, in quanto spesso si rinuncia, sia da parte del VCS sia del VMS, ad una copertura completa dei nodi in favore di un risparmio di banda.

[1] Il LAB2 della facoltà di Ingegneria di Bologna.

- **Impiego di banda**: Viene rilevato anche ora l'ammontare del numero di byte di ogni trasmissione da parte del VMS.
- **Latenza ai cambiamenti**: Può essere utile conoscere anche la reattività del sistema ai cambiamenti, ad esempio dopo quanto tempo la presenza di un nodo in località viene rilevata, oppure dopo quanto tempo un nodo che ha eseguito join ad un gruppo viene effettivamente inserito. Questa misura dipende fortemente però dalla frequenza di esecuzione non solo del VMS, ma anche del PENS e del PS. Sono questi ultimi infatti che rilevano e comunicano a tutti i nodi in località i cambiamenti di topologia e la presenza di nuovi nodi e gruppi.

5.2.1 Risultati del test

Dopo aver configurato il testbed per l'emulazione (vedi Appendice C.6), si è proceduto all'esecuzione del test.

E' stato eseguito un test in cui la durata del movimento dei nodi mediante MobiEmu era pari a 5 ore (18000 secondi nel comando per la generazione dello scenario con `setdest`). Data la grande mole di dati generati da ogni nodo durante l'emulazione, si è scelto di analizzare i risultati considerando degli intervalli di tempo significativi. In particolare si sono osservati i risultati dei vari parametri, singolarmente per i tre LME, su di un intervallo di osservazione pari ai primi 1000 istanti di attivazione. Si è voluto poi mettere in evidenza anche ulteriori aspetti, quali il corretto switching degli LME ad opera del VCS ed anche il valore della latenza della rete su operazioni di Join, per valutare il tempo impiegato da un nuovo membro a diventare visibile da ogni altro, cioè il tempo impiegato ad essere inserito nella vista.

Il controllo del corretto funzionamento della procedura di switching tra gli LME è importante per capire come avviene il passaggio del controllo tra le entità e quale degli LME presenti stia trasmettendo in un determinato istante. Inoltre, dato che ogni LME possiede una propria vista della località, è bene conoscere anche quale sia la copertura globale del gruppo a seguito di questi cambi di contesto tra i nodi gestori, eseguendo un merge delle coperture dei singoli LME.

5.2.1.1 Primo LME

Questo primo LME rappresenta il capitano che crea il gruppo all'inizio dell'emulazione ed è da questa entità che ha inizio la crescita e l'evoluzione del gruppo. Finchè gli altri due LME (ad esempio due sottotenenti) non si uniscono, tutti gli ME eseguono join da questo, dopodichè iniziano a gestire il gruppo anche gli altri LME, permettendo poi a tutti i nodi di unirsi.

Anche se il movimento casuale è identico per tutti i nodi, nell'intervallo scelto si osserva un movimento di questo primo LME molto limitato, che casualmente coincide con quello che potrebbe accadere nella realtà, cioè piccoli spostamenti per il gli alti ufficiali e grandi spostamenti per i vigili del fuoco semplici.

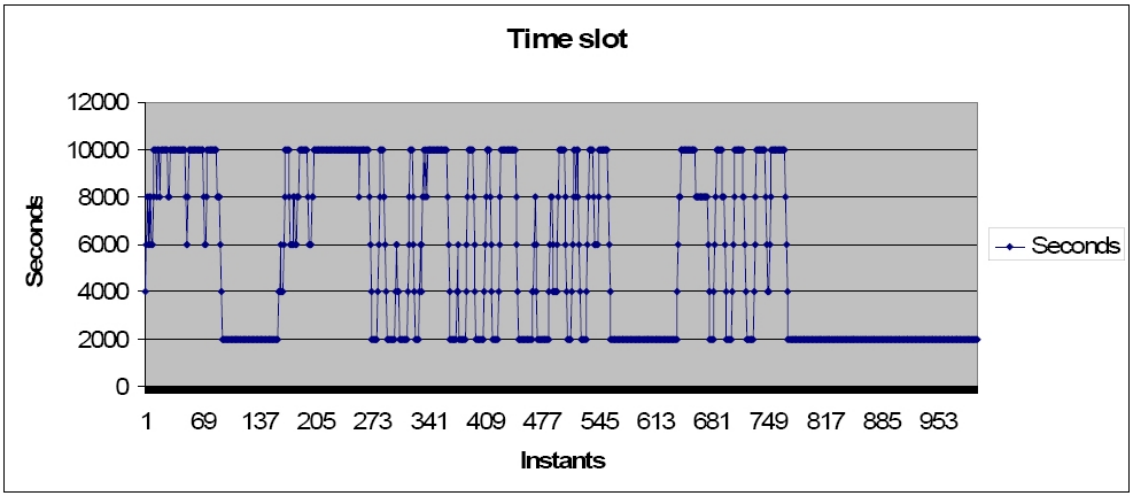

Figura 5.1: Emulazione: Istanti di attivazione LME1.

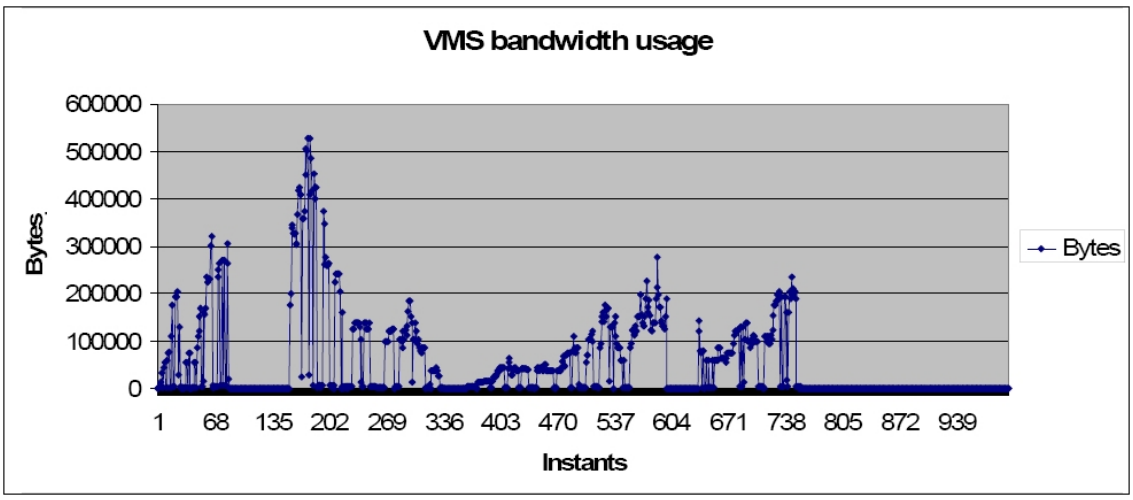

Figura 5.2: Emulazione: Utilizzo di banda LME1.

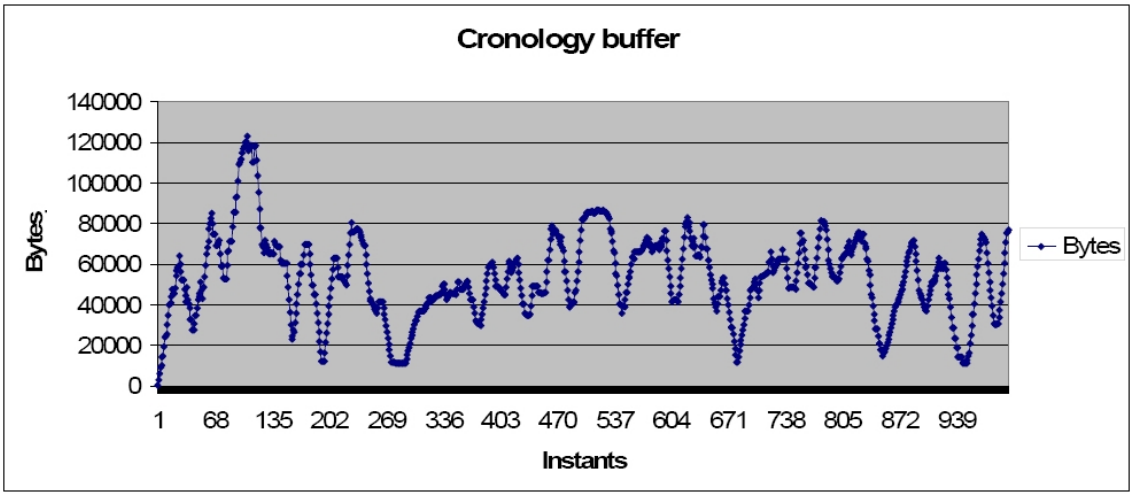

Figura 5.3: Emulazione: Dimensione buffer cronologia LME1.

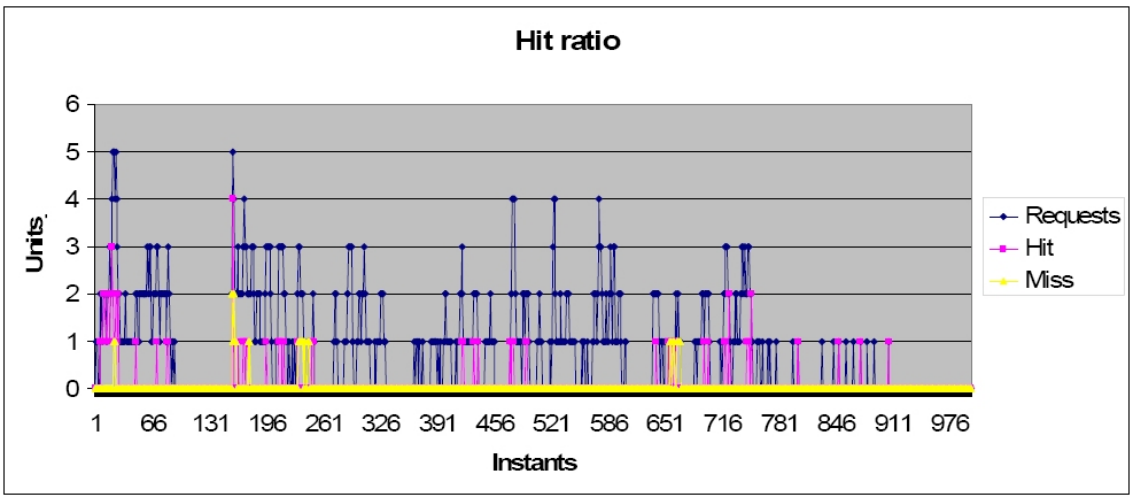

Figura 5.4: Emulazione: Hit ratio dei riferimenti in cache LME1.

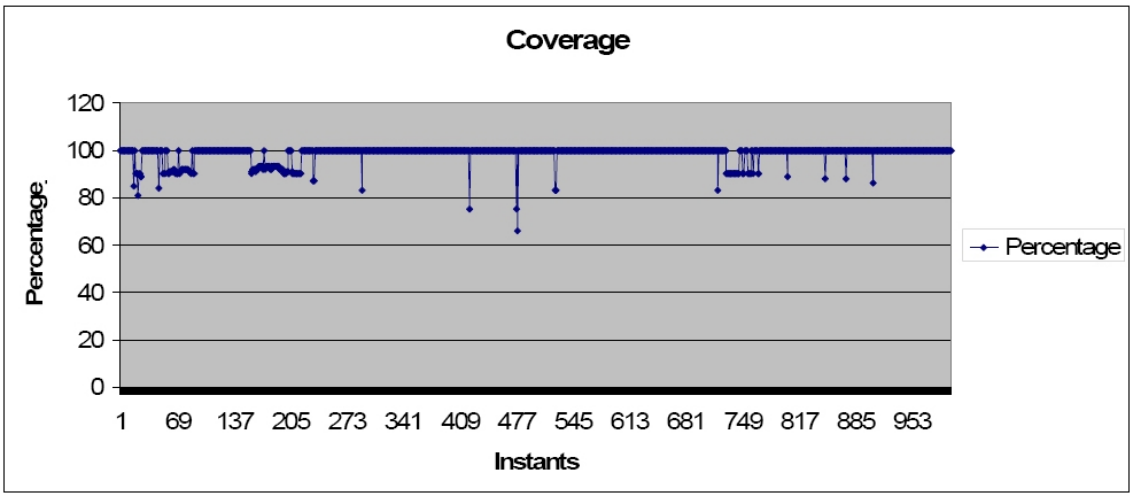

Figura 5.5: Emulazione: Copertura LME1.

Il grafico di Figura 5.1 mostra l'andamento del task del VMS in termini di istanti di attivazione. La rete alterna andamenti ad alta dinamicità ad altri meno dinamici ottenendo un valore medio per gli intervalli di attivazione pari a $5.23s$. E' da notare inoltre, che quando il VMS accelera il suo andamento, non sempre dipende solo dalla rete, ma è causato anche dalla sua temporanea disattivazione dovuta alla presenza di un'altro LME nelle sue vicinanze con vista migliore della sua. In questi casi infatti, al fine di riprendere al più presto l'esecuzione reagendo in modo reattivo al cambio di contesto, il VMS aumenta la propria frequenza di attivazione del task. In questo caso infatti, la lunga persistenza dell'intervallo sui 2 secondi è indice del fatto che l'LME non sta trasmettendo e tenti di ripristinare la sua esecuzione campionando più frequentemente la rete.

L'andamento del grafico di Figura 5.1 si rispecchia fedelmente in quello dell'utilizzo di banda, il quale evidenzia un valore nullo negli intervalli in cui non trasmette e dei valori proporzionali alla frequenza di esecuzione. Si nota infatti, che quando il VMS esegue più frequentemente, il valore della banda utilizzata è minore del caso ad intervalli di attivazione più ampi.

La Figura 5.3 mostra l'andamento dell'occupazione del buffer della cronologia, il quale non occupa mai, nell'intervallo di osservazione, valori troppo elevati. La Figura 5.4 mostra invece la hit ratio del buffer della cronologia: le richieste, a differenza della simulazione, sono ora molto inferiori e il valore della hit ratio si abbassa leggermente. Si verificano infatti più miss di quelle che ci si aspettava mediante la simulazione, ma questo è dovuto al fatto che i nodi ora possono uscire e rientrare anche dopo numerosi istanti di attivazione, che, se sono superiori al valore della cache (in questo caso pari a 15 entry), richiedono aggiornamenti non più presenti.

L'ultimo grafico infine mostra l'andamento della copertura dal punto di vista dell'LME in questione. La percentuale è sempre molto elevata con un valore medio del 98.62% e gli abbassamenti presenti sono dovuti alle scelte euristiche del VMS e del VCS. Ora, dato anche il minor numero di richieste rispetto allo scenario di simulazione, la copertura è

nettamente migliorata.

La seguente tabella riassume i parametri rilevati del primo LME.

Intervallo medio	$5.23s$
Tempo simulato	$1h, 27m$
Richieste	766
Hit	91
Miss	29
Hit ratio	75%
Copertura media	98.62%
Occupazione media buffer	$53.4Kb$
Aggiornamento medio	$54.9Kb$

Tabella 5.1: Primo LME.

5.2.1.2 Secondo LME

Questo LME svolge solamente la funzione di joining e rappresenta uno dei due sottotenenti. Una volta che il vigile del fuoco si è aggregato al gruppo creato dal capitano, questi inizia a gestirlo inviando le proprie viste context-dependent. Come si può notare dal grafico seguente però, l'attività del membro è sporadica all'interno dell'intervallo osservato e dipende dalla sua locazione all'interno dell'area di emulazione. Il cambiamento di contesto tra gli LME è meglio visibile in seguito nel relativo grafico.

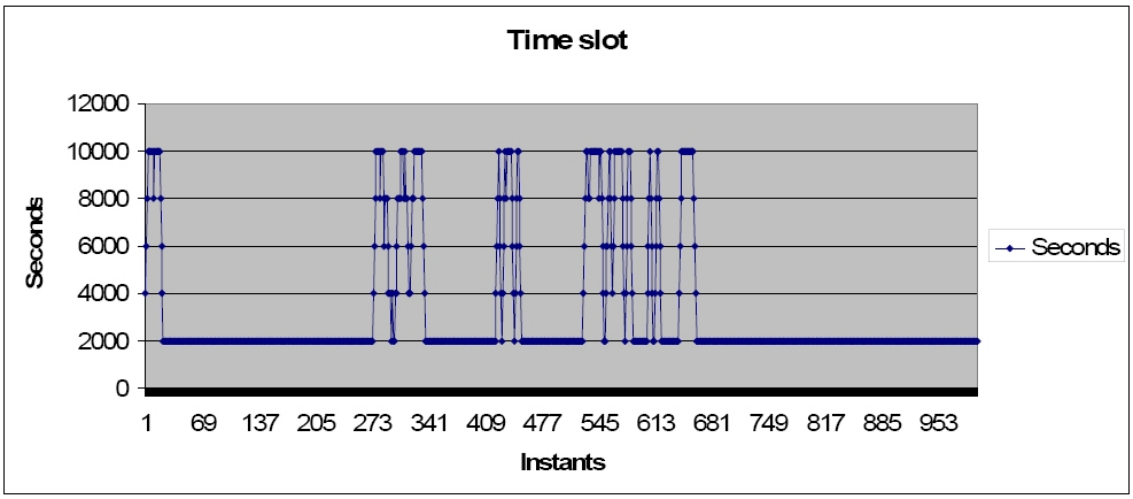

Figura 5.6: Emulazione: Istanti di attivazione LME2.

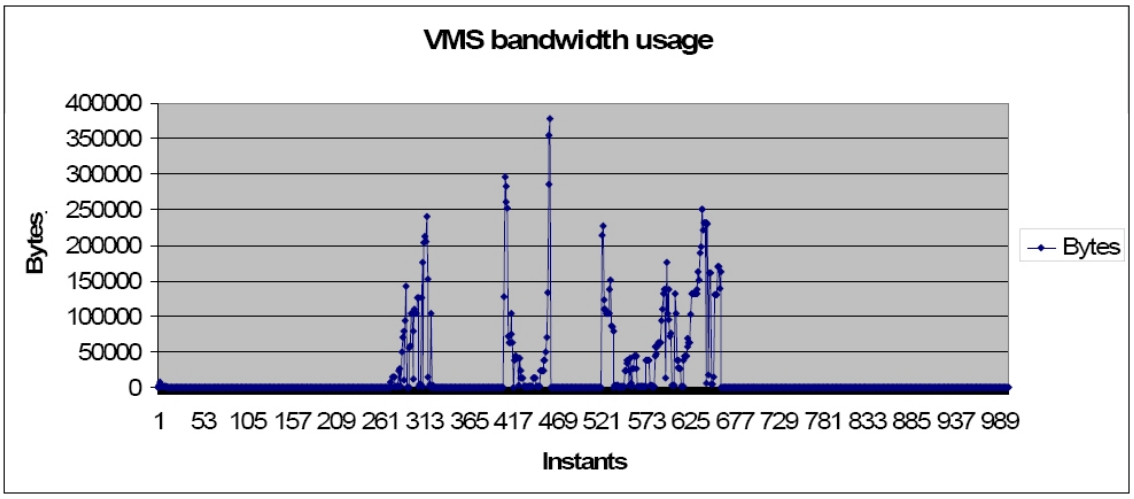

Figura 5.7: Emulazione: Utilizzo di banda LME2.

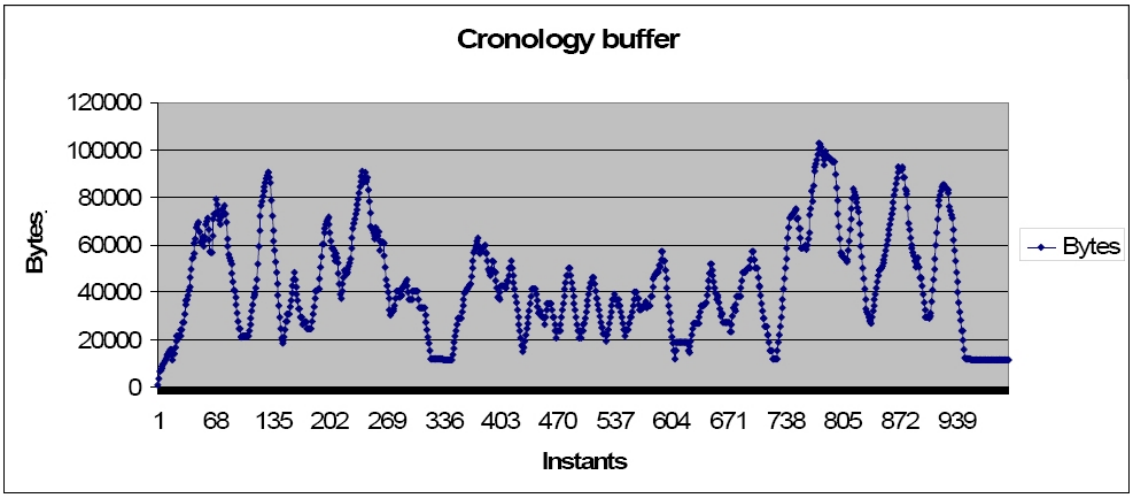

Figura 5.8: Emulazione: Dimensione buffer cronologia LME2.

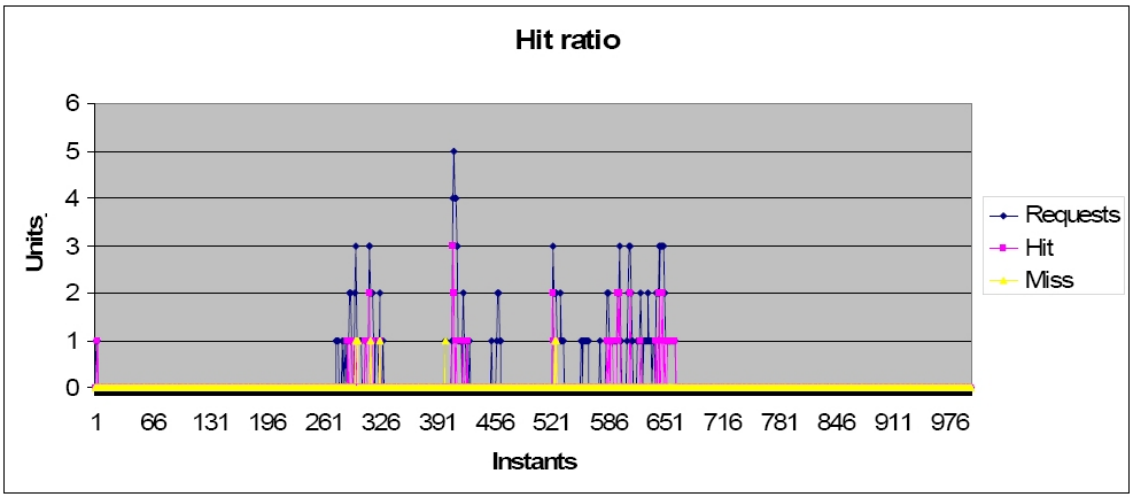

Figura 5.9: Emulazione: Hit ratio dei riferimenti in cache LME2.

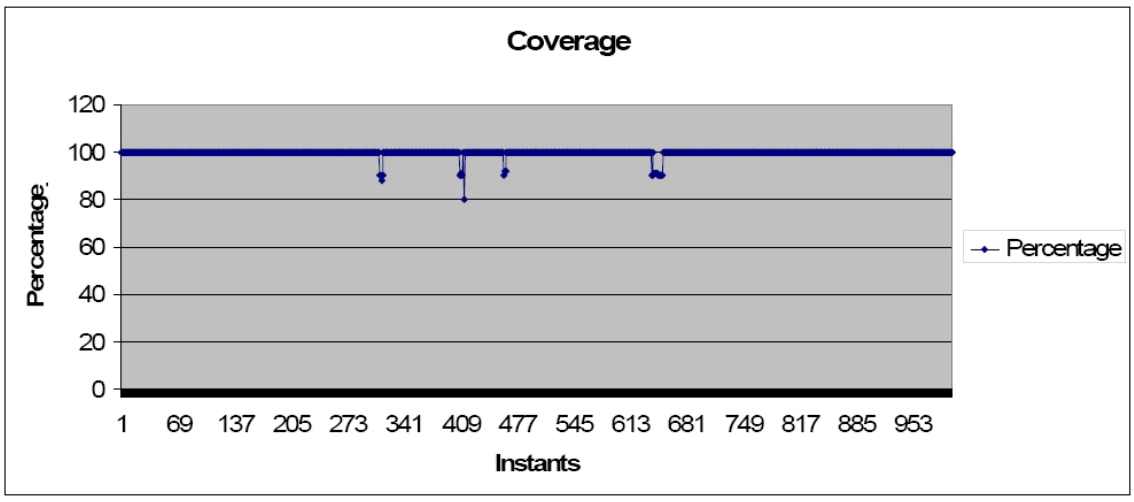

Figura 5.10: Emulazione: Copertura LME2.

Gli intervalli in cui l'LME trasmette sono evidenti anche dalla Figura 5.7 dell'utilizzo di banda. In essa infatti sono evidenti gli intervalli di trasmissione che interrompono le zone di non invio. Il buffer della cronologia ha un andamento abbastanza simile a quello del primo LME, mentre il grafico della hit ratio presenta delle richieste solamente nelle zone in cui il membro propaga gli aggiornamenti. Quando non lavora infatti, nessun membro ha necessità di richiedere un'aggiornamento perso. La percentuale della hit ratio è ora aumentata all'87%.

Il grafico della copertura infine, a causa del basso lavoro svolto dall'entità rimane sempre sul valore massimo, con un valore medio del $99,74\%$. La seguente tabella mostra i risultati del secondo LME analizzato.

Intervallo medio	$3.2s$
Tempo simulato	$53m$
Richieste	220
Hit	57
Miss	8
Hit ratio	87%
Copertura media	99.74%
Occupazione media buffer	$43.3Kb$
Aggiornamento medio	$16.4Kb$

Tabella 5.2: Secondo LME.

5.2.1.3 Terzo LME

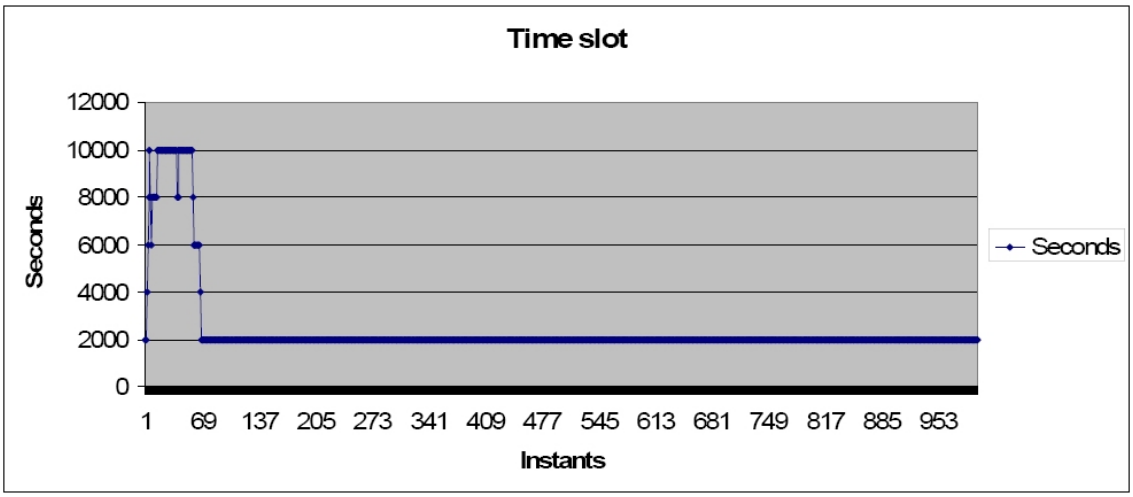

Figura 5.11: Emulazione: Istanti di attivazione LME3.

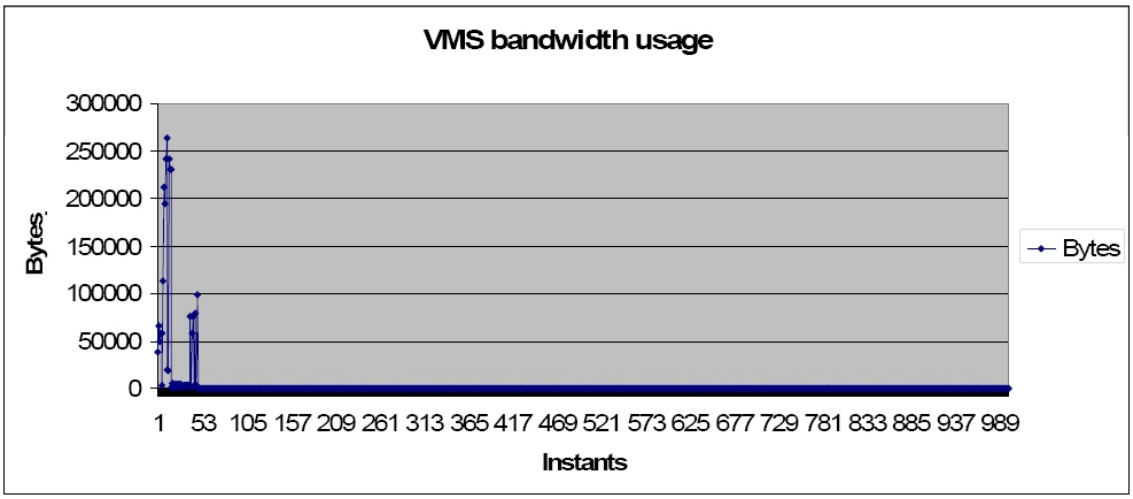

Figura 5.12: Emulazione: Utilizzo di banda LME3.

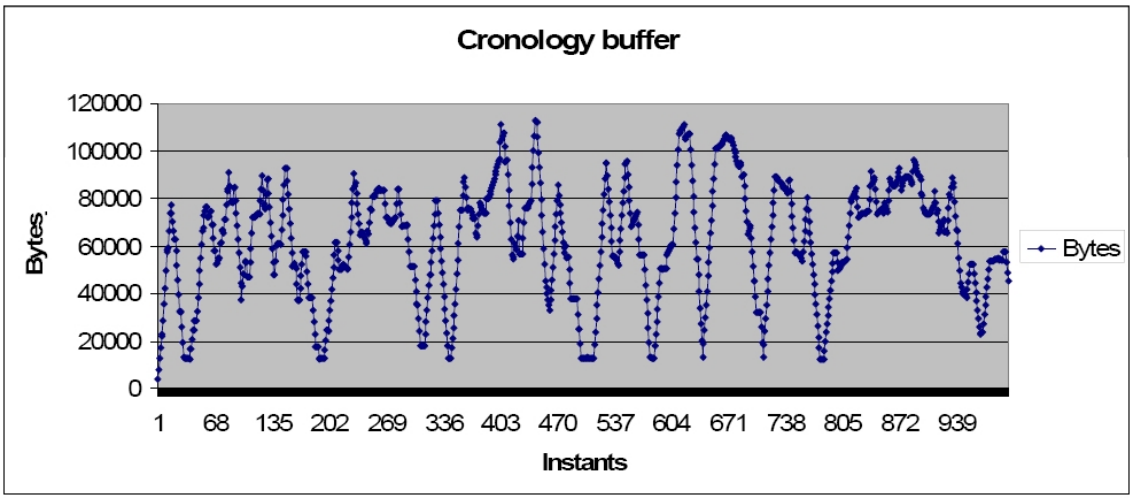

Figura 5.13: Emulazione: Dimensione buffer cronologia LME3.

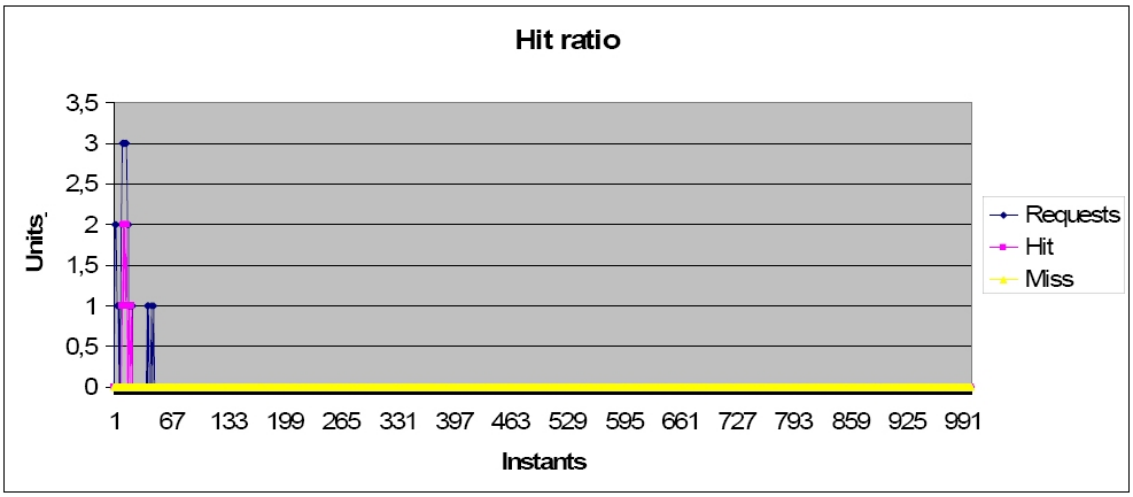

Figura 5.14: Emulazione: Hit ratio dei riferimenti in cache LME3.

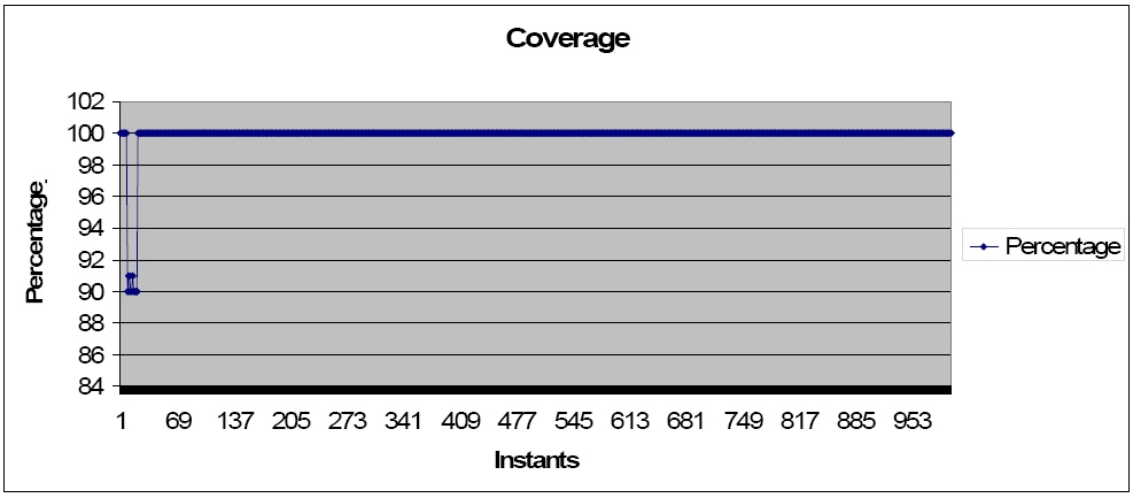

Figura 5.15: Emulazione: Copertura LME3.

Quest'ultimo LME, a causa della trasmissione degli altri due LME ed alla permanente presenza, all'interno dell'intervallo di osservazione, nella località del primo LME, a parte la fase iniziale non attiva mai la propria trasmissione. Questo si riflette su tutti i grafici riportati in quanto rimane in ascolto degli altri LME trasmittenti.

L'intervallo di attivazione rimane dunque fisso sui due secondi, al fine di riprendere il controllo, mentre l'utilizzo di banda è nullo per tutto l'intervallo di tempo osservato. Le richieste chiaramente non avvengono e le poche giunte negli istanti iniziali sono state soddisfatte, ottenendo una hit ratio del 100%.

La copertura della località dal suo punto di vista rimane fissa al 100% anche se in realtà non è proprio tale. Infatti la percentuale di copertura, quando non trasmette, dipende dall'LME trasmittente e dal suo lato di osservazione è sempre massima. La copertura della rete sarà meglio evidenziata nel grafico relativo.

La seguente tabella riporta i risultati del test per il terzo LME.

Intervallo medio	$2.44s$
Tempo simulato	$40m$
Richieste	40
Hit	12
Miss	0
Hit ratio	100%
Copertura media	99.87%
Occupazione media buffer	$61.9Kb$
Aggiornamento medio	$2.5Kb$

Tabella 5.3: Terzo LME.

La seguente tabella invece riassume i risultati dei tre LME analizzati.

Parametro	**1° LME**	**2° LME**	**3° LME**
Istanti di attivazione	1000	1000	1000
Intervallo medio	$5.23s$	$3.2s$	$2.44s$
Tempo simulato	$1h, 27m$	$53m$	$40m$
Richieste	766	220	40
Hit	91	57	12
Miss	29	8	0
Hit ratio	75%	87%	100%
Copertura media	98.62%	99.74%	99.87%
Occupazione media buffer	$53.4Kb$	$43.3Kb$	$61.9Kb$
Aggiornamento medio	$54.9Kb$	$16.4Kb$	$2.5Kb$

Tabella 5.4: Tabella riassuntiva dei risultati degli LME.

Per quanto riguarda il numero di nodi che hanno preso parte al gruppo, cioè si sono uniti mediante operazioni di join, è pari a 30, cioè tutti i nodi presenti in località. Questi nodi si sono uniti non appena hanno trovato il gruppo mediante l'operazione di discovery, ed hanno continuato a collaborare, come ci si attendeva, per tutte e cinque le ore di emulazione.

5.2.1.4 Copertura globale

Come già accennato, ogni LME possiede una vista esclusiva della sua località e pertanto le percentuali di copertura rilevate si devono riferire secondo il punto di vista dell'LME in questione. Per avere una migliore conoscenza riguardo la copertura della rete globale è necessario eseguire una sorta di merge dei dati riguardanti le singole percentuali di copertura rilevate dagli LME.

Quando un LME non trasmette infatti, assesta il proprio livello di copertura pari al 100%, ma dal punto di vista dell'LME che sta trasmettendo in quello stesso istante, tale valore può non corrispondere al massimo ma essere eventualmente solamente inferiore. Al fine di determinare la copertura globale quindi, si sono considerati i valori rilevati da ciascun LME in un'intervallo pari a 3000 istanti di attivazione, e si è determinato per ciascun istante, il minimo riscontrato dalle tre entità.

L'andamento della copertura globale è mostrato dal grafico di Figura 5.16.

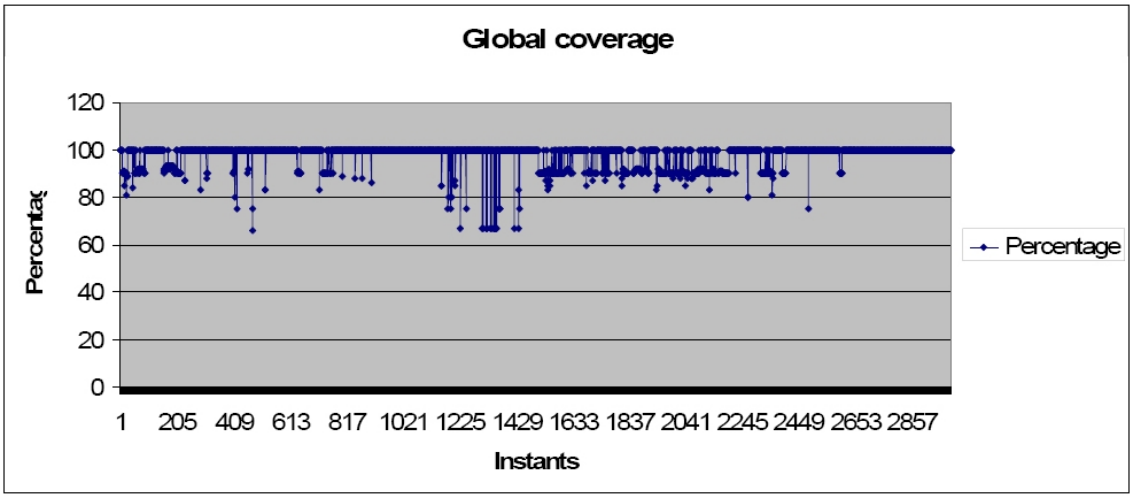

Figura 5.16: Copertura globale per il gruppo.

Questo andamento è stato ottenuto pertanto dalle coperture dei tre LME i quali possiedono ciascuno, a seconda degli intervalli di attivazione, una zona di responsabilità in tale grafico.

La percentuale di copertura fornita dai tre LME è sempre molto alta per tutto l'intervallo considerato e si assesta ad un valore medio del 97.63%. Questo sta a dimostrare il buon lavoro svolto dal VMS e dal VCS di istanza in ognuno degli LME.

5.2.1.5 Latenza alle operazioni di join

Mediante questa analisi si vuole mostrare la reattività del sistema all'arrivo di nuovi nodi nel gruppo. Risulta infatti interessante conoscere dopo quanto tempo, un nodo che ha eseguito join al gruppo, risulta visibile all'interno della vista di contesto dell'LME che ha permesso la sua entrata, e quindi il tempo impiegato per essere visibile a tutti i membri della località.

A questo fine sono stati memorizzati nei file di log, dei timestamp relativi all'istante in cui ciascun membro accedeva al gruppo e, osservando le entry della vista dell'LME gestore, è possibile capire quanti secondi sono stati impiegati per l'inserimento del membro nella vista.

E' bene notare però, che questa misura dipende in maniera predominante dalle velocità di esecuzione dei componenti attivi di AGAPE, quali il PENS il PS ed il VMS, nonchè del traffico di rete presente in quel determinato momento, dato che l'operazione di join si basa sullo scambio di messaggi.

Il grafico di Figura 5.17 mostra il valore riscontrato per 20 dei 27 ME dell'emulazione. Le operazioni di join di queste 20 entità, avvengono in un range temporale di circa 25 minuti dall'inizio dell'emulazione e dunque per ognuna di esse si hanno condizioni di traffico sempre diverse. In particolare si avrà traffico maggiore all'aumentare dell'istante in cui avviene tale unione al gruppo.

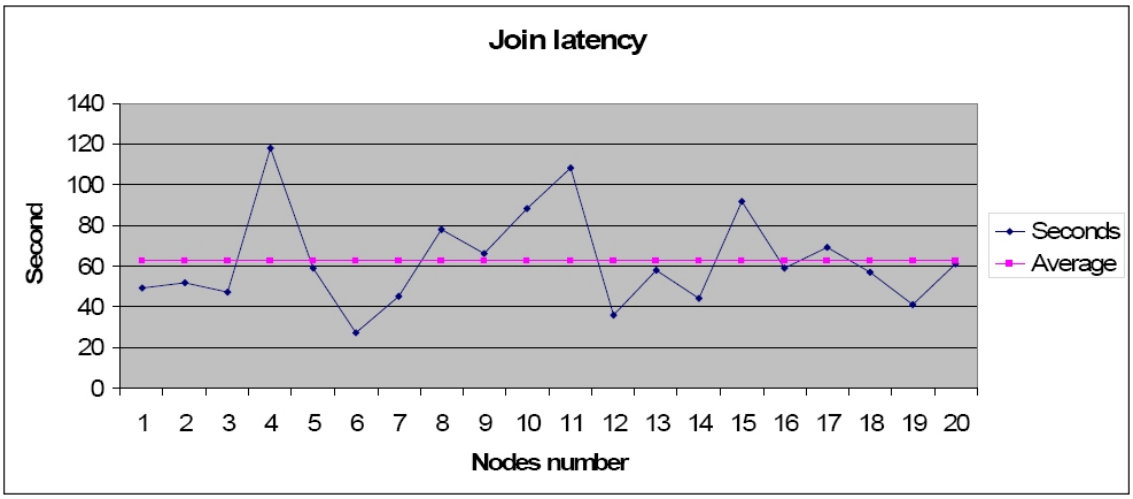

Figura 5.17: Latenza alle operazioni di join.

L'andamento risulta pertanto non omogeneo tra le entità e presenta un valore medio di $62.7s$. Questo ritardo nell'inserimento in vista è dovuto al fatto che il membro che vuole entrare nel gruppo invia il proprio profilo non immediatamente al momento della richiesta, ma successivamente quando riscontra di non essere presente nella vista di gruppo. Quando il profilo giunge all'LME, esso inserisce il membro all'interno della vista e può quindi essere raggiungibile da tutti gli altri membri del gruppo co-locati.

5.2.1.6 LME switching

Al fine di verificare il corretto funzionamento del VCS, si intende mostrare in che modo, all'interno di un'intervallo temporale, gli LME si passano l'autorizzazione a gestire il gruppo. Si considera ora un'intervallo di osservazione leggermente minore dei precedenti, in particolare una finestra temporale di $1024s$ dalla partenza dell'emulazione.

In questo intervallo di tempo gli LME si muovono secondo le proprie direzioni e per diversi momenti si trovano ad essere co-locati. Il cambio del controllo tra i nodi gestori viene ben descritto dal grafico di Figura 5.18, in cui viene rappresentato lo stato ON dal valore 1 sull'asse delle ordinate e lo stato OFF dal valore 0. Gli LME vengono numerati con lo stesso identificatore che compare nella GUI di MobiEmu in modo tale da essere meglio riconosciuti.

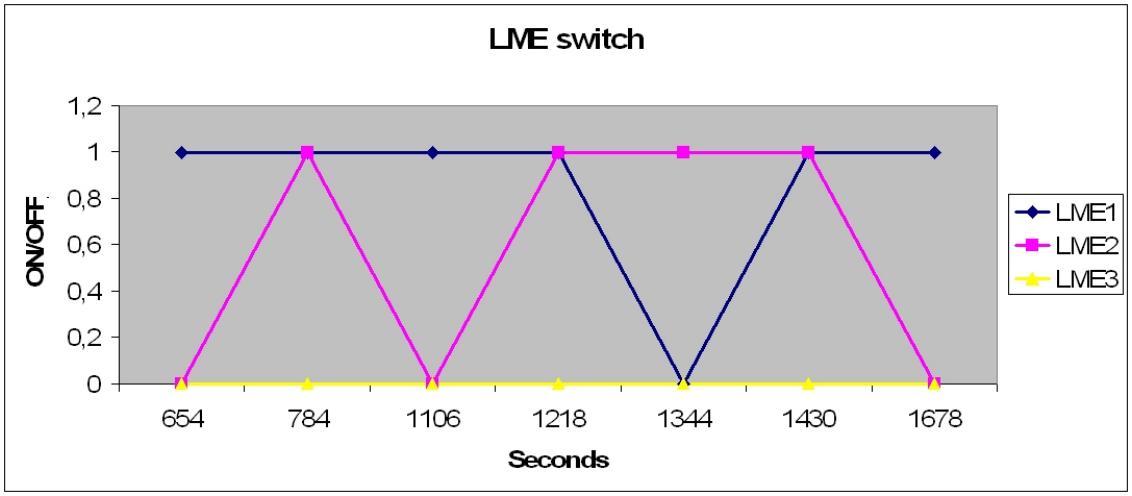

Figura 5.18: Scambio del controllo tra gli LME.

Istante 654. Dopo 654 secondi dall'inizio dell'emulazione gli unici LME presenti nel gruppo sono il nodo stesso che lo ha creato ed LME2, ma dato che la vista di LME2 è tutta contenuta in quella di LME1, nel grafico di Figura 5.18 viene posto sullo stato ON (valore 1 nel grafico) solamente LME1. Questa situazione iniziale dell'intervallo osservato viene rappresentata anche da uno screen-shot estratto dall'interfaccia grafica di MobiEmu durante l'esecuzione del movimento, e riportato in Figura 5.19. In particolare vengono indicati con un pallino giallo gli LME trasmittenti e con un pallino bianco gli LME il cui VCS non ha concesso l'autorizzazione a trasmettere. Tutte le altre entità sono nodi ME.

In questo caso, la vista di gruppo di LME1 contiene tutti i nodi presenti in figura, mentre quella di LME2 li contiene tutti tranne l'unico ME irraggiungibile con identificatore 20. Il VCS di LME2 pertanto riconosce che la vista che si vuole inviare è tutta contenuta in quella di LME1 e non autorizza dunque il VMS alla trasmissione.

A questo momento dell'emulazione LME3 è ancora al di fuori della località definita da LME1, che in questo caso è $h = 1hop$, e pertanto non può effettuare l'operazione di join.

Più tardi, precisamente dopo circa 700 secondi, anche LME3 entra in località di LME1 e si unisce al gruppo esistente.

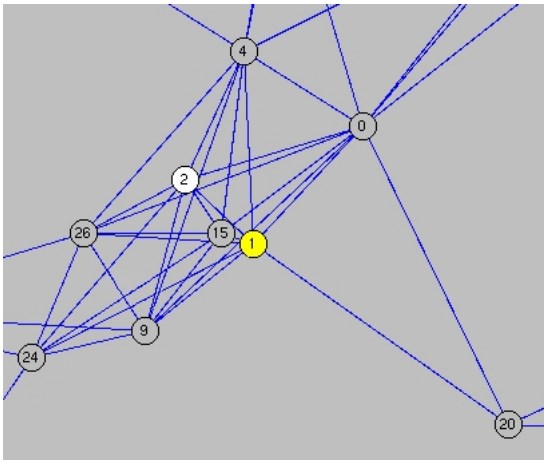

Figura 5.19: Istante 654: Trasmette solo LME1.

Istante 711. Questo istante non viene indicato nel grafico del cambio di controllo in quanto rappresenta il momento di entrata nel gruppo del terzo LME, cioè di LME3. In questo istante infatti tale nodo ha effettuato la richiesta di join al gruppo ed è stato accettato. Come si può notare dalla figura seguente però, a tale nodo non è permesso trasmettere in quanto lo sta già facendo LME1 e non è necessaria la sua trasmissione ai fini della copertura del gruppo.

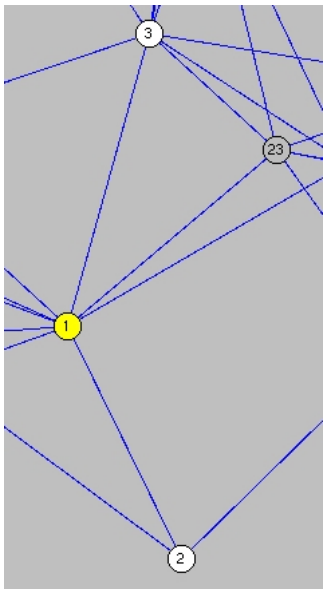

Figura 5.20: Istante 711: Entrata nel gruppo di LME3.

Istante 784. In questo preciso istante, LME2 si allontana da LME1 ed ottiene pertanto il permesso alla trasmissione dal proprio VCS. Nella figura seguente viene mostrata questa situazione: LME2 si è allontanato sufficientemente, tale da non rilevare le trasmissioni di LME1 e, dato che LME3 non sta trasmettendo, il proprio VCS crede di essere l'unico LME in località e concede il permesso alla trasmissione al VMS. In questo modo LME1 continua il proprio lavoro ed LME2 riprende a trasmettere. Il terzo LME invece, come si nota dalla figura, risulta essere nell'intersezione delle due località definite dagli altri LME e riceve pertanto le viste di gruppo da parte di entrambi. Osservando tali viste, riconosce che esse coprono totalmente i membri del gruppo della sua località, e pertanto il VCS blocca la sua trasmissione che risulterebbe totalmente ridondante.

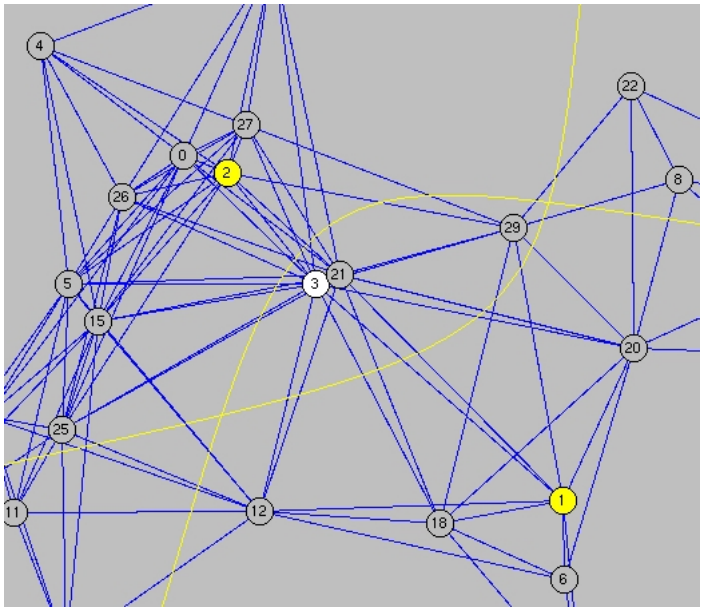

Figura 5.21: Istante 784: Trasmettono LME1 ed LME2.

Nel grafico relativo al cambio del controllo, all'istante 784 vengono pertanto posti a 1, cioè sullo stato ON, sia LME1 che LME2.

Istante 1106. Dopo aver lavorato ad una certa distanza, i due LME trasmittenti ritornano ad essere co-locati. In questo movimento continuo reciproco dei due LME, un ruolo particolare viene svolto da LME3 che, essendo costantemente co-locato con LME1, non riceve mai, all'interno di questo intervallo di osservazione, l'autorizzazione a trasmettere. Questo accade in primo luogo perchè la sua vista non risulta essere mai migliore di quella di LME1, ed in secondo luogo perchè il VCS previene continui passaggi di controllo che causerebbero un degrado delle prestazioni.

In Figura 5.22 viene mostrata questa situazione. Quando il nodo LME2 rientra in località di LME1 quest'ultimo sta trasmettendo, mentre LME3 è ancora bloccato dal proprio VCS. In questo istante LME1 possiede una vista di gruppo contenente 14 membri, cioè tutti quelli visibili in figura, mentre LME2 ed LME3 possiedono esattamente un sottoinsieme

di tale vista. I VCS dei rispettivi LME prevengono la propagazione da parte del VMS dei propri aggiornamenti periodici.

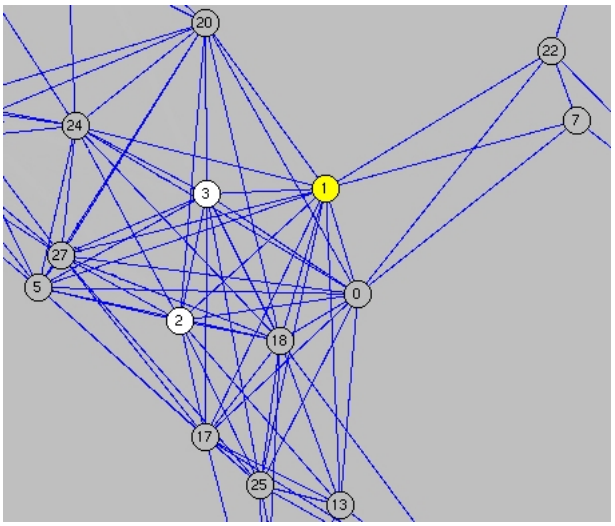

Figura 5.22: Istante 1106: Trasmette solo LME1.

Istante 1218. In questo istante LME2 riprende a lavorare in quanto si è di nuovo allontanato da LME1 portandosi in una zona a sud dell'area emulata, dove esso è l'unico LME gestore. LME1 invece è rimasto co-locato con LME3 e continua a possedere la vista migliore tra i due. Nel grafico del passaggio del controllo vengono pertanto posizionati su ON sia LME1 che LME2, mentre rimane fisso a OFF l'LME3.

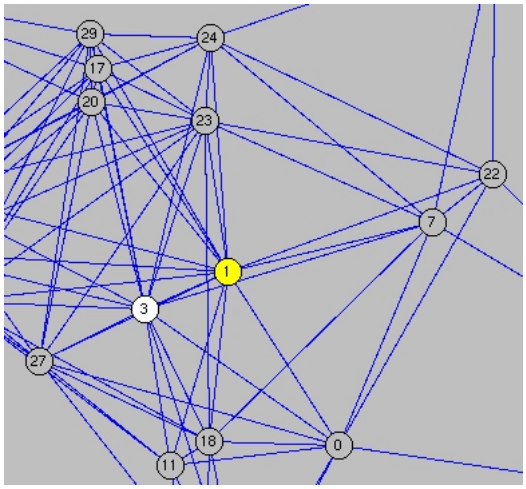

Figura 5.23: Istante 1218: Trasmettono LME1 ed LME2 (non visibile).

La vista di LME1 (16 membri) contiene infatti ancora una volta quella di LME3 (14 membri), che risulta pertanto impossibilitato alla propagazione.

Istante 1344. In questo istante LME2 diventa di nuovo co-locato con LME1 ma in questo caso, a differenza dei precedenti istanti, possiede una vista migliore di quest'ultimo e di LME3. Essendo ora in una posizione più centrale (vedi figura seguente) rispetto ai circostanti nodi e rispetto alla posizione di LME1, ora LME2 contiene la vista di LME1 e pertanto è concesso solo a lui il permesso alla trasmissione.

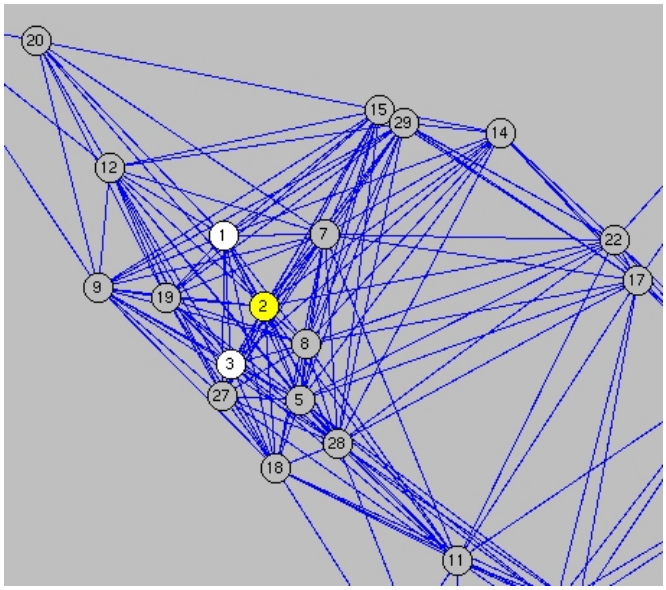

Figura 5.24: Istante 1344: Trasmette solo LME2.

La vista di LME2 contiene ora 18 membri del gruppo, cioè tutti quelli visibili tranne ME17, mentre LME1 ed LME3 ne contengono solamente un sottoinsieme.

Istante 1430. A causa del continuo movimento di LME2, nodo molto più dinamico di LME1 ed LME3, quest'ultimo si trova, dopo appena un minuto e mezzo dal precedente rilevamento, sufficientemente distante da LME1 tale da permettere la sua ripresa di trasmissione.

Come mostra chiaramente la figura seguente, i due LME trasmittenti non rilevano le viste di gruppo reciproche e pertanto, dato che LME3 non sta trasmettendo, credono di essere gli unici LME in località e pertanto propagano le viste senza la necessaria consultazione del VCS.

I nodi che si trovano nell'intersezione delle due località, ricevono pertanto le viste di gruppo da parte di entrambi gli LME e costruiscono la propria vista di gruppo mediante il merge di queste due. Tra i nodi dell'intersezione è presente però anche LME3 che, trovandosi in una posizione di mezzo tra le due località, possiede nella propria vista di gruppo gran parte dei membri di entrambe le località. Dato che la copertura svolta da LME1 ed LME2 è sufficiente a coprire anche la località di LME3, il VCS di quest'ultimo riconosce questo fatto dall'analisi delle viste ricevute e non autorizza il VMS a trasmettere.

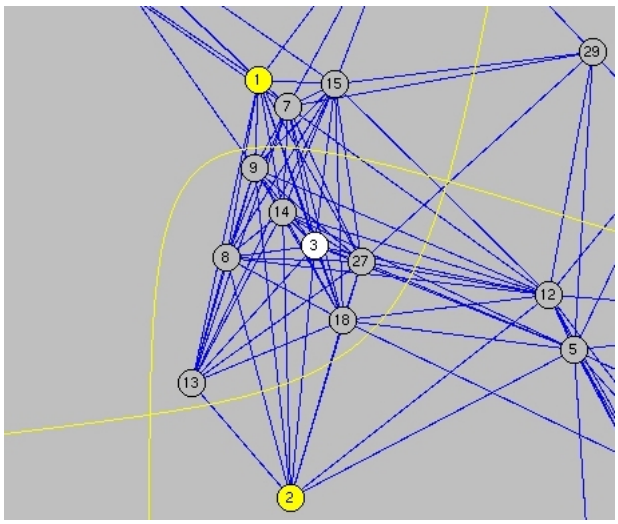

Figura 5.25: Istante 1430: Trasmettono LME1 ed LME2.

Istante 1678. Dopo lo spostamento di LME2 in una zona più a nord, LME1 torna di nuovo ad essere in una posizione centrale rispetto al resto dei nodi della rete e pertanto gode della vista migliore. Ora LME1 contiene nella propria vista tutti i nodi visibili in Figura 5.26 (in totale 16) tranne ME4 che risulta essere in una posizione abbastanza periferica. LME3 ed LME2 possiedono ancora una volta un sottoinsieme della vista di LME1.

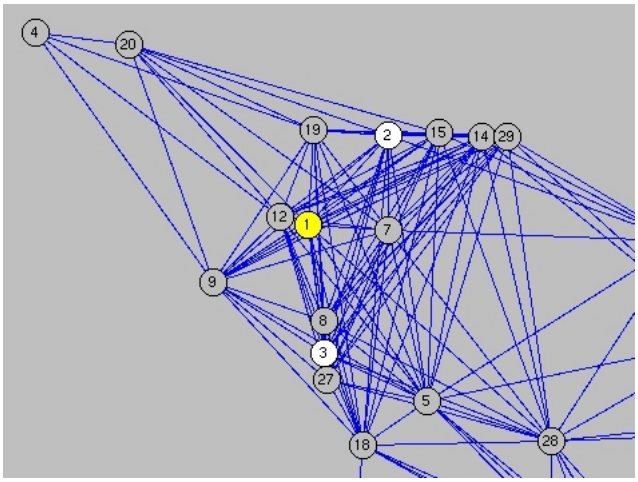

Figura 5.26: Istante 1678: Trasmette solamente LME1.

Conclusioni

Lo sviluppo di servizi collaborativi avanzati in scenari MANET, solleva problemi complessi che richiedono l'adozione di nuove linee guida per il design e l'implementazione del supporto alla collaborazione. In questa tesi abbiamo mostrato come la possibilità di beneficiare della completa visibilità del profilo dei vicini possa costituire il fondamento per lo sviluppo di applicazioni collaborative in scenari MANET.

La presente tesi introduce AGAPE, un sistema di gestione dei gruppi che promuove e supporta lo sviluppo di servizi collaborativi avanzati in ambienti MANET. In particolare, in questo lavoro ci siamo focalizzati sull'analisi, sullo sviluppo e sulla implementazione del *View Manager Service* (VMS) di AGAPE che consente la creazione, la gestione e la disseminazione delle viste ai membri del gruppo. In particolare, il VMS non solo fornisce ad ogni entità di AGAPE la completa visibilità dei membri del gruppo presenti nella stessa località di rete, ma propaga a livello applicativo la completa visibilità delle loro caratteristiche ed attributi. Una ulteriore peculiarità del sistema è quella di consentire una gestione adattativa delle viste in funzione del contesto e, in particolare, dell'attuale grado di dinamicità della rete. Al fine di testare il sistema, è stato inoltre implementato il prototipo di un'applicazione collaborativa per scenari di protezione civile. Il prototipo implementato ha confermato che il supporto alla gestione dei gruppi fornito da AGAPE è adatto al supporto di applicazioni collaborative in scenari Mobile Ad hoc NETworks.

Questi primi risultati ottenuti, possono stimolare ulteriori attività di ricerca al fine di migliorare l'attuale prototipo e di verificare l'applicabilità del sistema nello sviluppo di servizi collaborativi in diversi scenari applicativi.

Appendice A

Simulazione

Mediante la *simulazione* si tenta di ricreare, in ambito esclusivamente software, una situazione di rete reale all'interno della quale sviluppare e testare l'applicazione durante la fase di *design*. Quest'ultima pertanto non rappresenterà la versione finale, ma risulterà essere una versione implementata ad hoc al fine di poter essere eseguita all'interno dell'ambiente di simulazione. Per questo motivo la simulazione risulta molto limitata dal punto di vista dell'affidabilità della misura, in quanto l'oggetto di tale misura non è l'applicazione vera e propria, ma un modello di essa che ne riproduce il comportamento.

La simulazione offre la possibilità di eseguire esperimenti ripetibili e controllabili con un modesto overhead. Spesso però, possono essere effettuate delle ipotesi e delle semplificazioni che possono portare poi a risultati piuttosto differenti da quelli reali. Il livello di fedeltà è dunque abbastanza basso, ma già attraverso la simulazione si comprendono a grandi linee quali saranno i comportamenti macroscopici del sistema.

La simulazione è risultata molto utile in fase di progetto, per la verifica ed il testing degli algoritmi sviluppati, ma non altrettanto utile in fase di testing finale, a causa del livello di precisione eccessivamente basso. Per quest'ultima fase infatti, verrà utilizzata la tecnica dell'emulazione (vedi Capitolo 5), in quanto permette il testing dell'applicazione vera e propria svincolata dall'ambiente di simulazione, raggiungendo pertanto un livello di precisione della misura molto vicino alla realtà.

A.1 Tool grafico di simulazione

Al fine di valutare le prestazioni delle procedure e degli algoritmi già durante la fase di progetto, è stato sviluppato un tool grafico che permette la simulazione di una rete ad hoc all'interno della quale viene creato un gruppo gestito da un unico LME. Il tool permette l'impostazione di diversi parametri, al fine di simulare diverse condizioni di comportamento e quindi il maggior numero possibile di situazioni di rete. In particolare il tool di simulazione permette di settare i seguenti parametri:

- **Numero di ME**: Dato che esiste solo un gruppo ed un solo LME gestore è necessario decidere quanti nodi ME inserire nell'area di simulazione.

- **Livello di dinamicità**: Questo parametro varia da 0 a 5 ed indica il livello di movimento dei nodi della rete, cioè di quanto si spostano da un'istante all'altro. Maggiore è il livello di dinamicità, più grandi saranno gli spostamenti effettuati da tutti i nodi della rete tra due istanti consecutivi.Si va da un livello 0 in cui i nodi rimangono immobili fino ad un livello 5 in cui i nodi subiscono spostamenti significativi.

- **Istanti di tempo**: E' necessario per impostare la durata della simulazione. Questa viene misurata in numero di istanti di attivazione del task del VMS e non rappresenta pertanto una misura assoluta del tempo. Dato che gli intervalli di attivazione sono variabili, a causa della natura context-aware del sistema, la somma del numero di istanti di attivazione non rappresenta una misura del tempo assoluta ma soltanto relativa al livello di dinamicità della rete. Infatti uno stesso numero di istanti di attivazione, in due reti a dinamicità diverse, corrisponde ad un tempo totale differente nei due casi.

- **Probabilità di errore**: Dato che in un sistema simulato non avviene la trasmissione di messaggi attraverso interfaccia wireless, è necessario, al fine di avvicinarsi il più possibile alla situazione reale, realizzare un meccanismo di perdita di messaggi. Attraverso questo parametro si imposta la probabilità di perdita dei messaggi: si va da un minimo dello 0% in cui tutti i messaggi arrivano a destinazione fino ad un massimo del 50%.

- **Dimensione località**: E' possibile mediante questo parametro settare il raggio d'azione dell'LME che definisce, in questo caso, anche l'unica località presente nell'area simulata. Ora infatti la località non viene intesa come numero di hop dall'LME che la definisce, ma ristretta al concetto di range di comunicazione del nodo gestore. In questo ambiente simulato infatti è sufficiente avere un concetto di località più restrittivo: tutti i nodi ME presenti nell'area di copertura dell'LME appartengono alla località.

L'esecuzione realizzata attraverso il tool (lanciata mediante il tasto `run` della GUI) rappresenta una successione di istanti di attivazione dell'LME, pari al numero selezionato, che ricreano quello che si avrebbe in realtà. Prima dell'attivazione del task dell'LME però, il tool simula il movimento dei nodi, in modo tale che, una volta attivato, il nodo gestore riesca a cogliere i cambiamenti di topologia. I risultati della simulazione vengono raccolti all'interno di file di testo per permettere un'approfondita analisi a posteriori.

Un'assunzione fondamentale eseguita all'interno del simulatore, e che non rappresenta una situazione abbastanza realistica, è che tutti i nodi ME, a partire dal primo istante di attivazione, appartengono già al gruppo. Dato che la procedura di join richiede l'intervento del J/LMS, quest'ultima è stata evitata, in quanto il target della simulazione è esclusivamente il VMS, realizzando nodi appartenenti tutti al gruppo gestito dall'unico LME presente nell'area di simulazione.

L'interfaccia grafica del tool di simulazione permette di monitorare alcuni dei risultati di interesse. Sono stati implementati infatti un pannello di output per l'LME ed uno dedicato alle uscite degli ME, selezionabili uno per volta da un menu a tendina. Inoltre

viene visualizzato in output anche la lista dei PID membri del gruppo in ogni istante d'attivazione.

Una cosa fondamentale inoltre, al fine di comprendere l'evoluzione del sistema, è quella di osservare graficamente i cambiamenti di topologia della rete e vedere quali sono i nodi che in ogni istante entrano ed escono dalla località. A questo fine è stato necessario implementare nel simulatore un'area rappresentante la zona di simulazione, all'interno della quale posizionare tutti i nodi ME creati. In quest'area viene poi inserito l'LME gestore con l'indicazione della località da esso realizzata, al fine di osservare quali sono i nodi effettivamente appartenenti al gruppo. I nodi della località vengono indicati mediante dei pallini rossi ma, per poter essere identificati correttamente, è necessario riportare in figura anche delle altre informazioni, quali il PID all'interno del gruppo, la distanza dall'LME e il livello di batteria residuo.

E' importante notare che per rendere più realistica la simulazione, si è scelto di variare la distanza dall'LME in questo modo: quando il nodo è fuori località la distanza rimane invariata, mentre una volta all'interno di essa questa varia da un minimo di 1 hop ad un massimo di 11 hop a seconda della vicinanza o meno dal centro. Il termine hop viene inteso ora come unità di misura per valutare la distanza dal centro del cerchio che definisce la località. Per quanto riguarda invece il livello di batteria residuo si è scelto di generarlo in modo casuale per ciascun nodo in un range cha varia da 1% a 100%.

La Figura A.1 mostra l'interfaccia grafica dell'ambiente di simulazione.

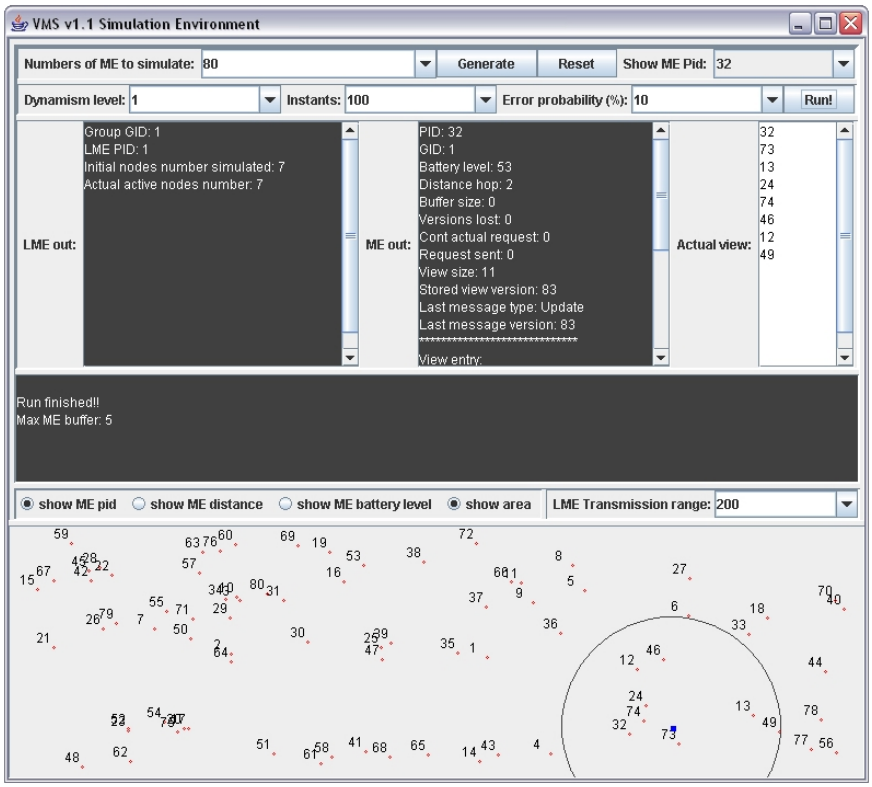

Figura A.1: GUI dell'ambiente di simulazione.

A.2 Risultati dei test

Sono stati eseguiti una serie di *run* di simulazione sulla versione definitiva dell'applicazione, con il vincolo già citato dell'implementazione ad hoc per tale ambiente. In ogni esecuzione sono stati variati i settaggi selezionabili dall'interfaccia in modo da simulare molteplici situazioni di rete.

I parametri osservati ed analizzati sono i seguenti:

- **Hit ratio**: Data la presenza di un buffer contenente dati da ritrasmettere è bene conoscere la percentuale di riferimenti trovati in memoria. Questo parametro misura la percentuale delle richieste andate a buon fine sul totale delle richieste ricevute.
- **Dimensione buffer**: Misura la dimensione in numero di byte occupati del buffer `cronologyBuffer`.
- **Intervallo di attivazione**: Essendo una peculiarità del VMS l'adattatività degli istanti di attivazione, si vuole conoscere, mediante questo parametro, l'evolversi degli istanti nel tempo. Ad ogni attivazione viene memorizzato nel file di log l'intervallo di successivo.
- **Copertura**: Questo parametro valuta la copertura della rete in rapporto alle richieste di ritrasmissione eseguite dai nodi. Si intende la percentuale di nodi, rispetto al totale, che possiedono la vista coerente, cioè stabile con la situazione di rete attuale.
- **Impiego di banda**: Mediante questo parametro si misura il numero di byte utilizzati ad ogni attivazione per la trasmissione degli aggiornamenti periodici dall'LME. Questo valore include il numero di byte per le eventuali ritrasmissioni.

A.2.1 Prima simulazione

In questa prima prova è stata scelta una rete poco dinamica con livello di dinamicità pari a 1, un numero di nodi nell'area di simulazione pari a 100, una probabilità di perdita dei messaggi pari al 20% ed una durata pari a 1000 istanti di attivazione.

Osservando i risultati delle pagine successive si nota da subito l'adattatività del VMS rispetto ai cambiamenti topologici. Il grafico relativo agli istanti di attivazione ci riporta infatti un comportamento più o meno standard fino all'istante 450 in cui l'intervallo rimane prevalentemente tra gli 8 e 10 secondi. Da quell'istante in poi invece la situazione diventa più dinamica e si osservano delle contrazioni continue dell'intervallo di attivazione. Tutto questo si riflette anche sulla quantità di dati inviati per gli aggiornamenti e quindi anche sulla dimensione del buffer della cronologia. Questi infatti, a partire dall'istante 450, iniziano a crescere e mantengono un andamento duale fino alla fine della simulazione.

Per quanto riguarda le ritrasmissioni risulta giusta la scelta della gestione FIFO per il buffer, ottenendo infatti una hit ratio del 100%. Non si verifica nemmeno una miss per tutta la durata della simulazione. La differenza tra il numero di richieste ed il numero di hit rappresenta il numero di richieste non soddisfatte a causa di un rifiuto da parte dell'algoritmo.

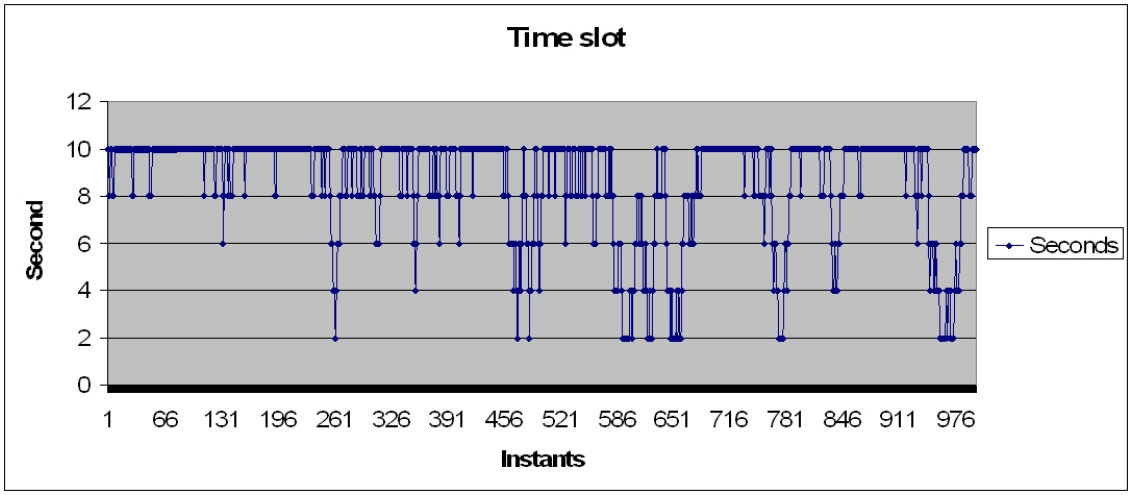

Figura A.2: 1° Simulazione: Istanti di attivazione.

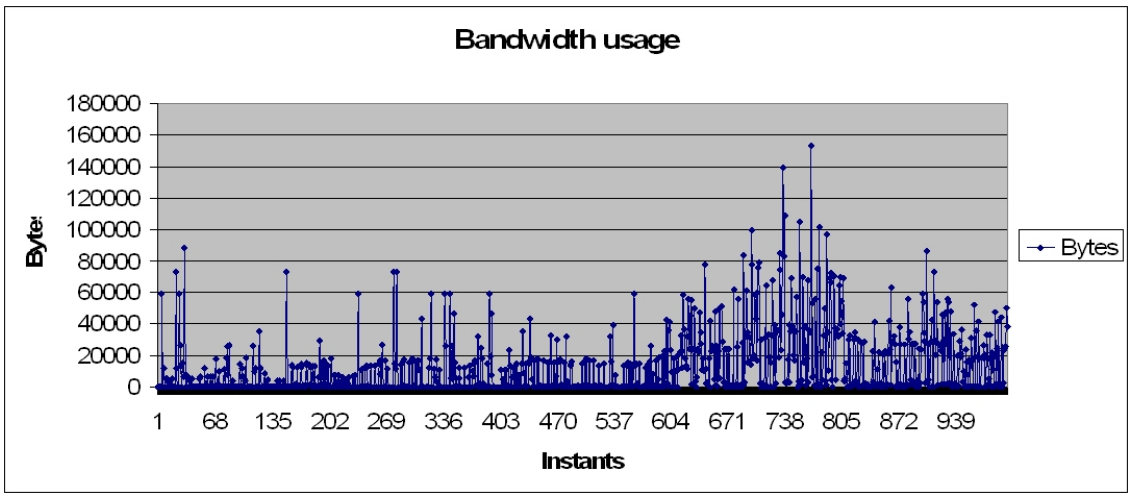

Figura A.3: 1° Simulazione: Utilizzo di banda.

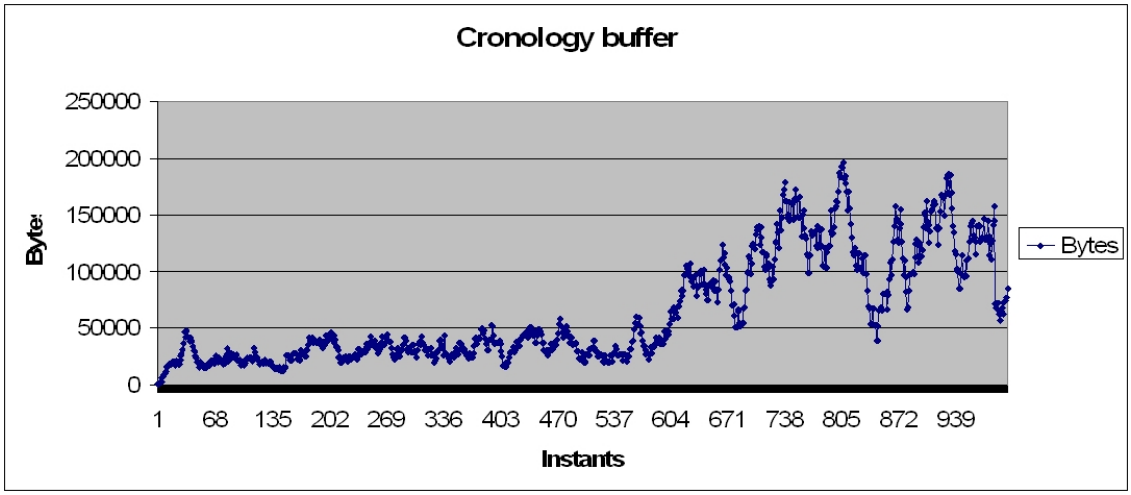

Figura A.4: 1° Simulazione: Dimensione buffer cronologia.

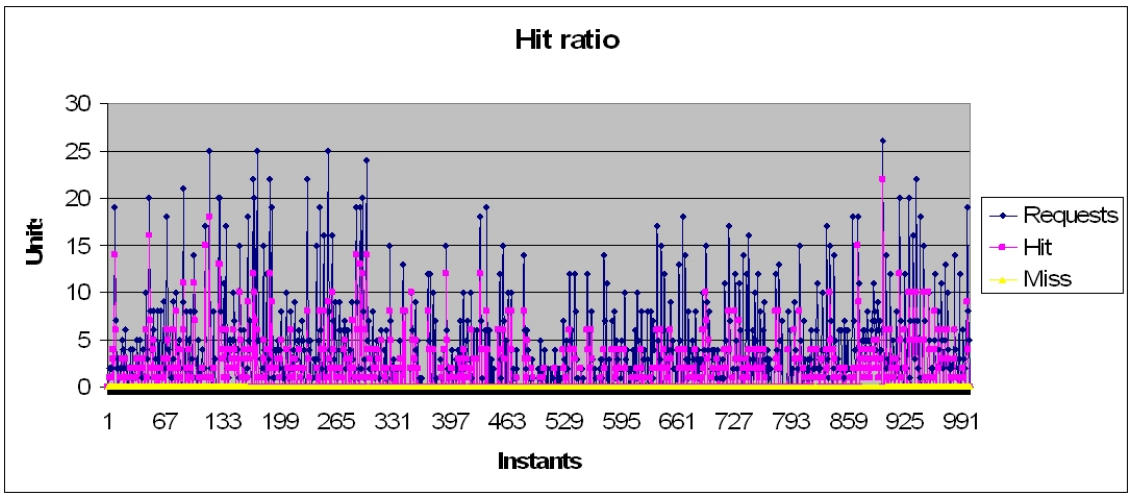

Figura A.5: 1° Simulazione: Hit ratio dei riferimenti in cache.

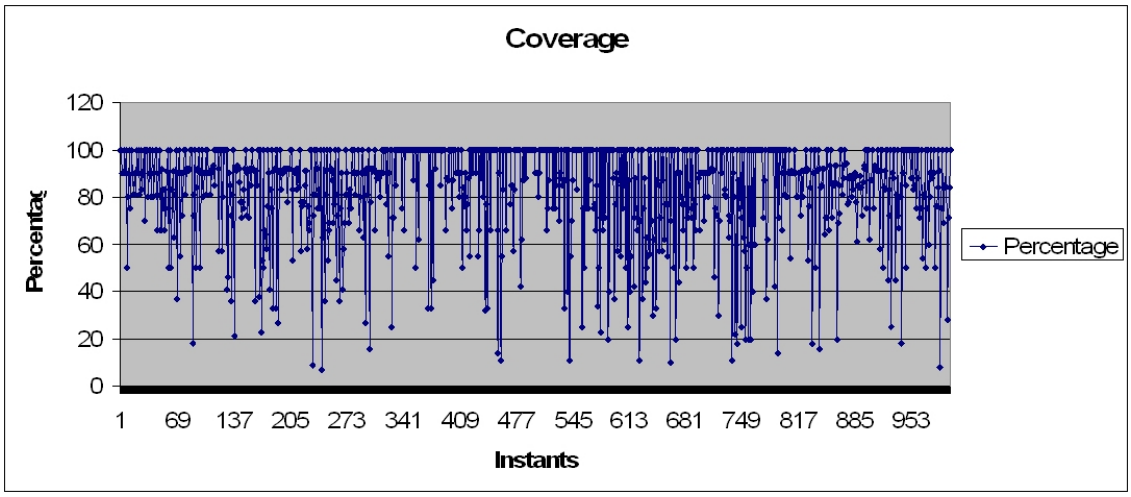

Figura A.6: 1° Simulazione: Copertura.

Infine il grafico relativo alla copertura ci mostra un andamento piuttosto variabile, con la prevalenza però di valori compresi tra l'80% ed il 100%. Le brusche cadute in questo grafico sono dovute ad un numero eccessivo di rifiuti da parte dell'algoritmo.

La seguente tabella riporta alcuni risultati rilevanti del test.

Intervallo medio	$8.57s$
Tempo simulato	$2h, 23m$
Richieste	3723
Hit	1703
Miss	0
Rifiuti	2020
Hit ratio	100%
Copertura media	83.77%
Occupazione media buffer	$63.6Kb$
Aggiornamento medio	$12.9Kb$

Tabella A.1: Prima simulazione.

A.2.2 Seconda simulazione

In questo secondo test si sono variate le condizioni della rete rendendola più dinamica e meno affidabile dal punto di vista della consegna dei messaggi. Si è infatti impostato un livello di dinamicità pari a 3 ed una probabilità di errore ora del 33%. Il numero dei nodi presenti nell'area di simulazione è rimasto invariato a 100 ME ed il numero di istanti di attivazione è stato abbassato a 500 unità.

Osservando il grafico di Figura A.7 relativo agli istanti di attivazione si nota che, anche con un piccolo aumento del livello di dinamicità della rete, ora la maggioranza degli intervalli si assesta nella fascia tra i 4 e gli 8 secondi, pur avendo alcuni abbassamenti sui 2 secondi ed alcuni innalzamenti sui 10 secondi. Questo dimostra la bontà dell'algoritmo di adattamento dell'intervallo che segue il livello di dinamicità della rete. Tutto sommato la situazione ottenuta non presenta un'andamento caratteristico ma un caso medio in cui si alternano tratti con piccoli cambiamenti ad altri con grandi variazioni.

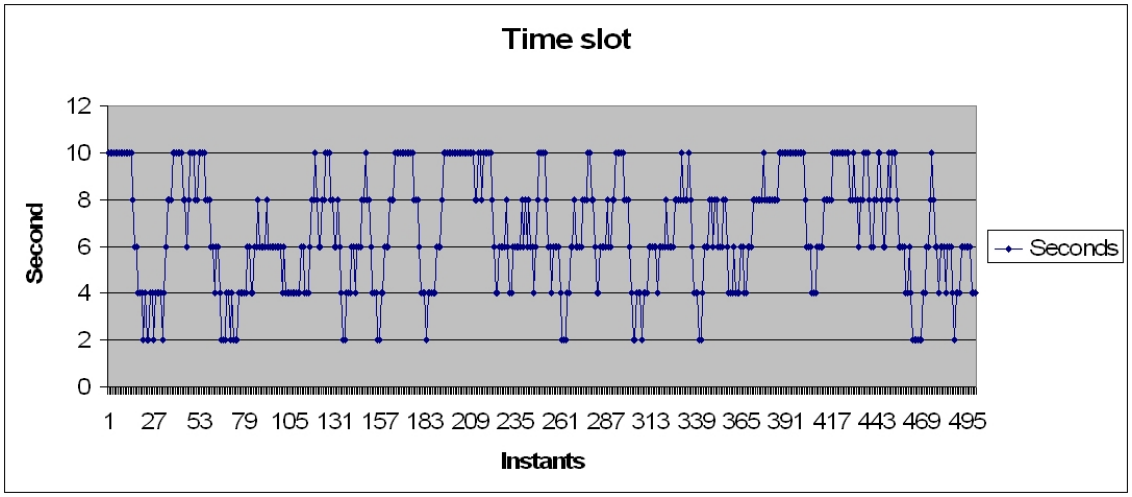

Figura A.7: 2° Simulazione: Istanti di attivazione.

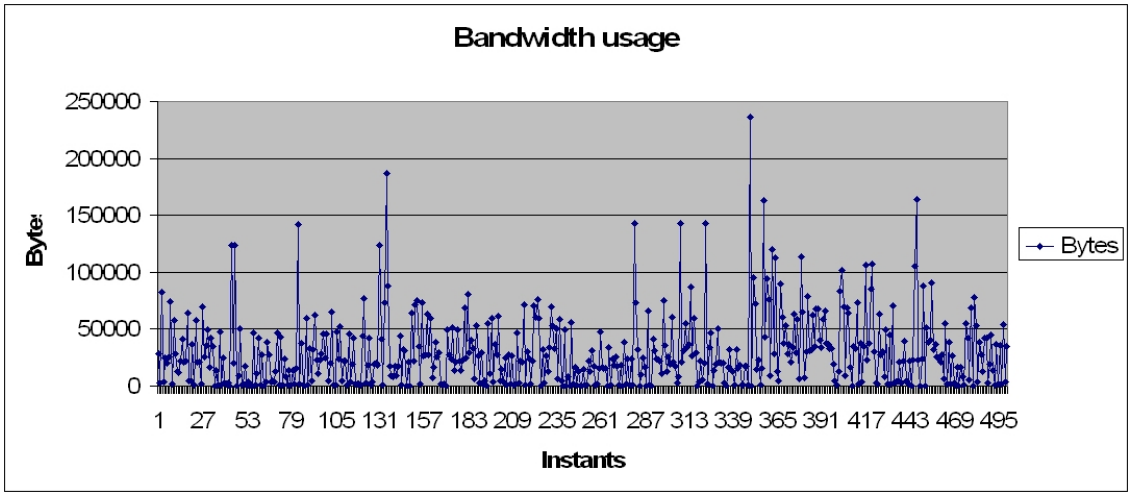

Figura A.8: 2° Simulazione: Utilizzo di banda.

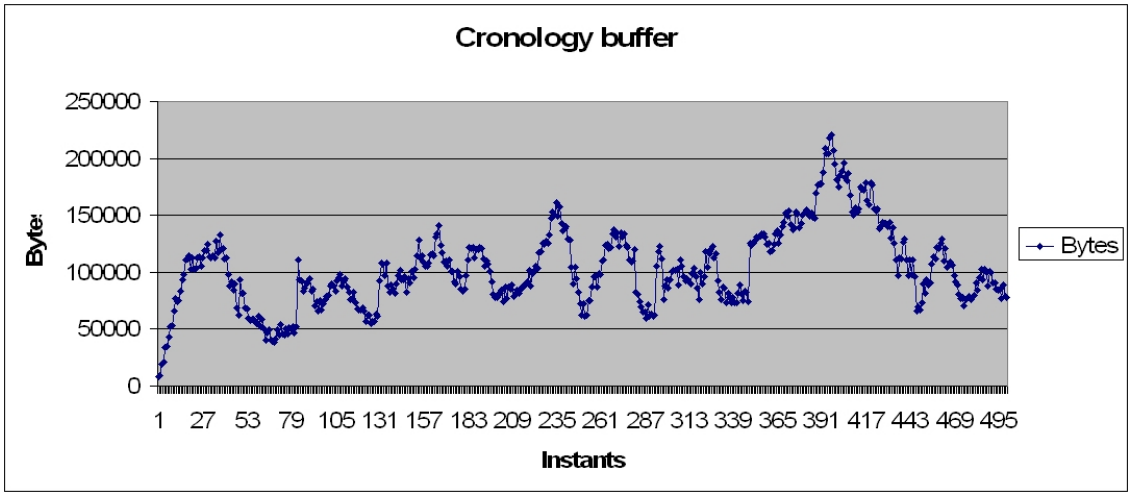

Figura A.9: 2° Simulazione: Dimensione buffer cronologia.

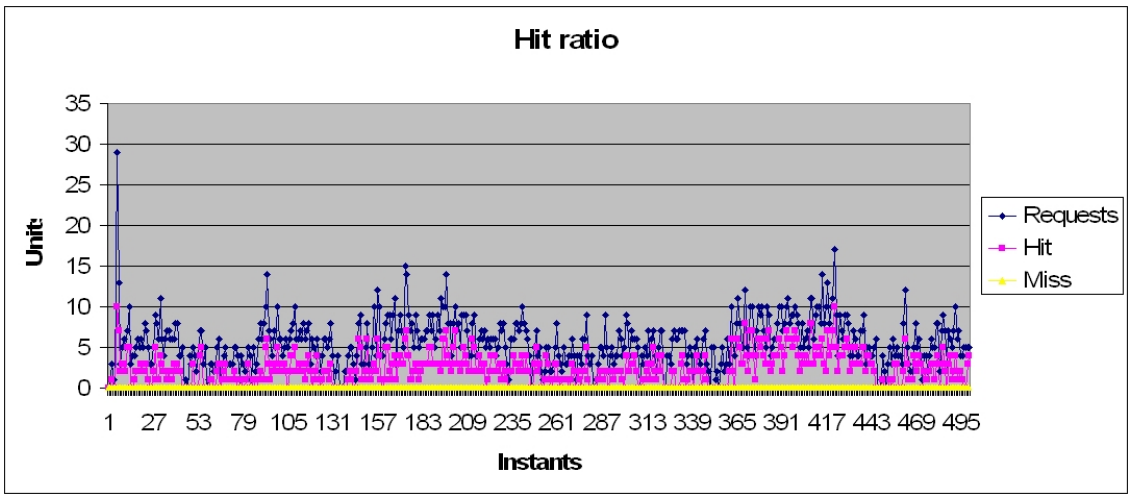

Figura A.10: 2° Simulazione: Hit ratio dei riferimenti in cache.

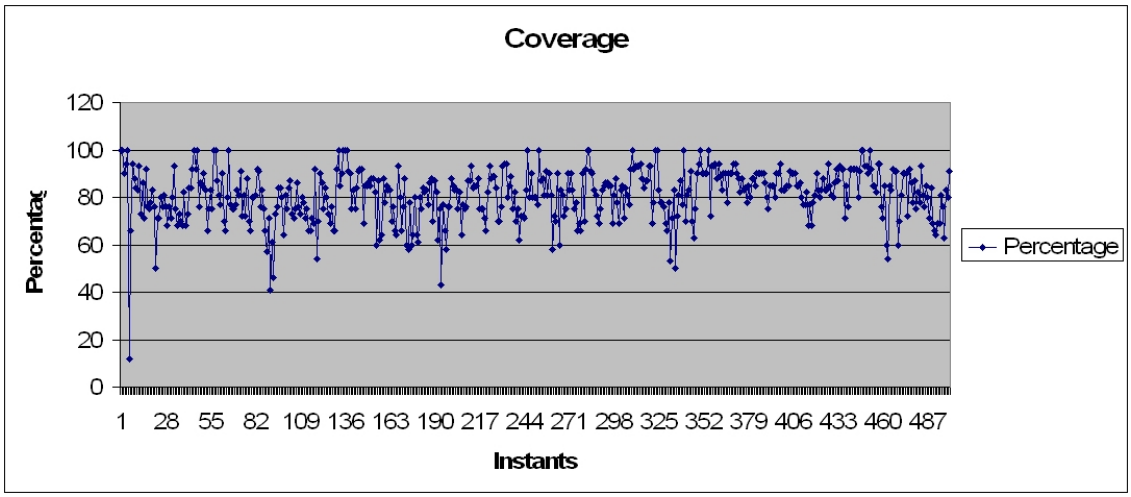

Figura A.11: 2° Simulazione: Copertura.

Questo andamento medio degli istanti di tempo si riflette ancora una volta sull'utilizzo di banda per l'invio degli aggiornamenti ed indirettamente anche sulla dimensione del buffer contenente tali aggiornamenti. Nel primo si osserva un'andamento più o meno costante con valori quasi sempre sotto i $100Kb$ con un piccolo aumento verso la fine a partire dagli istanti intorno al 350. Questa situazione si riscontra anche nella dimensione del buffer che subisce questo aumento a partire da quell'istante, ma rimanendo comunque per quasi tutta la durata della simulazione sempre in una fascia tra i $50Kb$ ed i $150Kb$.

Per quanto riguarda la hit ratio si ottengono di nuovo ottimi risultati riscontrando nessuna miss per tutta la durata della simulazione. La copertura, anche se diminuita rispetto al caso precedente, rimane sempre su ottimi valori all'interno di una fascia tra il 60% ed il 100%.

La tabella seguente riporta i risultati principali del test.

Intervallo medio	$6.7s$
Tempo simulato	$56m$
Richieste	2801
Hit	1167
Miss	0
Rifiuti	1634
Hit ratio	100%
Copertura media	80.9%
Occupazione media buffer	$102.9Kb$
Aggiornamento medio	$29.2Kb$

Tabella A.2: Seconda simulazione.

A.2.3 Terza simulazione

In questo terzo test si rendono le condizioni della rete ancor più dinamiche ed ancor meno affidabili del caso precedente. Viene scelto un livello di dinamicità pari a 5 ed una percentuale di errore del 50%. Il numero di nodi simulati nell'area viene aumentato a 200 ed il numero di istanti di attivazione è di nuovo pari a 1000.

Il risultato più rilevante è quello degli istanti di attivazione e che indirettamente si ripercuote anche sulla dimensione degli invii e del buffer. Essendo ora la rete molto dinamica il numero di cambiamenti di topologia nell'unità di tempo è elevato, pertanto l'algoritmo di adattamento sente la necessità di procedere più velocemente. Osservando il grafico infatti si nota che gli istanti di attivazione rimangono per tutta l'intera simulazione tra i 2 ed i 4 secondi, con delle eccezioni sui 6 ed 8 secondi.

Il campionamento più frequente della topologia si dovrebbe riflettere anche sul grafico della banda utilizzata per gli invii presentando un grafico con valori sempre molto bassi. La situazione in realtà è proprio questa, ma il grafico, dato che considera anche i byte utilizzati per le ritrasmissioni, risulta spurio a causa di un numero molto elevato di richieste e di ritrasmissioni (vedi grafico relativo). Questo è dovuto al fatto che la probabilità di errore selezionata per questo test è molto alta.

Anche il buffer della cronologia ne risente di questa alta dinamicità e presenta valori maggiori rispetto ai casi precedenti, ma comunque sempre al di sotto di $1Mb$. Questo aumento netto rispetto agli altri due test è dovuto anche al raddoppio in area simulata dei nodi presenti.

La hit ratio è ancora ottima nonostante il grandissimo numero di richieste da soddisfare presentando solamente 4 miss ed una percentuale di hit pari al 99.92%. La copertura rimane sempre nella fascia tra il 60% ed il 100%, con una media molto simile ai precedenti casi.

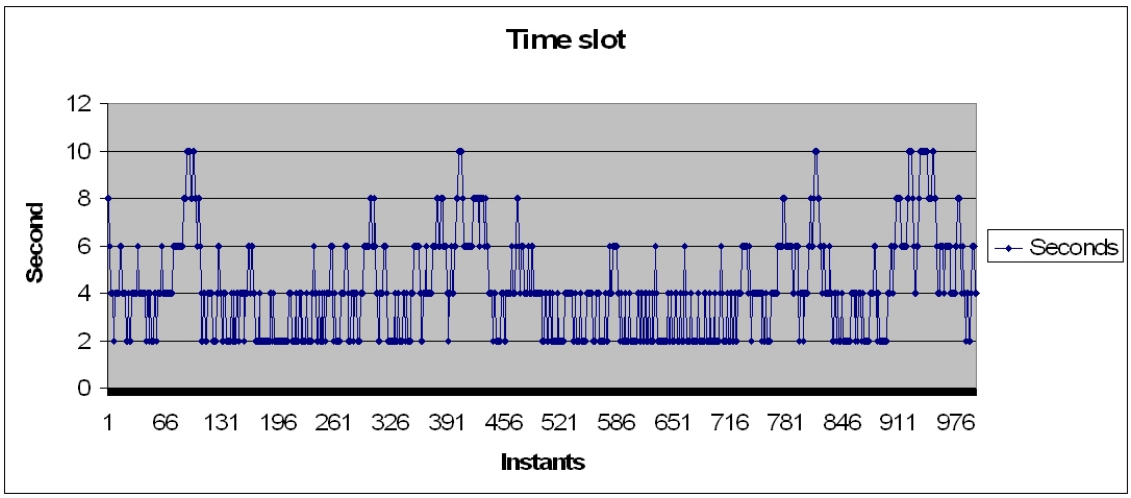

Figura A.12: 3° Simulazione: Istanti di attivazione.

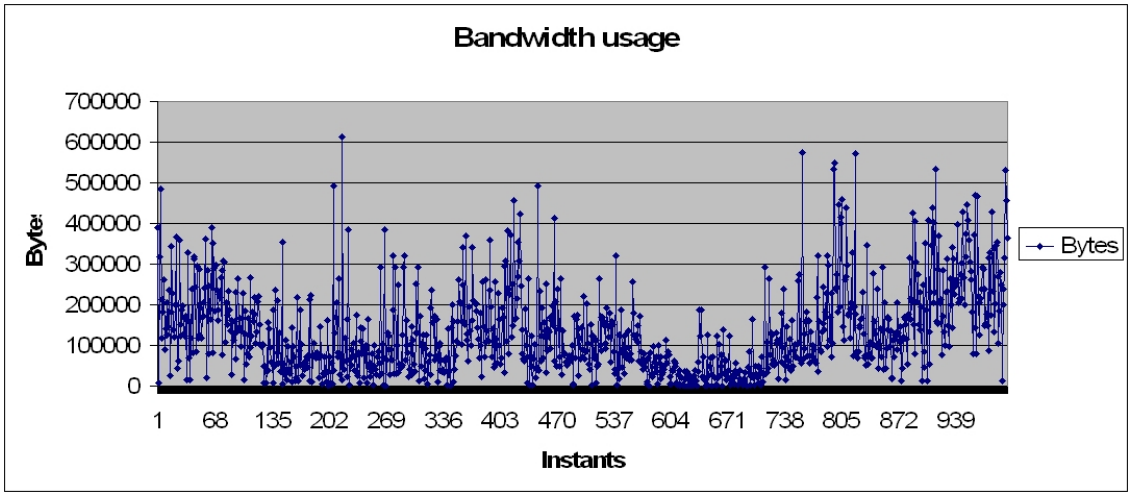

Figura A.13: 3° Simulazione: Utilizzo di banda.

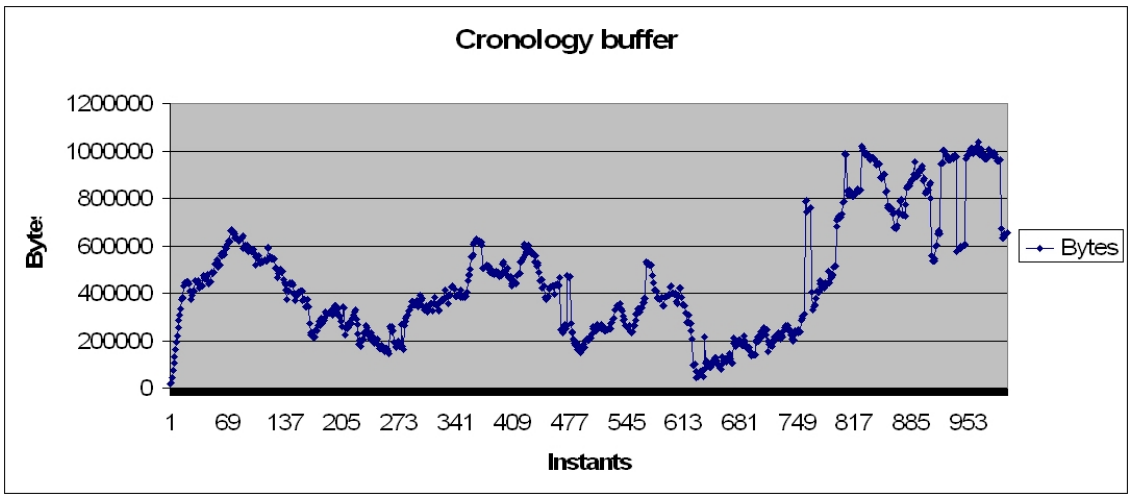

Figura A.14: 3° Simulazione: Dimensione buffer cronologia.

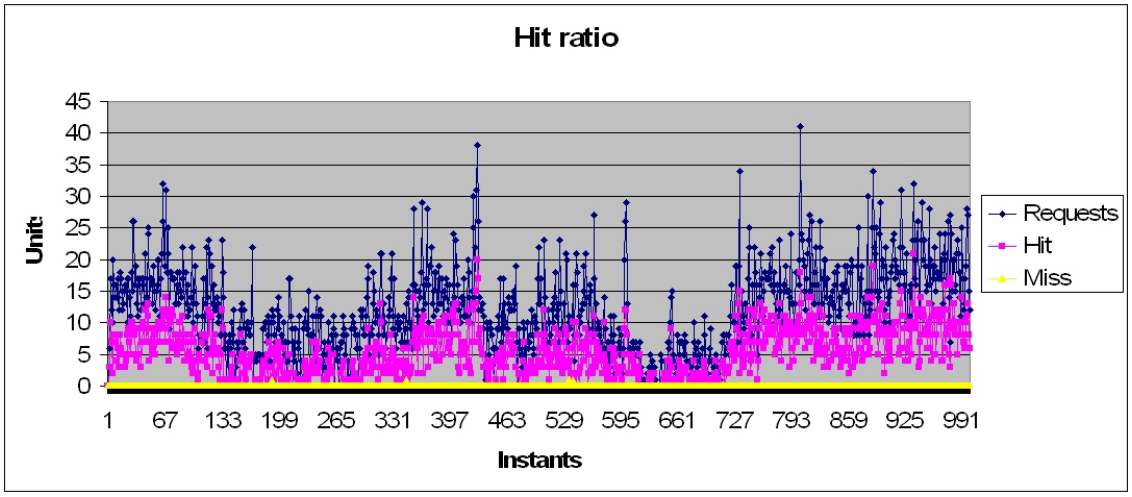

Figura A.15: 3° Simulazione: Hit ratio dei riferimenti in cache.

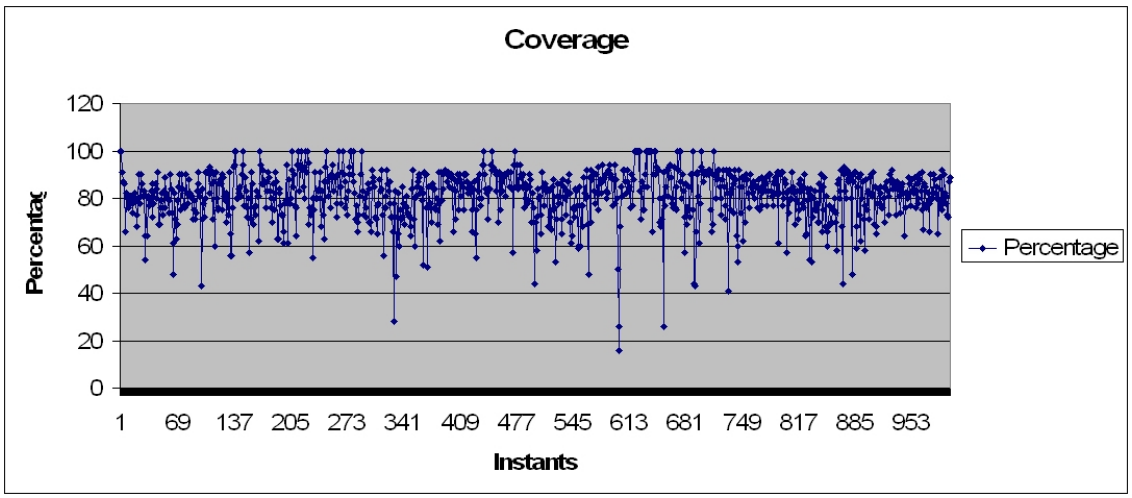

Figura A.16: 3° Simulazione: Copertura.

La tabella seguente riporta i risultati principali del test.

Intervallo medio	$4.13s$
Tempo simulato	$1h, 9m$
Richieste	11950
Hit	5308
Miss	4
Rifiuti	6638
Hit ratio	99.92%
Copertura media	81.2%
Occupazione media buffer	$455.2Kb$
Aggiornamento medio	$131.6Kb$

Tabella A.3: Terza simulazione.

A.2.4 Quarta simulazione

Si intende mostrare ora come, nel caso di alta dinamicità, l'algoritmo adattativo porti benefici in termini di banda utilizzata e di memoria occupata dal buffer. Si vuole pertanto ripetere un identico test, in assenza di errori in trasmissione, per non inquinare il grafico della banda, e con due livelli di dinamicità diversi, uno minimo ed uno massimo. Il risultato che ci si attende è quello di avere a grandi linee la stessa grandezza media degli aggiornamenti ed una stessa occupazione media del buffer.

Si esegue un primo test (caso A) con 100 nodi nell'area, probabilità dello 0%, 1000 istanti di attivazione e livello di dinamicità 1, poi si eseguirà lo stesso test con il livello di dinamicità elevato a 5 (caso B). Si riporteranno per ciascuno il grafico degli istanti di attivazione, dell'uso di banda e dell'occupazione del buffer.

Osservando i grafici a coppie si distingue immediatamente che l'unico tipo di grafico in cui balzano agli occhi delle vistose divergenze è quello degli istanti di attivazione. Chiaramente nel caso A ci si assesta intorno ai 10 secondi mentre nel caso B tra i 2 ed i 6 secondi. In pratica, al fine di ottenere grafici simili nell'impiego di banda e nella dimensione del buffer, il VMS del caso B è costretto ad andare più velocemente restringendo i propri intervalli di attivazione. Il risultato che si vuole evidenziare è proprio questo ed è una caratteristica chiave del componente sviluppato.

Il grafico relativo all'utilizzo di banda infatti mostra, in entrambi i casi , un'andamento pressochè identico, rimanendo sempre al di sotto dei $100Kb$ ed incontrando sporadiche situazioni al di sopra di questo limite. Questo andamento simile è ancor più visibile dai loro valori medi.

Infine anche il grafico del buffer della cronologia è molto simile presentando un'andamento che rimane nella stessa fascia di occupazione e con valori medi approssimabili.

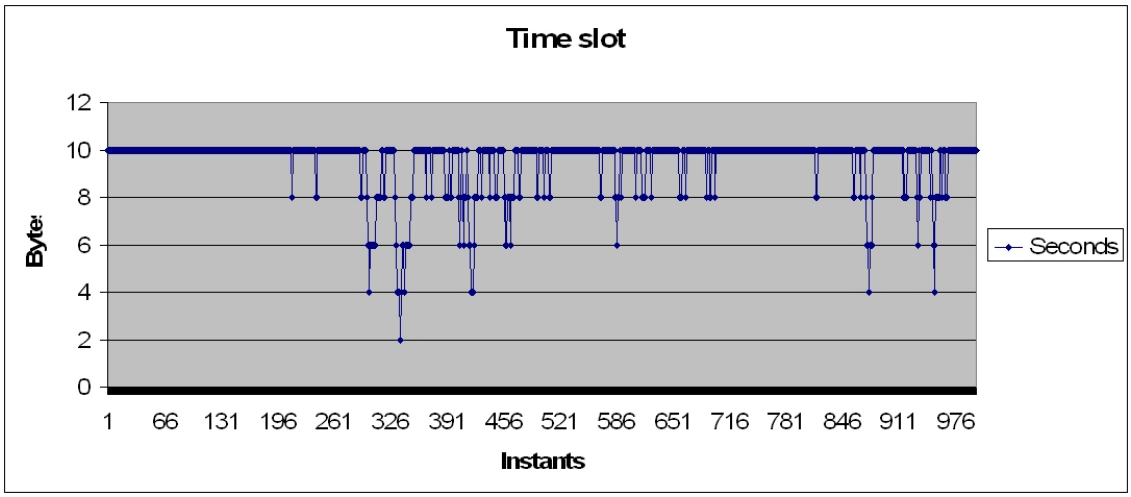

Figura A.17: 4° Simulazione: Istanti di attivazione (caso A).

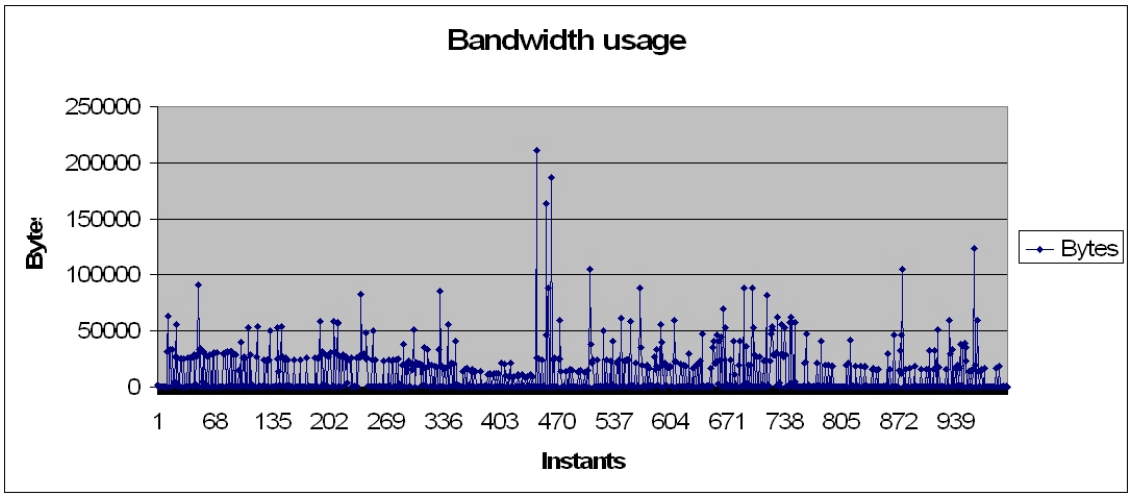

Figura A.18: 4° Simulazione: Utilizzo di banda (caso A).

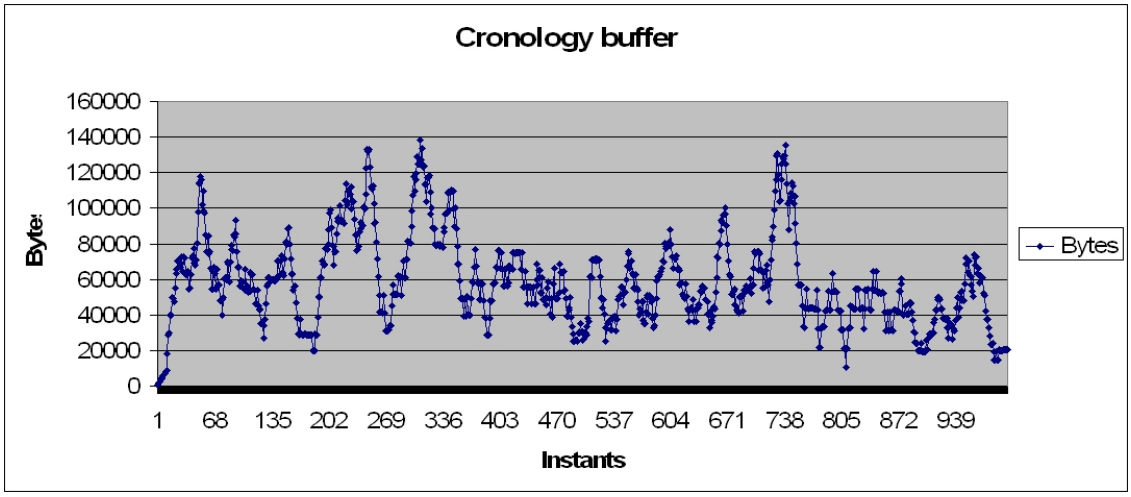

Figura A.19: 4° Simulazione: Dimensione buffer cronologia (caso A).

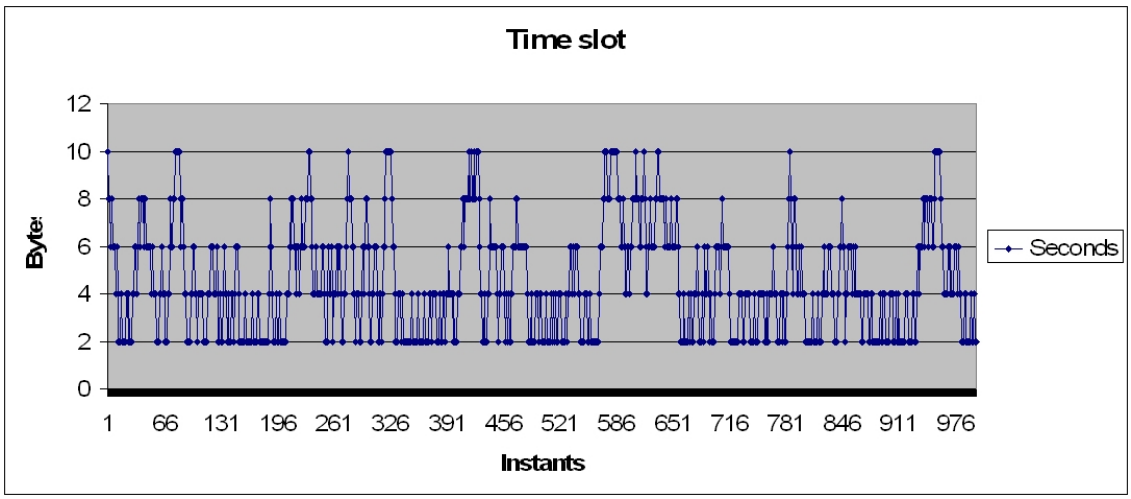

Figura A.20: 4° Simulazione: Istanti di attivazione (caso B).

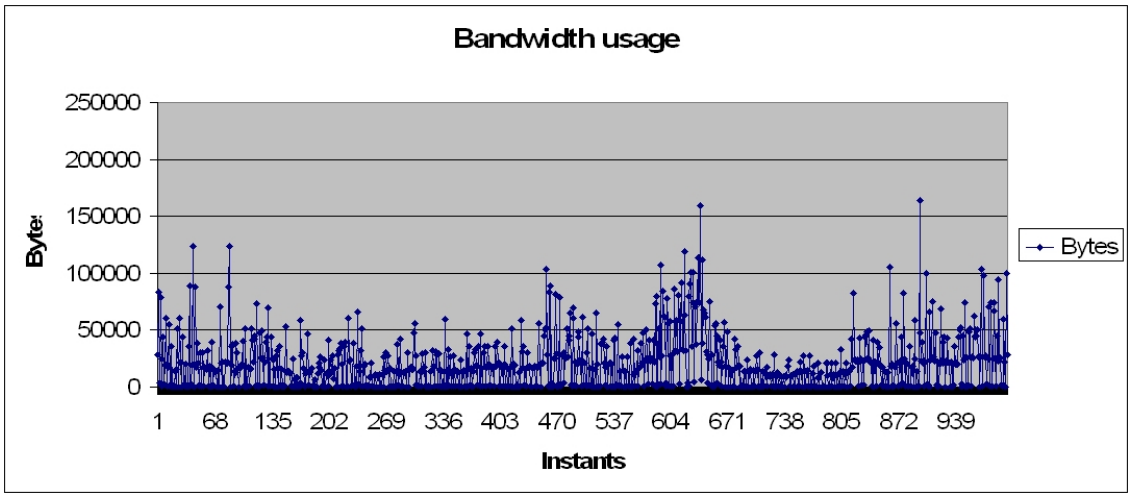

Figura A.21: 4° Simulazione: Utilizzo di banda (caso B).

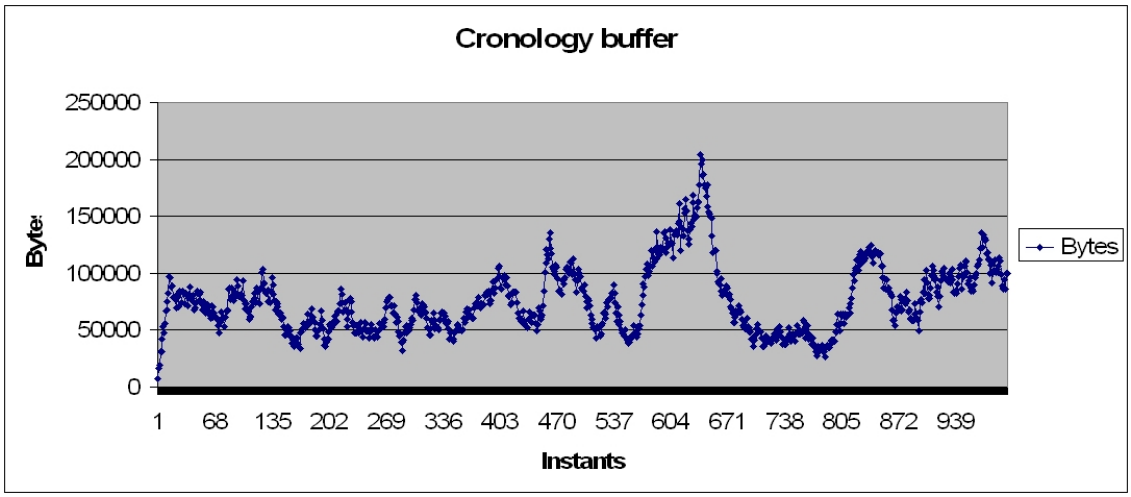

Figura A.22: 4° Simulazione: Dimensione buffer cronologia (caso B).

La seguente tabella mette a confronto i risultati di quest'ultima simulazione.

Parametro	**Caso A**	**Caso B**
Livello dinamicità	1	5
Intervallo medio	$9.55s$	$4.46s$
Tempo simulato	$2h, 39m$	$1h, 14m$
Occupazione media buffer	$58.4Kb$	$75.6Kb$
Aggiornamento medio	$11.5Kb$	$17.2Kb$

Tabella A.4: Quarta simulazione.

La tabella mostra come la durata complessiva della simulazione, cioè l'esecuzione di 1000 istanti di attivazione, sia eseguita nel caso a bassa dinamicità in 2 ore e 39 minuti, mentre nel caso ad alta dinamicità in solamente 1 ora e 14 minuti. Questa è la differenza sostanziale tra i due casi ed è causata dall'adattatività del sistema ai cambiamenti di topologia.

L'occupazione media del buffer e la dimensione media di un'aggiornamento sono del tutto paragonabili, e questo dimostra appunto l'efficacia dell'algoritmo di adattamento utilizzato. Occorre precisare però che al contempo, se da una parte diminuisce la quantità di dati da trasmettere, dall'altra, a causa della maggiore frequenza di esecuzione, si utilizzano maggiori risorse di calcolo, cosa che non sempre è adatta ai dispositivi mobili. Considerando però che tale algoritmo dovrà essere eseguito da nodi di tipo LME, di natura più potenti e non vincolati da risorse, l'adattatività può risultare una caratteristica da privilegiare rispetto la quantità e la complessità della computazione.

Appendice B

Politiche di caching

In ambito *mobile* la gestione della memoria risulta essere un aspetto critico da tenere in considerazione nel progetto di applicazioni software. A causa delle limitate risorse dei dispositivi wireless, è necessario infatti occupare sempre la minore dimensione possibile della memoria di sistema, mantenendo i thread attivi per il tempo strettamente necessario e deallocando il più frequentemente possibile gli oggetti e le strutture dati non più utilizzabili.

Nel sistema in esame vengono utilizzate e mantenute in memoria diverse strutture dati, alcune delle quali possono essere gestite mediante algoritmi, al fine di minimizzare l'occupazione di memoria. E' il caso del buffer contenente gli oggetti inviati periodicamente chiamato `cronologyBuffer` che rappresenta anche la struttura ad occupazione maggiore. Questa struttura è necessaria infatti per poter ritrasmettere degli aggiornamenti nel caso in cui i membri ne facciano richiesta, ma è del tutto inutilizzata nel caso in cui nessun membro richieda tali ritrasmissioni. E' possibile pertanto operare delle euristiche al fine di decidere se un entry del buffer potrà essere o meno utilizzata nell'immediato futuro e quindi scegliere o meno di eliminarla liberando spazio in memoria.

La tecnica attraverso la quale vengono eliminate le entry da una struttura dati, prende il nome di *politica di rimpiazzo*, ed in letteratura ne esistono diverse, ognuna con pregi e difetti. La scelta della tecnica più adatta dipende dal contesto applicativo e dal tipo di utilizzo della struttura dati da gestire. Le politiche di rimpiazzo sono importanti quindi, ogni qualvolta si dispone di poche risorse e si intende utilizzarle al meglio.

Nel caso del `cronologyBuffer` infatti è necessaria una politica di rimpiazzo che elimini gli aggiornamenti che non verranno probabilmente utilizzati nell'immediato futuro. E' possibile applicare a questo scopo politiche esistenti, quali la *Least Frequently Used* (LFU), la *Least recently Used* (LRU) oppure la *First In First Out* (FIFO), le quali non hanno come target specifico l'ambito wireless ma che risultano ugualmente del tutto adeguate. In alternativa è anche possibile scegliere la strada della creazione di una politica *ad-hoc*, al fine di realizzare il miglior algoritmo per il caso in esame.

E' stato sviluppato infatti un'algoritmo di rimpiazzo per il buffer della cronologia e lo si è comparato con altre due politiche esistenti, le quali sono risultate essere le più adatte tra tutte al caso in questione e sono la LFU e la FIFO. Attraverso questo confronto è stato possibile valutare e scegliere la politica con le migliori performance in termini di *hit ratio*[1]

[1]La *hit ratio* è la percentuale di riferimenti in cache andati a buon fine sul totale delle richieste.

ed occupazione di memoria. Al fine di valutare le performance di ognuna delle tecniche analizzate, sono state eseguite delle simulazioni mediante il tool di simulazione sviluppato ed esposto in Appendice A.

B.1 Soluzione proposta

Un algoritmo di caching adatto al caso di studio in esame deve essere basato sulla frequenza di utilizzo e deve eseguire un calcolo probabilistico riguardo le richieste che avverranno in futuro. Il buffer contenente la cronologia degli invii può essere gestito sia stabilendo una dimensione fissa per il numero di entry e rimpiazzando le entry mediante uno degli algoritmi di rimpiazzo, oppure essere gestito dinamicamente.

L'algoritmo proposto per questa gestione effettua una eliminazione dinamica delle entry che, in base ad una scelta probabilistica, ritiene non serviranno più nell'immediato futuro. L'idea alla base di questa soluzione è la seguente:

> Se, per un certo numero di istanti consecutivi l'LME non riceve alcuna richiesta di ritrasmissione per un determinato aggiornamento, è ragionevole pensare che anche nell'immediato futuro non ne riceva e quindi non sia più necessario il suo mantenimento in cache.

In pratica un'entry viene mantenuta in cache per tutto il tempo per cui il nodo riceve richieste e non appena, per una serie di istanti consecutivi, non ne riceva più, viene eliminata concludendo che con molta probabilità non sia più necessaria. Il valore di questo numero di istanti rappresenta una *finestra di attesa* per giudicare se un'entry è eliminabile o meno. Maggiore è il valore di questa finestra, indicata in seguito con il termine `WAIT`, maggiore è la probabilità che l'entry non venga più referenziata nell'immediato futuro. Mediante questo algoritmo la gestione del buffer avviene in modo dinamico variando il numero di entry in un range non limitato superiormente.

L'algoritmo di rimpiazzo del buffer viene mostrato in Figura A.1 ed è suddiviso in due parti fondamentali, una fase iniziale di *controllo* ed una di *eliminazione* dell'entry.

Durante la fase di controllo viene valutata l'adeguatezza del valore della finestra WAIT, mediante il controllo della percentuale di *miss*, cioè il numero di riferimenti falliti in cache, sul totale delle ritrasmissioni dell'ultima attivazione del task. Se tale percentuale è minore o uguale al 20% si può mantenere il valore di WAIT pari al valore della precedente attivazione, se invece, tale valore supera la percentuale limite, è necessario allargare la finestra di attesa al fine di ottenere una *miss ratio* inferiore. E' chiaro che aumentando il valore della finestra risulta più difficoltosa l'eliminazione delle entry dal buffer con conseguente aumento della sua dimensione. Nel caso in cui la finestra venga aumentata, il nuovo valore viene mantenuto per un tempo pari a `K` istanti di attivazione, trascorsi i quali è possibile decrementarla di un'unità fino a ritornare, nel caso di successive ritrasmissioni a bassa miss ratio, al valore di default (pari a due istanti[2]). Mediante questa fase di controllo è possibile modificare dinamicamente i criteri di eliminazione delle entry.

[2]Il valore della finestra pari a due istanti è stato scelto in seguito a simulazioni e ritenuto adeguato in rapporto alla dimensione del buffer ed al valore della miss ratio.

Una volta valutata l'adeguatezza della finestra `WAIT` si passa alla fase di eliminazione vera e propria. Si preleva pertanto un'entry dal buffer e si verifica se in quel determinato istante ci sono richieste per quella versione. Nel caso ci siano viene mantenuto un contatore `CONT` a zero, il quale indica il numero di istanti consecutivi per cui non si hanno richieste per quel determinato numero di versione. Se invece in quell'istante non è stata ricevuta alcuna richiesta il contatore viene incrementato di un'unità. Se è stato raggiunto il valore della finestra è possibile eliminare l'entry altrimenti si continua a mantenere l'oggetto nel buffer.

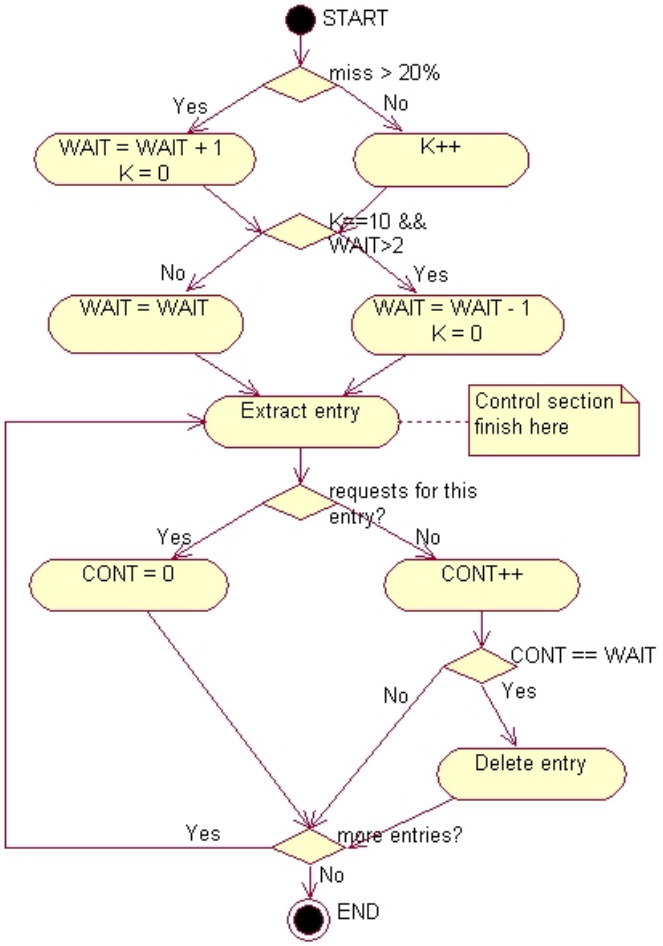

Figura B.1: State diagram dell'algoritmo di rimpiazzo proposto.

L'algoritmo appena presentato può essere visto come un caso particolare dell'LFU che rimpiazza le entry meno frequentemente referenziate e che verrà trattato nel seguente paragrafo.

B.2 Least Frequently Used

Questa politica di sostituzione si basa sulla frequenza dei riferimenti in memoria. La LFU mantiene un contatore per ogni entry del buffer che misura il numero di accessi e mediante confronto elimina ad ogni attivazione sempre l'entry con valore del contatore più basso. Il valore di questi contatori viene aggiornato ogni qualvolta un'entry riceve una richiesta. Per poter capire in quale istante occorre eliminare l'entry selezionata, l'algoritmo LFU deve conoscere necessariamente a priori la dimensione della cache in numero di entry. Quando viene raggiunto il valore limite e si intende inserire nel buffer una nuova entry, l'algoritmo esegue l'eliminazione per lasciare il posto al nuovo inserimento.

Tale algoritmo eliminando sempre l'entry meno acceduta, fornisce buoni risultati nei casi in cui la probabilità di accesso alle pagine di memoria è la stessa per tutte, e quindi con molta probabilità quelle meno accedute in un certo intervallo di tempo, saranno anche quelle meno accedute nell'immediato futuro. Nel caso in esame però le entry in cache non hanno tutte la stessa probabilità di essere richieste ma esiste una probabilità decrescente all'aumentare del tempo di presenza in buffer. Le entry più vecchie infatti avranno meno probabilità di essere accedute di quelle appena entrate e pertanto, ai fini di una buona gestione della cache, nel nostro caso è necessario tenere conto anche di questo aspetto. L'LFU puro pertanto non si adatta al meglio al caso di studio in esame in quanto non tiene conto del tempo di permanenza in cache.

A causa di questa mancanza di LFU il numero di miss in cache risulta abbastanza elevato. Una modifica alla dimensione del buffer non aiuta al miglioramento delle prestazioni perchè rimane il problema della non considerazione dell'età dell'entry. Inoltre LFU può soffrire anche di un'ulteriore problema nel caso in cui, invece del numero di entry massimo, si considera la dimensione in byte della memoria. Può accadere infatti che venga eliminata un'entry il cui spazio liberato in memoria non è sufficiente all'inserimento di una nuova, forzando magari l'eliminazione di un'ulteriore entry.

E' evidente come la soluzione proposta del paragrafo precedente sia una specializzazione della LFU, in quanto elimina sempre in base alla frequenza degli accessi ma lo fa solo quando non giungono richieste di ritrasmissione per un determinato numero di istanti. Si vanno in pratica a misurare non più il numero di accessi ma il numero di istanti consecutivi in cui non si sono verificate richieste. Grazie a questa modifica la dimensione della cache varia dinamicamente nel tempo e non deve necessariamente fissare una dimensione per capire quando rimpiazzare.

B.3 First In First Out

La politica FIFO è la più semplice delle tre analizzate ed è quella col minore carico computazionale. In questo caso viene rimpiazzata sempre l'entry che risiede da più tempo in cache, cioè la prima ad essere entrata tra tutte quelle presenti. Questa caratteristica di FIFO la rende particolarmente adatto al caso in esame in quanto rimpiazza sempre l'entry più datata che sarà, tra tutte, quella con la minore probabilità di essere referenziata in futuro. Ancora una volta come nel caso precedente ci si trova però di fronte al problema

di dimensionare la cache in numero di entry in modo tale da permettere all'algoritmo di capire quando è il momento di eliminare. Dimensionando opportunamente il buffer mediante prove sperimentali si possono ottenere buone performance da questa tecnica di rimpiazzo applicata al suddetto caso di studio.

A causa della necessità di fissare una dimensione massima del buffer, tale politica può presentare problemi nel caso questa dimensione sia troppo piccola. Al contrario invece, maggiore è la dimensione della cache, minore è il tasso di miss, con un conseguente aumento della dimensione in termini di byte occupati. Dimensionando opportunamente il buffer, eseguendo un trade-off tra la miss ratio e lo spazio occupato, si ottengono nel caso in esame ottime performance.

B.4 Analisi delle performance

Al fine di testare il funzionamento delle tre politiche al caso di studio e poter essere confrontate, sono stati effettuati dei test mediante il tool di simulazione utilizzato per il progetto del componente. Per ognuno di essi sono state analizzate le tre politiche di sostituzione nelle medesime condizioni di rete. Sono stati implementati pertanto tre buffer, ognuno gestito secondo una delle tre politiche, ed è stata effettuata un'analisi comparativa delle performance. Si sono ripetuti pertanto tre esperimenti in diverse condizioni di dinamicità della rete e di percentuale di perdita dei messaggi, su di un'intervallo di osservazione di 50 istanti di attivazione.

B.4.1 Prima simulazione

Questo primo caso simula una rete poco dinamica (livello 1 del simulatore) con percentuale di errore del 33%. La dimensione della cache nel caso di politica LFU e FIFO è pari a 10 entry. I grafici dei risultati sono riportati nella penultima sezione dell'appendice.

Come si può notare la soluzione proposta presenta un bassissimo tasso di miss a differenza di LFU che, dopo una fase iniziale, presenta sempre miss ed in alcuni casi anche in numero elevato. La politica FIFO invece si dimostra molto affidabile presentando, solo in un singolo caso, una miss in cache contro le quattro della soluzione proposta.

Nel grafico di Figura B.5 viene mostrato il numero di miss a confronto nei tre casi. La LFU mostra, come ci si aspettava, l'andamento peggiore a causa della mancanza di informazioni riguardo l'età delle entry, eliminando molto spesso anche entry appena memorizzate e quindi molto probabilmente referenziabili. La soluzione proposta e la FIFO invece hanno ottimi comportamenti in quanto rimpiazzano sempre le entry più datate, con un piccolo vantaggio della più semplice FIFO. A causa di performance in linea ed alla bassa capacità computazionale è preferibile in questo primo test la scelta di una politica FIFO.

Nel grafico di Figura B.6 infine viene mostrata l'evoluzione del numero di entry in cache nel caso di utilizzo dell'algoritmo proposto. La figura mostra una fase di caricamento iniziale in cui non si verificano richieste ed una volta a regime, il numero di entry presenti in cache rimane sempre all'interno di una fascia compresa tra le 6 e le 12 entry. Un vantaggio di questa soluzione quindi è quello di non dover decidere a priori la dimensione da fissare

al buffer, lasciando all'algoritmo stesso il compito di adattarla a seconda della situazione di rete.

B.4.2 Seconda simulazione

In questo secondo test viene mantenuta la dinamicità della rete e il tasso di errore come nel caso precedente e viene solamente modificato il valore del contatore CON (vedi 4.3.3.2) sul lato client, che corrisponde al numero di tentativi da eseguire prima di smettere di fare richieste di ritrasmissione. Nel caso precedente infatti tale contatore era pari a 5 tentativi mentre ora è stato impostato a 10. Questa modifica si ripercuoterà sulla dimensione del buffer nel caso della politica proposta in quanto, arrivando più richieste, risulta più difficile sostituire le entry.

In questa seconda prova il numero di miss della soluzione proposta è cresciuto ad 8 unità contro le 4 del caso precedente. La LFU si comporta in modo pessimo come nel caso precedente mentre la FIFO ha ancora una volta ottenuto ottimi risultati riscontrando un totale di 7 miss. Ancora una volta la FIFO rappresenta la soluzione a miglior performance, anche grazie alla minore occupazione di memoria, in confronto alla soluzione proposta. Quest'ultima infatti ora a regime mantiene sempre un numero di entry compreso in una fascia tra 12 e 16 unità.

B.4.3 Terza simulazione

In quest'ultima prova viene eseguito un confronto solamente tra la politica FIFO e la soluzione proposta in quanto è stata dimostrata l'inefficacia della politica LFU applicata al caso in esame. Viene impostato il numero di tentativi `CON` pari a 5 e viene scelta una dimensione del buffer in modalità FIFO pari a 5 entry, in modo tale da evidenziare i problemi di questa politica al diminuire della dimensione della cache. E' da notare però che al fine di avere un confronto realistico tra le due politiche, è necessario impostare una dimensione per la FIFO intorno alle 10 o 15 unità, in quanto queste ultime sono all'incirca il numero di entry medio occupato dall'algoritmo proposto. Con una dimensione del buffer pari a 15 entry infatti, simulazioni con politica FIFO hanno dimostrato un comportamento infallibile dal punto di vista del numero di miss che sommato al fatto di avere una complessità minima rappresenta la soluzione ottima.

Osservando i risultati è evidente da subito l'aumento del numero di miss nel caso di politica FIFO, che in questo caso sale a 9 miss su 50 istanti di attivazione, mentre la soluzione proposta non incorre in nessun miss per tutto il tempo di funzionamento. Anche la dimensione del buffer in numero di entry è diminuito a causa della diminuzione della finestra CON assestandosi su valori compresi nella fascia tra le 4 e le 10 unità, con valor medio di 7.

In questo caso particolare quella proposta rappresenta la soluzione migliore non avendo nessuna miss ed occupando in cache un numero di entry medio pari a 7 unità, contro i 9 miss ed una dimensione pressochè uguale ottenuti dalla politica FIFO.

B.4.4 Risultati delle simulazioni

Le seguenti figure mostrano i risultati dei test su un totale di 50 istanti di attivazione. E' da notare che nei grafici riguardanti i risultati delle ritrasmissioni di ogni politica, la differenza tra il numero di richieste e il numero di hit, cioè di ritrasmissioni, rappresenta il numero di richieste rifiutate dall'algoritmo delle ritrasmissioni visto in 4.3.5.3. Il numero di miss rappresenta pertanto il numero di richieste soddisfacibili non andate a buon fine a causa della mancata presenza in cache.

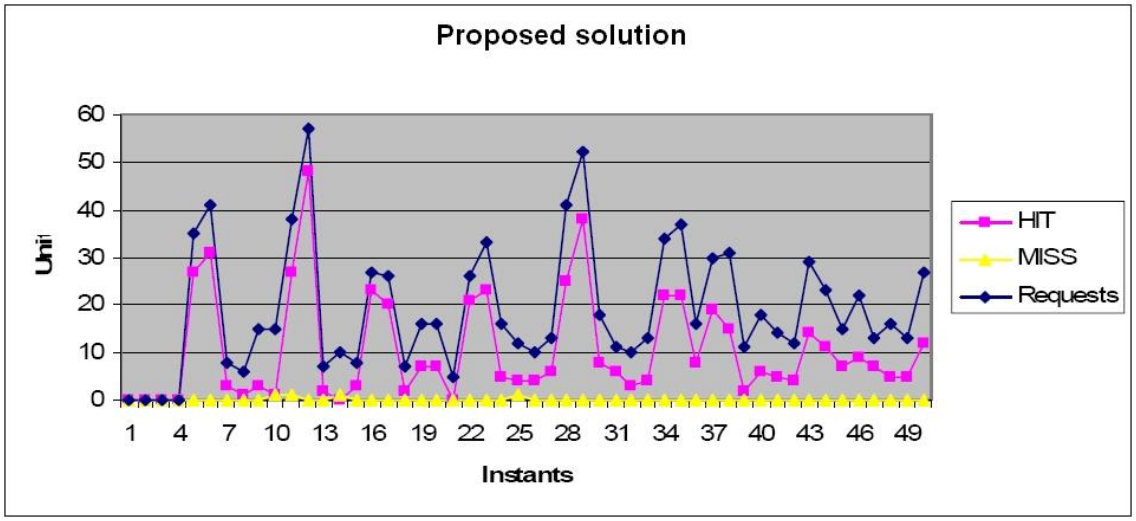

Figura B.2: 1° simulazione: Soluzione proposta.

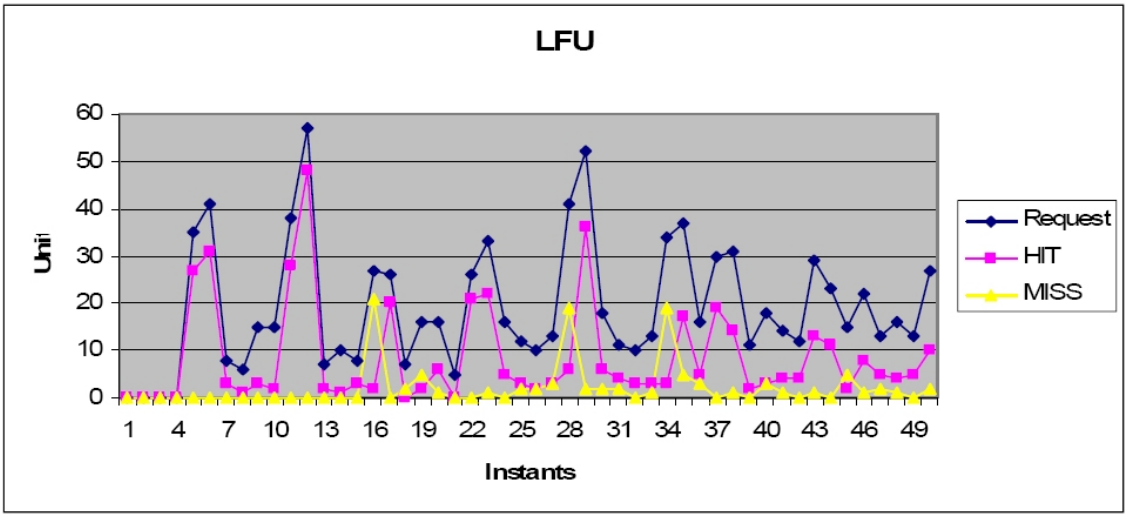

Figura B.3: 1° simulazione: LFU.

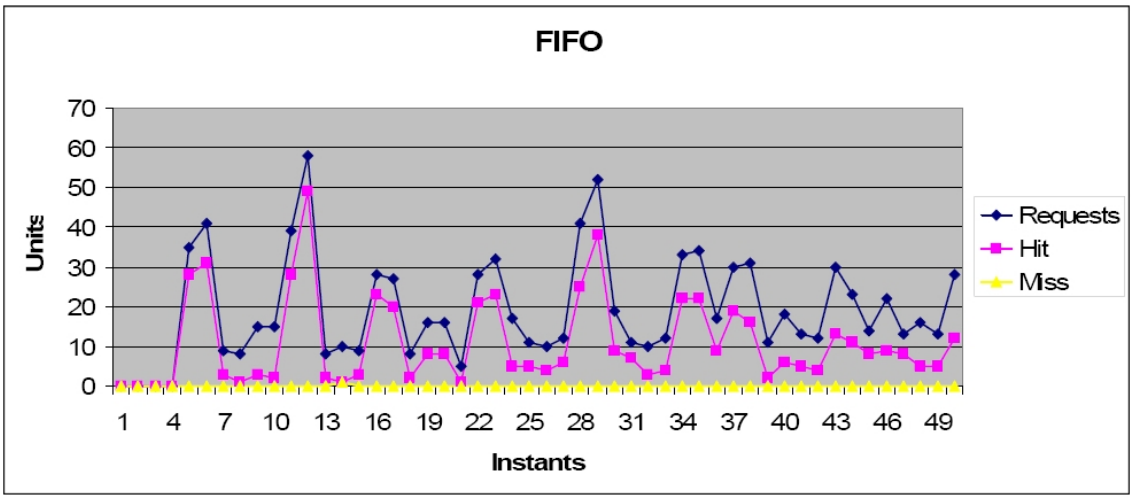

Figura B.4: 1° simulazione: FIFO.

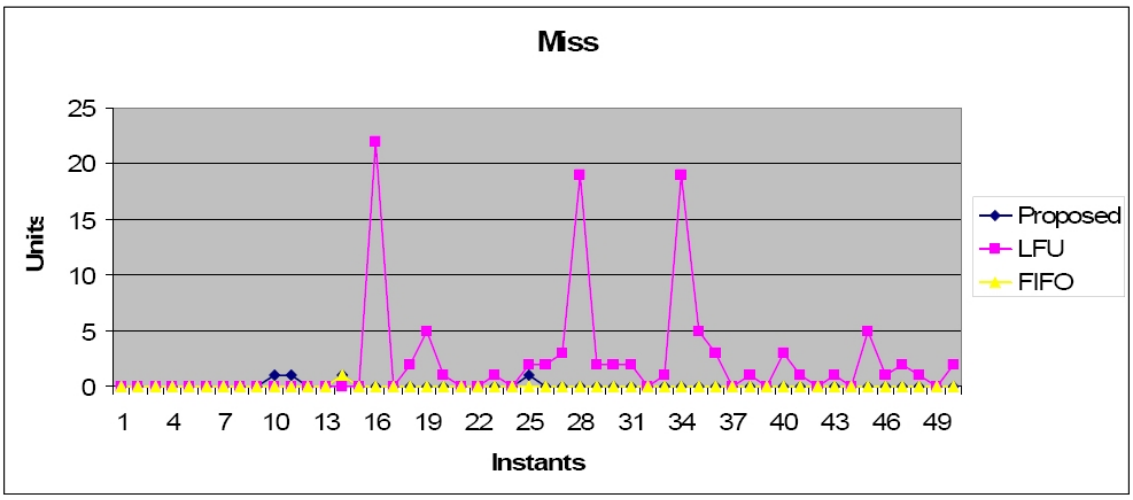

Figura B.5: 1° simulazione: Rapporto dei Miss nei tre casi.

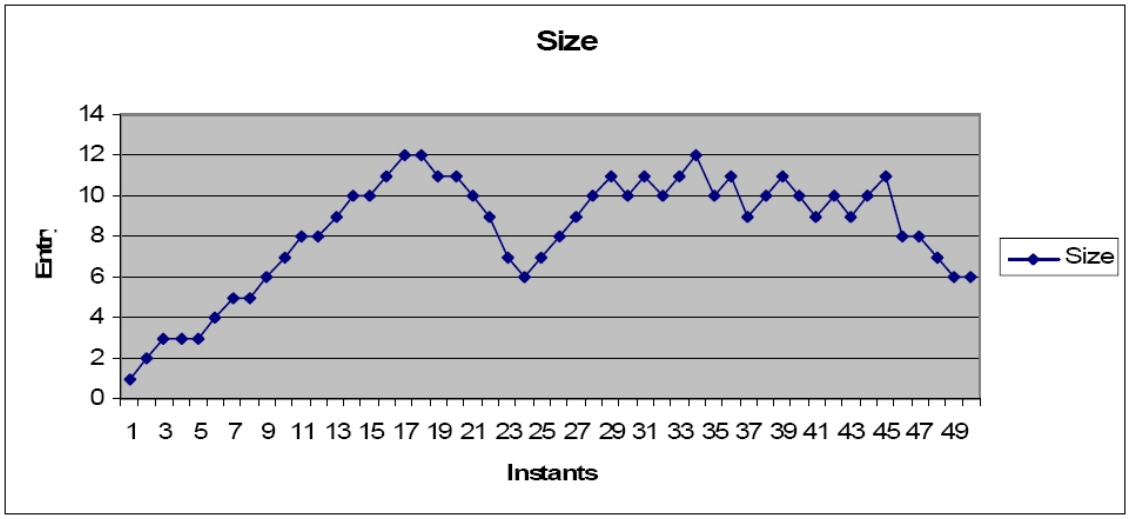

Figura B.6: 1° simulazione: Dimensione buffer cronologia.

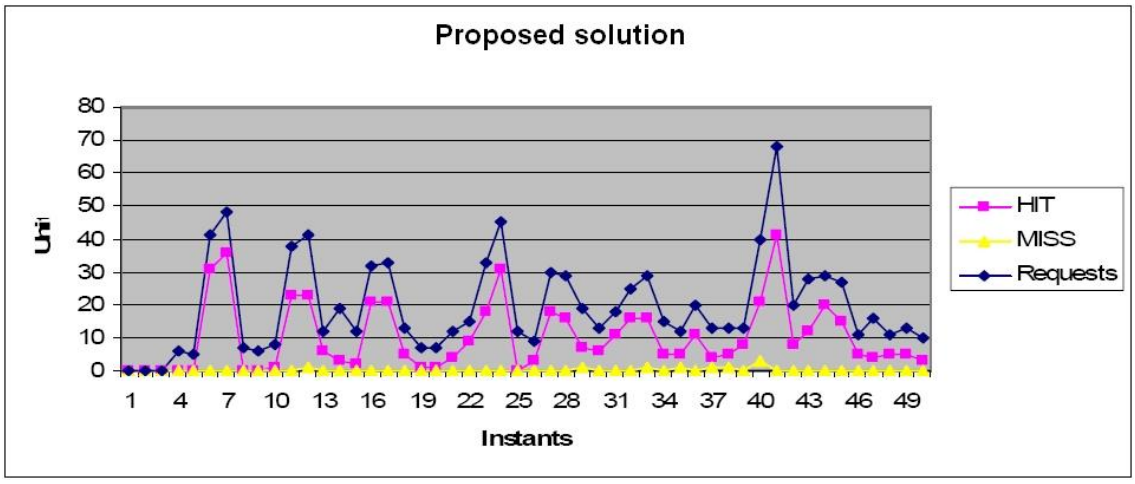

Figura B.7: 2° simulazione: Soluzione proposta.

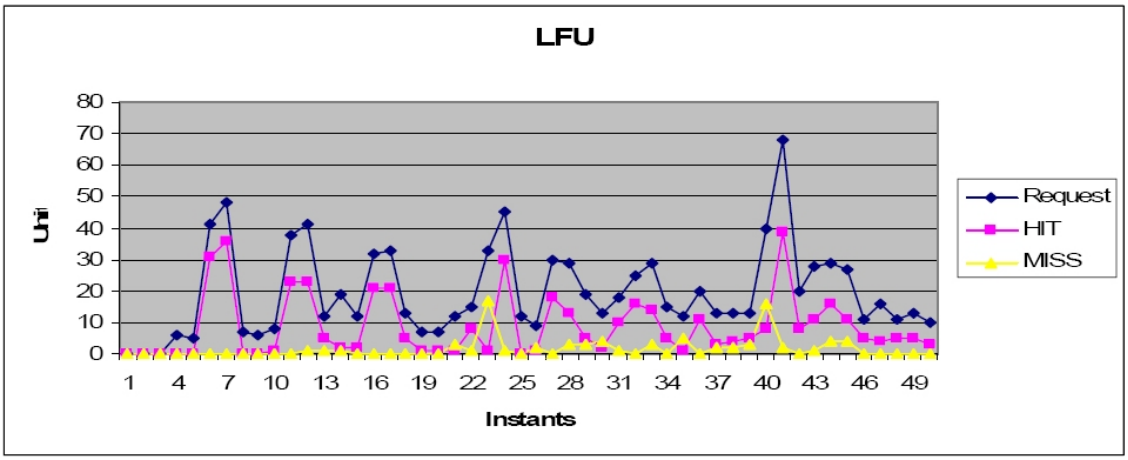

Figura B.8: 2° simulazione: LFU.

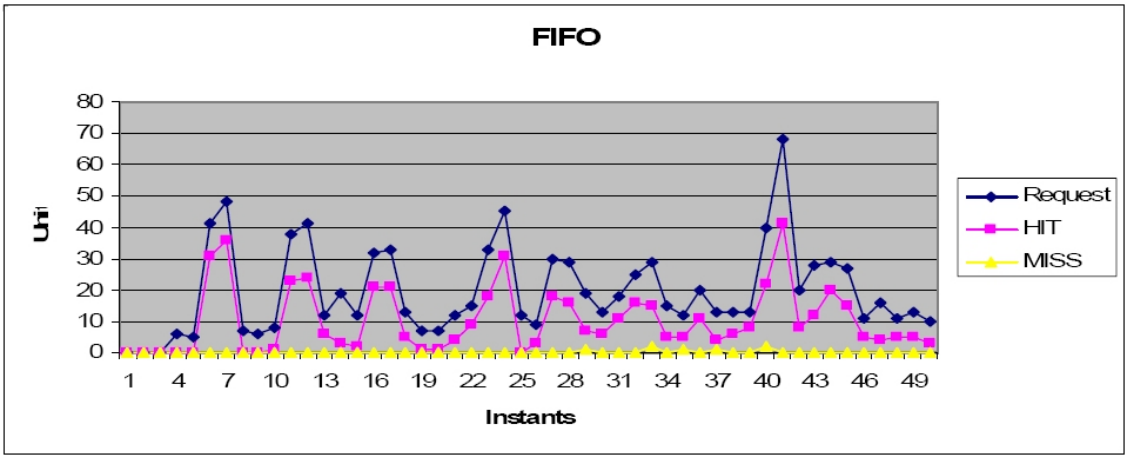

Figura B.9: 2° simulazione: FIFO.

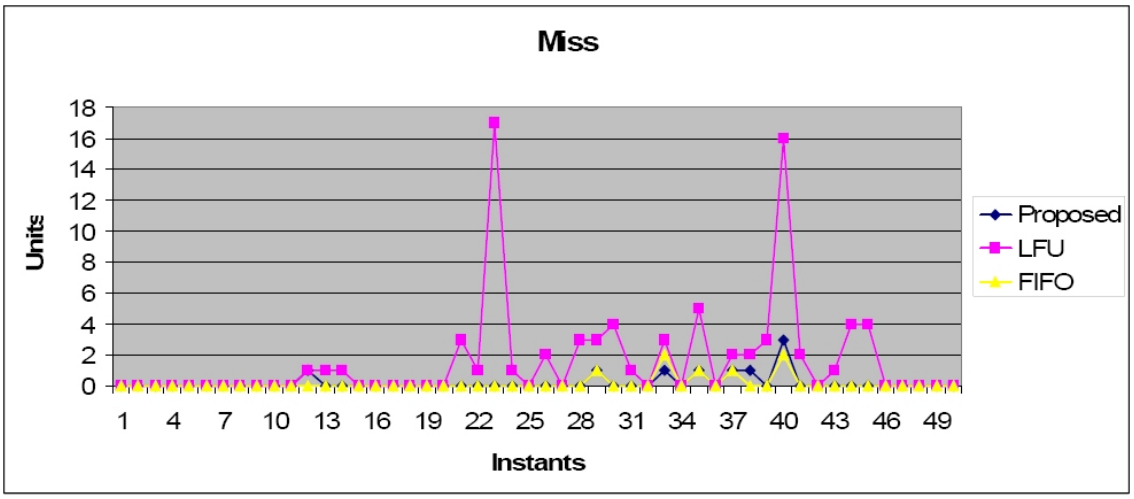

Figura B.10: 2° simulazione: Rapporto dei miss nei tre casi.

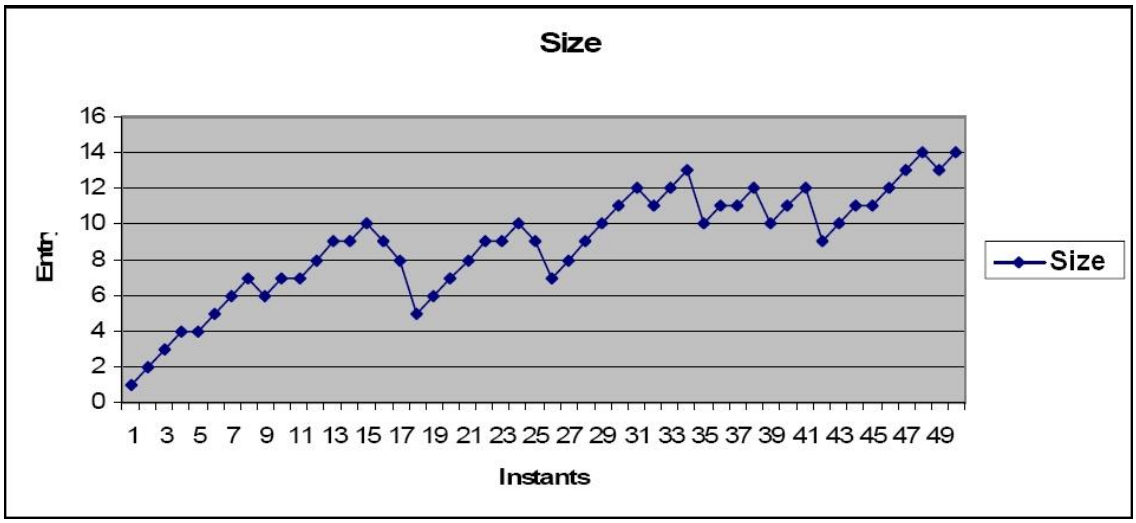

Figura B.11: 2° simulazione: Dimensione buffer cronologia.

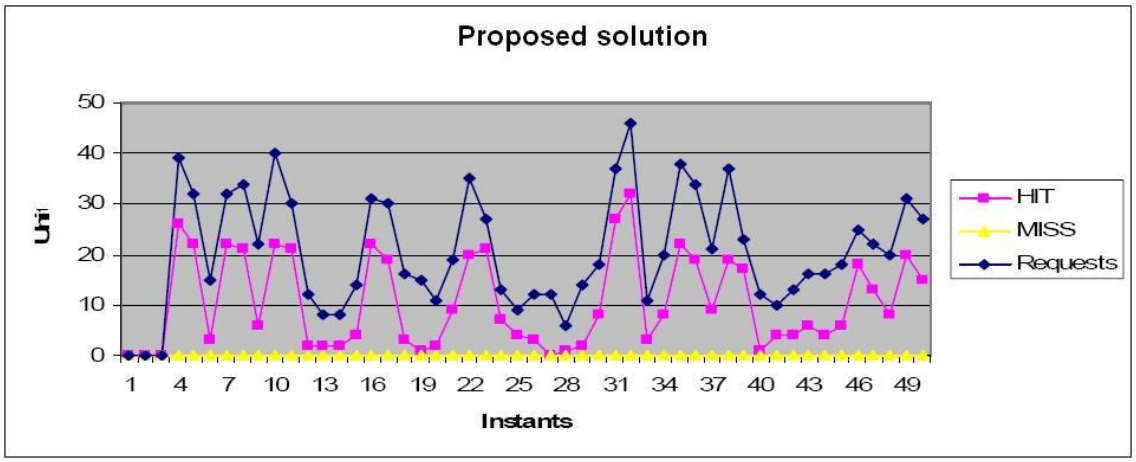

Figura B.12: 3° simulazione: Soluzione proposta.

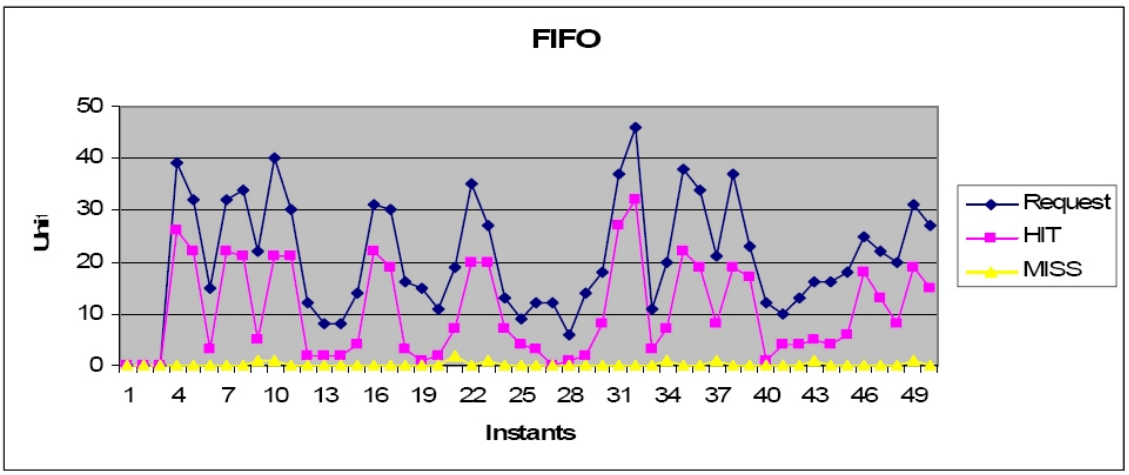

Figura B.13: 3° simulazione: FIFO.

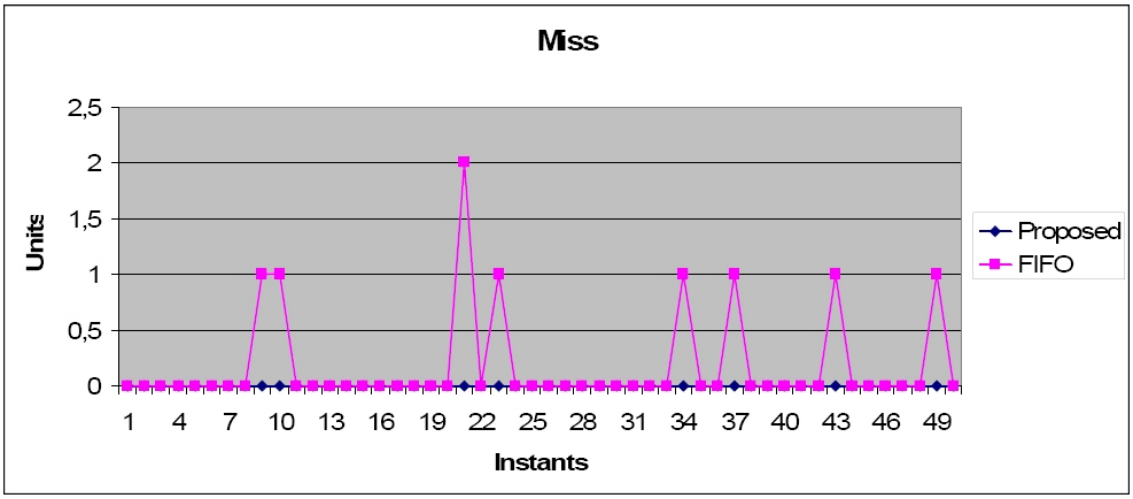

Figura B.14: 3° simulazione: Rapporto dei miss nei due casi.

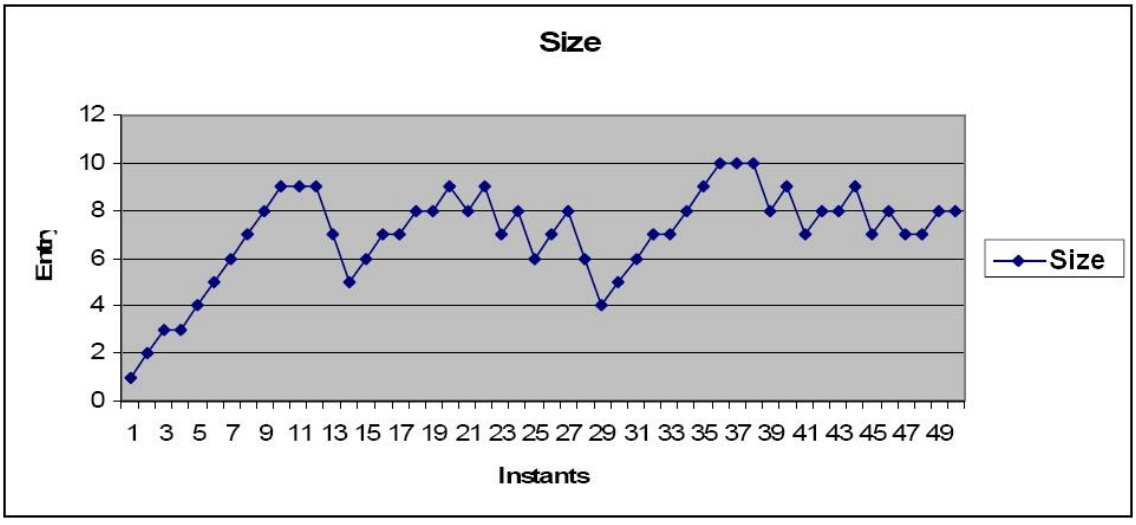

Figura B.15: 3° simulazione: Dimensione buffer cronologia.

B.5 Conclusioni

La soluzione LFU risulta essere la peggiore tra le tre politiche analizzate per i motivi precedentemente elencati e viene pertanto immediatamente esclusa dal confronto.

La politica FIFO invece si presta ad essere, nel caso in esame, un'ottima soluzione per la sua semplicità d'uso e per le ottime performance riscontrate. E' da notare però che i risultati ottenuti con queste simulazioni sono probabilistici, in quanto gli errori di trasmissione degli aggiornamenti avvengono con distribuzione probabilistica, e quindi non rappresentano un caso generale ma solo una simulazione di quella che potrà la situazione reale. In ogni caso i dati forniti dalle simulazioni dimostrano che con dimensione adeguata, la politica FIFO possiede un'altissima hit ratio.

La soluzione proposta presenta un buon comportamento, data la natura ad hoc per cui è stata sviluppata. Si hanno grossi vantaggi soprattutto dal punto di vista di gestione della memoria occupata, in quanto si demanda all'algoritmo il compito di variare la sua dimensione in rapporto alla situazione di rete. Non è necessario pertanto limitare superiormente a priori tale dimensione lasciandola evolvere naturalmente dall'algoritmo stesso. Uno dei punti di forza di questa soluzione risulta essere pertanto l'utilizzo efficiente della memoria mediante la gestione dinamica del buffer, occupando in ogni istante sempre lo stretto necessario per cercare di soddisfare al meglio le richieste di ritrasmissione. La soluzione proposta pertanto presenta i pregi di entrambe le tecniche: è una politica di sostituzione basata sulla frequenza come LFU ed elimina indirettamente le entry più datate come in FIFO.

Nonostante questo, la politica che risulta avere il miglior compromesso tra complessità computazionale, occupazione di memoria e hit ratio, risulta essere la FIFO, che, con una dimensione pari a 15 entry, ha dimostrato godere delle migliori performance. La politica adottata nella gestione del buffer `cronologyBuffer` è pertanto la FIFO.

Appendice C

MobiEmu

MobiEmu è una piattaforma software che permette di emulare reti ad hoc di qualunque scala e con qualunque scenario di mobilità, senza il reale movimento fisico dei nodi.

C.1 Architettura

MobiEmu usa una rete fissa di n macchine Linux per emulare una MANET di n nodi. In una rete ad hoc reale la connettività tra i nodi della topologia è dinamica, dal momento che i nodi entrano ed escono frequentemente dal range di comunicazione di ogni altro nodo. MobiEmu simula questa situazione del mondo reale sulla rete emulata, modificando dinamicamente i filtri dei pacchetti (mediante la tecnica del *packet filtering*) su ogni nodo. Il traguardo raggiunto da questo tool, è quello di ricreare la stessa dinamicità di una rete ad hoc reale su una rete di nodi fissi, in modo tale che il testing e l'analisi delle MANET possono essere facilmente realizzati all'interno di un laboratorio.

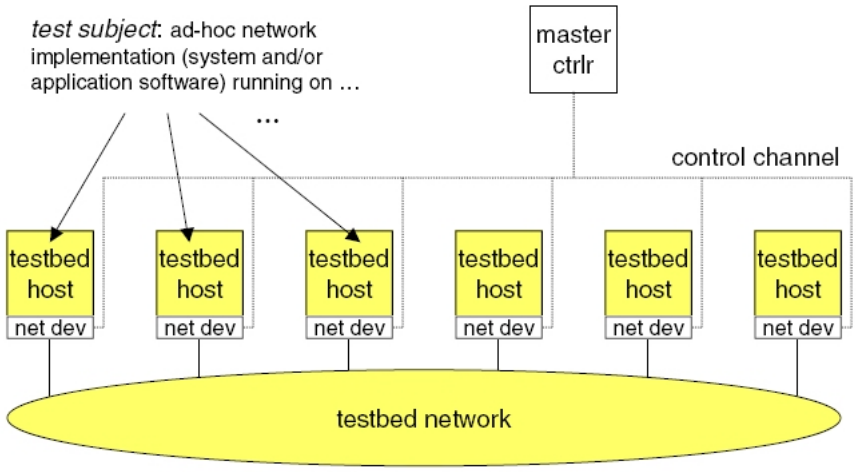

Figura C.1: Struttura del testbed realizzato da MobiEmu.

L'emulazione fornita dal software è detta *scenario-driven* in quanto MobiEmu realizza il movimento dei nodi basandosi su di un file di input, all'interno del quale vengono elencate le posizioni e i movimenti in ogni istante di tutti i nodi della rete. Il tool fornisce anche un'interfaccia grafica, attraverso la quale l'utente può previsualizzare, controllare e supervisionare in tempo reale l'evoluzione della rete nel tempo.

Per testare l'implementazione di una rete ad hoc con MobiEmu, l'utente deve eseguire il software oggetto del test su ogni macchina del testbed realizzato e trattare tale macchina come se fosse in una reale rete ad hoc.

La Figura C.1 illustra l'architettura del testbed realizzato da MobiEmu. Ogni host (*testbed host*) è una macchina che emula un nodo mobile della rete ad hoc ed è connesso con ogni altro nodo mediante una rete dedicata (*testbed network*). Quest'ultima può essere una rete locale di qualunque tipo, sia Ethernet o 802.11 WLAN. Sebbene fisicamente la testbed network sia completamente connessa, MobiEmu forzerà, mediante il packet filtering, una topologia parzialmente connessa al livello data link. In questa figura oltre ai testbed host ed alla testbed network viene indicato un'ulteriore componente, il *master ctrlr*, che si incarica della sincronizzazione di tutti gli host della rete.

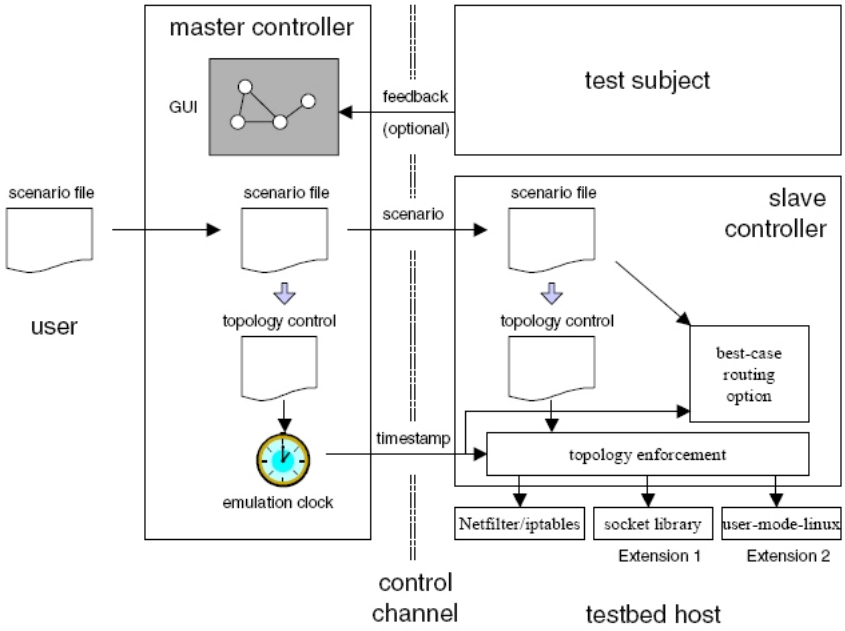

Figura C.2: Architettura del tool MobiEmu.

L'architettura su cui si basa MobiEmu è infatti di tipo master/slave. Il *master controller* viene eseguito su un host al di fuori della testbed network, mentre lo *slave controller* viene eseguito su ciascuna macchina testbed host. Il master controlla tutti gli slave e sincronizza le loro azioni: esso indica, mediante un canale di controllo chiamato *control channel*, quando cambia la connettività in topologia, forzando i cambiamenti sui nodi slave. Questo canale di controllo deve essere separato dalla testbed network al fine di evitare l'interferenza con le operazioni della rete ad hoc.

In Figura C.2 in basso a destra sono visibili due importanti estensioni di MobiEmu che permettono il suo impiego su testbed di diverso tipo. La *socket-library extension* che permette di eseguire il filtraggio dei pacchetti non direttamente sullo strato data link ma su file, e la *User-Mode-Linux extension*, che permette di eseguire MobiEmu anche su macchine virtuali create appunto con UML[1].

C.2 Master controller

Il controllore master è il coordinatore dell'intero testbed. Esso gestisce l'emulazione e fornisce all'utente un'interfaccia grafica per supervisionare ed inizializzare il test.

Prima di iniziare l'emulazione è necessario caricare nel master lo scenario attraverso l'interfaccia grafica. Questo scenario è un'elenco di definizioni di locazioni e di movimenti di tutti i nodi della rete con relativi timestamp, al fine di determinare in ogni istante dove si trova ogni nodo della rete. Nella release utilizzata (versione 1.2) MobiEmu accetta due tipi di formati per questo scenario: il nativo di NS2 ed un formato semplificato. Il primo è lo stesso formato di scenario mobile usato nel simulatore di rete NS2 con l'estensione wireless CMU (vedi [CMU]). Allo stesso modo MobiEmu utilizza un qualsiasi scenario generato dal tool `setdest`, contenuto nell'estensione CMU, per guidare la sua emulazione. Lo scenario semplificato invece fornisce lo stesso funzionamento, senza però la specifica sintassi di NS2, al fine di essere più svincolato da quest'ultima piattaforma.

Il comando `setdest` di CMU fornisce l'impostazione di diversi parametri riguardo la creazione dello scenario, come il numero di nodi, la velocità massima dei nodi in metri/secondo, la durata dell'emulazione, l'area espressa in metri all'interno della quale si muovono i nodi ed infine la durata delle pause in cui i nodi rimangono immobili. Per tutti questi motivi e per la facilità di generazione di questo tipo di scenario si è scelto di utilizzare questo formato.

Esempi di linee contenute all'interno dello scenario generato da `setdest` sono le seguenti:

```
$ns_ at 10.516961623942 "$god_ set-dist 11 14 2" $ns_ at 79.611442293090 "$
node_(6) setdest 207.394884908138 151.048233364251 0.140749383084"
```

La prima linea indica che all'istante, misurato in secondi, $10.51s$ dall'inizio dell'emulazione, il nodo 11 dista dal nodo 14, 2 hop. Essendo 2 hop i due nodi sono fuori range reciproco ed i messaggi inviati in unicast dall'uno non vengono ricevuti dall'altro in quanto filtrati da MobiEmu. La seconda linea invece indica che, all'istante $79.6s$, il nodo 6 cambierà la propria direzione di moto seguendo la nuova definita dalle coordinate $X = 207.39$

[1] User Mode Linux è un kernel Linux opportunamente modificato che permette di essere eseguito su di un'altro kernel Linux. Il kernel UML gira pertanto sulla macchina host in *User Space* come un normale altro programma ed accede, tramite il kernel host, all'hardware della macchina. Normalmente infatti il kernel Linux comunica con la macchina ed i programmi in User Space comunicano con il kernel, allo stesso modo il kernel UML non comunica direttamente con la macchina ma con il kernel host. I programmi possono essere eseguiti sul kernel UML come se fossero eseguiti su un normale kernel Linux. Grazie ad UML è possibile emulare diverse macchine Linux su una stessa macchina host, aventi tutte la stessa configurazione hardware della macchina ospitante. (vedi [UML])

ed $Y = 151.04$, che corrispondono alle coordinate sull'area di emulazione, ad una velocità di $0.14m/s$ e la manterrà fino a nuova segnalazione dettata dallo scenario.

Il primo passo dell'emulazione è pertanto l'invio dello scenario da parte del master a tutti gli slave. In seguito, sia il master che gli slave genereranno i cambiamenti di topologia in maniera indipendente.

A questo punto, una volta caricato ed inviato in broadcast lo scenario, può iniziare l'emulazione. Il master inizializza un clock generale, chiamato *master clock*, all'istante zero. Quando il clock raggiunge un timestamp associato con una regola di controllo della topologia, il master ordinerà gli slave coinvolti dal cambiamento di eseguire questa regola. Questo viene fatto inviando in broadcast il timestamp e, mediante il matching da parte degli slave sullo scenario in loro possesso, avviene l'esecuzione della regola su tutti gli slave. In pratica gli slave non devono mantenere un proprio clock, ma eseguono le regole contenute nello scenario quando gli viene loro ordinato dal master controller.

Il progresso del clock di emulazione avviene a tempo reale, però la GUI permette all'utente di aumentare o diminuire la velocità di esecuzione ed anche di mettere in pausa l'emulazione per poi ripristinarla. Tutto questo però influirà esclusivamente sul clock dell'emulazione e non sull'oggetto del testing che continuerà a lavorare alla stessa velocità. Mettendo in pausa ad esempio l'emulazione si avrà il solo effetto di *"ibernare"* lo stato attuale della rete, mentre l'applicazione da testare continuerà ad eseguire normalmente. Questa caratteristica può rivelarsi molto utile per il debugging delle reti ad hoc.

Una volta caricato nel master lo scenario da utilizzare per l'emulazione, questo verrà mostrato sull'interfaccia in maniera grafica (vedi Figura C.3).

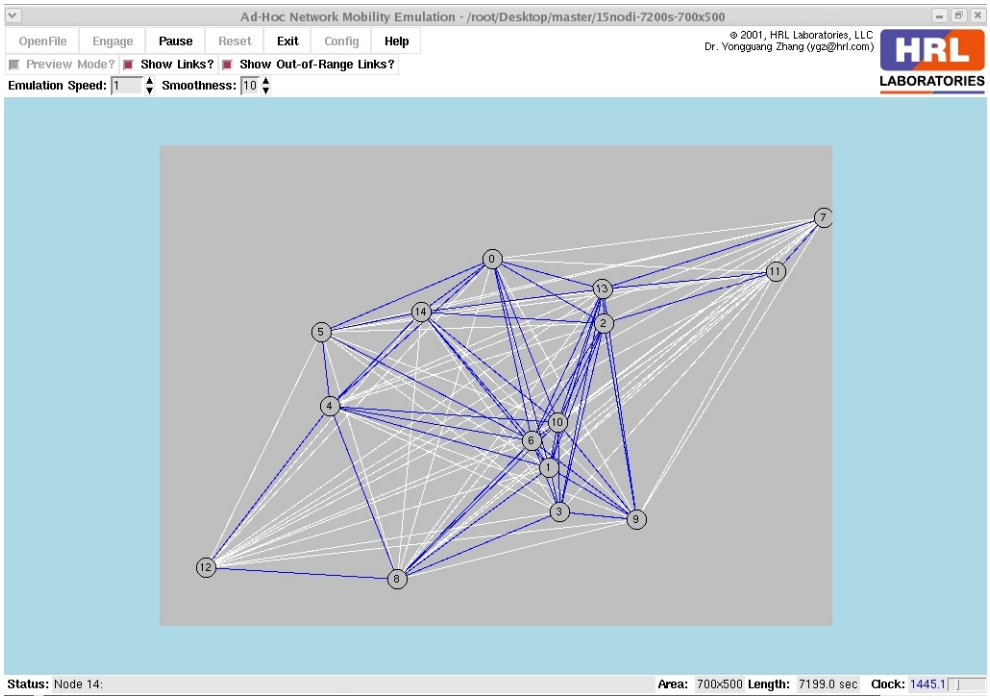

Figura C.3: Screen-shot dell'interfaccia del master controller.

Ogni nodo viene indicato mediante un cerchio con un'identificatore all'interno, il quale corrisponde all'identificatore presente nelle regole dello scenario. Quando due nodi sono in visibilità reciproca (come definito dallo scenario), questo viene indicato sul grafo mediante una linea blu che li interconnette, altrimenti verrà visualizzata una linea bianca. Pertanto l'insieme delle linee bianche e blue realizza un grafo completamente connesso mentre il solo insieme delle linee blue, che rappresenta la topologia vera e propria, realizza un grafo parzialmente connesso. Durante l'emulazione i nodi escono ed entrano nel range di ogni altro in base allo scenario ed i colori delle linee cambiano a seconda della distanza tra i nodi.

La GUI del master permette inoltre di decidere quali link visualizzare (in figura vengono mostrati entrambi), ed anche la possibilità di visualizzare prima dell'emulazione, l'evolversi della topologia per tutta la durata dell'emulazione. Il master infine permette anche la possibilità di colorare i nodi, cosa utile quando si hanno ruoli diversi tra gli host.

C.3 Slave controller

Nell'architettura di MobiEmu gli slave controller sono i responsabili della realizzazione della topologia. Se infatti un nodo A, in accordo allo scenario, è fuori dal range di un nodo B, questo deve filtrare tutti i pacchetti provenienti da B e viceversa. Questo deve essere eseguito al di sotto dello strato di rete in modo tale da essere trasparente all'applicazione oggetto del testing.

La tecnica utilizzata per selezionare i pacchetti è la stessa usata da molti firewall Unix, ed è chiamata packet filtering. Dal momento che MobiEmu è stato implementato in Linux, per eseguire il packet filtering vengono utilizzate le `Netfilter/iptables` (vedi [Rus]). Queste sono situate nello stack IP del kernel e nei drivers dei dispositivi di rete e manipolano i pacchetti in arrivo o in uscita all'host secondo regole predefinite. Tali regole però possono essere settate o cambiate in qualunque momento mediante comandi di interfaccia, ed è proprio quello che utilizzano gli slave per filtrare i pacchetti.

Ad esempio se un nodo A è fuori dal range di un nodo B e l'indirizzo MAC del nodo A è $01:23:45:67:89:0a$, MobiEmu setterà sul nodo B la seguente regola:

```
iptables -t mangle -A PREROUTING -m mac \
         --mac-source 01:23:45:67:89:0a -j DROP
```

Questa regola elimina (DROP) tutti i pacchetti provenienti dall'indirizzo MAC del nodo A. In seguito, se il nodo A si muove entro il range del nodo B, quest'ultimo rimuove semplicemente la regola e ripristina la comunicazione da A a B. Tipicamente la stessa cosa accade anche sul nodo A per filtrare i pacchetti provenienti da B. L'uso del filtraggio degli indirizzi MAC implica l'unico requisito di avere una singola LAN, cioè un'unica sottorete, in quanto se un pacchetto dovesse attraversare i limiti di una sottorete, il gateway cambierebbe l'indirizzo MAC della sorgente. Un'alternativa per poter eseguire MobiEmu anche su sottoreti diverse è quella di eseguire il packet filtering sugli indirizzi IP al posto degli indirizzi MAC.

C.4 Control channel

Attraverso il canale di controllo avvengono le interazioni master/slave. Questo canale viene implementato come un indirizzo multicast UDP. Per inviare messaggi il master e gli slave inviano semplicemente messaggi UDP ben formattati all'indirizzo multicast. Per ricevere messaggi invece essi rimangono semplicemente in ascolto sul tale indirizzo per intercettare i messaggi ad essi destinati.

Il canale di controllo implementa le seguenti interazioni:

- **Scenario**: Prima di ogni emulazione, il master controller notifica a tutti gli slave lo scenario selezionato per l'esecuzione.
- **Ping**: Il master controller verifica lo stato di tutti gli slave, se sono attivi e pronti a eseguire, mediante l'invio di tale messaggio in broadcast.
- **Ping reply**: Messaggio di risposta ad uno di ping. Se uno slave ha terminato di computare lo scenario ed è pronto per l'emulazione, invia tale messaggio contenente inoltre il proprio indirizzo IP e MAC.
- **Timestamp**: Il master controller invia in multicast il valore del clock di emulazione per innescare l'azione sugli slave.
- **Feedback**: Questo messaggio serve ad uno slave per inviare informazioni al master, come ad esempio per le opzioni di visualizzazione.

I primi tre tipi di messaggi sono necessari per la procedura di *handshake* tra il master e gli slave per la configurazione dell'emulazione. Un'emulazione diventa pronta per essere eseguita solo quando il master ha ricevuto risposta positiva (Ping reply) da tutti gli slave. Se viene perso anche un solo messaggio la procedura deve essere ripetuta. Quando tutti gli slave sono pronti, l'utente può far partire l'emulazione.

Alla partenza dell'emulazione il canale di controllo porta solamente messaggi di tipo Timestamp e Feedback. I messaggi Timestamp sono inviati solo nel caso di cambiamento di topologia e, per aumentare l'affidabilità, viene ripetuto 3 volte ad intervalli di $10ms$. I Timestamp duplicati vengono semplicemente ignorati dagli slave.

C.5 Best-case routing

Una caratteristica opzionale di MobiEmu è il suo supporto al testing di applicazioni di rete su più livelli. Al fine di poter eseguire una comunicazione unicast tra nodi distanti più di 1 hop, nel caso in cui esista una cammino tra questi due nodi, è necessario eseguire uno dei tanti protocolli di routing unicast visti nel Capitolo 2. Questa opzione di MobiEmu realizza proprio questa funzione, implementando al proprio interno un protocollo di routing necessario in questi casi. Non occorre pertanto procurarsi il protocollo ed eseguirlo, ma è sufficiente utilizzare quello implementato in MobiEmu. Questa caratteristica è stata ampiamente utilizzata per il testing del sistema in esame.

MobiEmu implementa il cosiddetto protocollo di *routing best-case* con funzionalità in ambito ad hoc senza eseguire un vero e proprio protocollo di routing. Tale algoritmo lavora in questo modo.

Quando uno slave riceve un messaggio Timestamp esso computa il cammino più corto, mediante l'algoritmo di Dijkstra[2], verso tutti i nodi della corrente topologia. Per ogni nodo che non è un diretto vicino, esso aggiunge uno specifico cammino nelle proprie routing table mediante il comando `route add`. Mediante le entry delle routing table ora i pacchetti vengono inoltrati al corretto vicino, anche nel caso di due nodi non in visibilità ma uniti da un cammino. Quando il nodo destinazione poi rientrerà nel range di comunicazione, il nodo mittente cancellerà la relativa entry dalla routing table. Nel caso in cui invece il cammino più breve subisce variazioni, cambiando il giusto next-hop l'entry della routing table relativa verrà modificata.

Dal momento che MobiEmu calcola il miglior cammino per ogni nodo della topologia, questo risulterà essere anche il miglior caso per il routing all'interno di quello scenario e nessun protocollo deterministico potrà ottenere un cammino migliore. Anche se questo caso non sarà implementabile nella vita reale, questo rappresenta una buona soluzione ai fini dello studio della rete.

C.6 Configurazione del testbed

Il primo passo nella preparazione del testbed di emulazione è stato la creazione, mediante il tool di NS2 `setdest`, di uno scenario di soccorso avente 30 nodi e con le caratteristiche più opportune al caso in esame. Sono stati forniti pertanto a riga di comando i seguenti parametri:

```
./setdest -v 1 -n 30 -p 5 -M 2 -t 18000 -x 1000 -y 800 > scenario_30nodi
```

Dopo essersi posizionati nella cartella contenente il comando `setdest` si è eseguita l'istruzione sopra indicata, la quale genera uno scenario con le caratteristiche passate come parametri e lo memorizza nel file `scenario_30nodi`. Quest'ultimo sarà il file scenario di input per il master e per tutti gli slave di MobiEmu. I parametri passati a riga di comando significano nell'ordine: la versione del tool `setdest`[3] (versione 1), il numero di nodi (30), la pausa ($5s$), la velocità massima raggiungibile ($2m/s$), la durata dell'emulazione ($18000s$, cioè $5h$) ed infine la coordinata x ($1000m$) ed y ($800m$).

Una nota particolare va fatta riguardo il tipo di movimento che assumono i nodi mediante lo scenario generato. Il movimento utilizzato da `setdest` è di tipo *Random Waypoint*, in cui i nodi assumono cambiamenti di direzione scelti in modo casuale. Ad essere rigorosi

[2]Edsger W. Dijkstra propose tale algoritmo per la determinazione di un cammino minimo all'interno di un grafo nel 1959. Mediante questa soluzione vengono assegnati dei pesi agli archi che connettono i nodi ed il cammino a minor peso tra due nodi rappresenta anche il cammino minimo tra di essi. Per una spiegazione più approfondita vedere [V02]

[3]In quanto ne esiste una seconda versione modificata dall'Università del Michigan e che introduce la possibilità di scegliere il tipo di movimento dei nodi, il tipo di velocità ed il tipo di pause. Questa seconda versione non è compatibile però con il tool MobiEmu.

questo tipo di movimento può risultare, come detto, differente da una situazione reale in cui ad esempio gli ufficiali di più alto grado rimangono piuttosto vicini l'uno all'altro, ma per le finalità del testing è più che soddisfacente.

Una volta generato lo scenario è stato necessario caricarlo sia sul nodo che funge da master controller sia su tutti i nodi slave. A questo fine non si è dovuto caricare fisicamente il file su ogni macchina in quanto tutte le macchine utilizzate per la prova caricavano la stessa HOME, in cui era presente lo stesso file di scenario.

Al fine di velocizzare l'inizializzazione del test, si è scelto di utilizzare un'ambiente fornito esclusivamente da Linux, chiamato `ssh`. Mediante `ssh` infatti è possibile accedere da una macchina host Linux ad altre n macchine Linux mediante un semplice comando e lanciare applicazioni e comandi dall'host come se si stesse seduti fisicamente davanti la macchina acceduta via remoto. Una volta lanciata un'applicazione ad esempio è possibile visualizzare sulla macchina host l'output di tale applicazione che in realtà sta eseguendo su una macchina diversa.

Lanciando gli slave di MobiEmu via `ssh` è stato sufficiente avere un'unica cartella (`/opt/slave`) contenente i file degli slave, dei quali ne sono state poi eseguite 30 istanze su 30 diverse macchine. Il comando attraverso il quale è stata lanciata l'emulazione degli slave è il seguente:

```
ssh myUserID@192.168.70.* "cd /opt/slave; sudo /opt/slave/emulc-route -rou-
te X eth0 230.0.0.1/3685/1/eth0"
```

Si lancia `ssh` mediante la propria UserID e l'indirizzo IP della macchina alla quale ci si vuole *loggare* (i numeri rappresentano la sottorete e l'asterisco indica il numero della macchina all'interno della sottorete). Tra doppi apici viene inserito invece il comando che si vuole eseguire in remoto su tale macchina. Questo prima si posiziona nella directory in cui si trova la cartella contenente i file dello slave (in questo caso `/opt/slave`) e quindi lancia il comando opportuno (in questo caso `emulc-route`).

Lo slave di MobiEmu fornisce diversi comandi per lanciare l'emulazione. In questo caso viene scelto `emulc-route` in quanto mediante esso viene forzato l'uso dell'algoritmo di routing *shortest-path*. Come si vede dalla linea di comando viene utilizzato anche il comando `sudo`. Dato che gli slave vanno a modificare le `iptables` mediante delle regole di interfaccia, cosa possibile solamente se si dispone dei permessi necessari, cioè di utente *root*, l'uso di `sudo` è stato necessario al fine di elevare i privilegi e permettere il filtering dei pacchetti.

Gli altri parametri passati al comando `emulc-route` sono rispettivamente, l'indicazione dell'utilizzo del routing (`-route`), il numero di nodo (`X`, a cui sostituire un numero da 0 a 29), l'interfaccia da utilizzare per l'emulazione (`eth0`), l'indirizzo multicast che realizza il control channel (`230.0.0.1`), la porta (`3685`), il TTL (`1` hop) ed infine l'interfaccia per il control channel (la stessa `eth0`).

A questo punto tutti gli slave sono in ascolto sul canale multicast creato, in attesa dell'inizio dell'emulazione. Sulla macchina host utilizzata per lanciare i comandi in remoto si lancia il master controller di MobiEmu. Dall'interfaccia grafica si imposta dal menu `Configure` l'indirizzo scelto per il canale di controllo (`230.0.0.1`), la porta (`3685`), il TTL

(1 hop) e l'interfaccia (`eth0`). A questo punto si carica lo scenario e premendo il tasto `Engage`, si inizia la procedura di handshake tra il master e gli slave. Il master inizia ad inviare messaggi di Ping ed attende le risposte da parte di ogni slave. Quando tutti i nodi avranno risposto significa che è possibile dare inizio all'emulazione e si illumina il tasto `Start`.

Prima di premere il tasto di inizio è necessario però lanciare l'applicazione da testare. Sulla HOME sono state create 30 cartelle ognuna contenente un'istanza dell'applicazione da eseguire ed ognuna lanciata su una macchina diversa. I risultati dei vari parametri analizzati saranno raccolti all'interno di ogni cartella. Il comando eseguito per il lancio dell'applicazione avviene ancora via `ssh` ed è il seguente:

```
ssh myUserID@192.168.70.* "cd $HOME/folder_name; java -Dagape.base=testing/
bin classpath=testing/bin Test.Main"
```

Ci si pone pertanto all'interno della cartella contenente l'applicazione e la si lancia fornendo gli opportuni parametri di configurazione. Il nome della classe di test è `Main` ed è contenuta all'interno del package `Test`.

Ora che sono stati lanciati gli applicativi da testare è possibile iniziare l'emulazione premendo il tasto `Start` sulla GUI del master controller.

Bibliografia

[BCM03] D.Bottazzi, A.Corradi, R.Montanari:*"AGAPE: a Location Group Membership Middleware Pervasive Computing Environments"*, 8th IEEE Int. Symposium on Computers and Communications (ISCC03), 2003.

[Y03] S.Yazbeck:*"IEEE 802.11 and Bluetooth: An Architectural Overview"*, CRC Press LLC, 2003.

[T03] A.S.Tanenbaum:*"Computer Networks 4^{th} Edition"*, Prentice Hall, 2003.

[C03] M.Conti:*"Body, Personal and Local Ad Hoc Wireless Network"*, CRC Press LLC, 2003.

[Z96] T.G.Zimmerman:*"Personal Area Networks: Near-field Intrabody Communication"*, IBM Systems Journal, 1996.

[MBS03] A.Mukherjee, S.Bandyopadhyay, D.Saha:*"Location Management and Routing in Mobile Wireless Networks"*, Artech House, 2003.

[LRG97] Lin, R., M.Gerla:*"Adaptive Clustering for Mobile Wireless Networks"*, IEEE journal on selected Areas in Communications, 1997.

[BB03] R.Beraldi, R.Baldoni:*"Unicast Routing Techniques for Mobile Ad Hoc Networks"*, CRC Press LLC, 2003.

[PB94] C.E.Perkins, P.Bhagwat: *"Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers"*, Computer Communications Review, 1994.

[C01] T.Clausen et al.: *"Optimized Link State Routing Protocol"*, IETF MANET Working Group Internet Draft, 2001.

[GHP01] M.Gerla, X.Hong, G.Pei: *"Fisheye State Routing Protocol (FSR) for Ad Hoc Networks"*, IETF MANET Working Group Internet Draft, 2001.

[KTC00] D.-K.Kim, C.-K.Toh, Y.-H.Choi: *"RODA: A New Dynamic Routing Protocol Using Dual Paths to Support Asymmetric Links in Mobile Ad Hoc Networks"*, Proceedings of IEEE IC3N, 2000.

[PR99] C.E.Perkins, E.M.Royer: *"Ad Hoc On Demand Distance Vector Routing"*, Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, 1999.

[PRD02] C.E.Perkins, E.M.Royer, S.R.Das: *"Ad Hoc On-Demand Distance Vector (AODV) Routing"*, IETF MANET working group, 2002.

[PC97] V.D.Park, M.S.Corson: *"A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks"*, Proceedings of IEEE INFOCOM, 1997.

[T97] C.K.Toh: *"Associativity based routing for ad hoc mobile networks"*, Wireless Personal Communications, 1997.

[PH99] M.R.Pearlman, Z.J.Haas: *"Determining the optimal configuration for the zone routing protocol"*, IEEE Journal on Selected Areas in Communications, 1999.

[HPS01a] Z J.Haas, M.R.Pearlman, P.Samar: *"The Interzone Routing Protocol (IERP) for Ad Hoc Networks"*, IETF MANET working group, 2001.

[HPS01b] Z J.Haas, M.R.Pearlman, P.Samar: *"The Intrazone Routing Protocol (IARP) for Ad Hoc Network"*, IETF MANET working group, 2001.

[NLB00] N.Nikaein, H.Labiod, C.Bonnet: *"Distributed Dynamic Routing Algorithm for Mobile Ad Hoc Networks"*, Proceedings of ACM MobiHoc, 2000.

[KV00] Y.-B.Ko, N.H.Vaidya: *"Location-Aided Routing (LAR) in mobile ad hoc networks"*, ACM/Baltzer Wireless Networks (WINET) Journal, 2000.

[CM99] S.Corson, J.Macker: *"Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations"*, ARFC2501, 1999.

[BMJHJ98] J.Broch, D.A.Maltz, D.B.Johnson, Y.-C.Hu, J.Jetcheva: *"A Performance Comparison Hop Wireless Ad Hoc Network Routing Protocols"*, Proceedings of the Fourth Annual International Conference on Mobile Computing and Networking, 1998.

[LG99] S.-J.Lee, M. Gerla: *"A Simulation Study of Table-Driven and On-Demand Routing Protocols for Mobile Ad Hoc Networks"*, IEEE Network, 1999.

[BLSG00] S.H.Bae, S.-J.Lee, W.Su, M.Gerla: *"The Design, Implementation, and Performance Evaluation of the On-demand Multicast Routing Protocol in Multihop Wireless Networks"*, IEEE Network, 2000.

[CW03] X.Chen, J.Wu: *"Multicasting Techniques in Mobile Ad Hoc Networks"*, CRC Press LLC, 2003.

[RP99] E.M.Royer, C.E. Perkins: *"Multicast Operation of the Ad Hoc On-Demand Distance Vector Routing Protocol"*, Proc. of MobiCom, 1999.

[KC01] T.Kunz, E.Cheng: *"Multicasting in Ad Hoc Networks: Comparing AODV and ODMRP"*, Workshop on Ad Hoc Communications, 2001.

[PP05] M.Puzar, T.Plagemann: *"NEMAN: A Network Emulator for Mobile Ad-Hoc Networks"*, Oslo University Technical Report#321, 2005.

[R01] A.Rowstron: *"Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems"*, ACM Press, 2001.

[OV01] K.Obraczka, K.Viswanath: *"Flooding for Reliable Multicast in Multi-Hop Ad Hoc Networks"*, Kluwer Academic Publishers, 2001.

[NS2] The Network Simulator 2 Home Page, `http://www.isi.edu/nsnam/ns/`.

[MB] The MobiEmu Project Home Page, `http://mobiemu.sourceforge.net/`.

[JEM] The Java Emulation Platform Home Page, `http://jemu.emuunlim.com/`.

[NN] The NIST Net Home Page, `http://snad.ncsl.nist.gov/nistnet/`.

[ONE] The Ohio Network Emulator Home Page, `http://masaka.cs.ohiou.edu/one/`.

[UML] The User Mode Linux Kernel Home Page, `http://user-mode-linux.sourceforge.net/`.

[CMU] The CMU Monarch Extension to the NS-2 Simulator Home Page, `http://monarch.cs.cmu.edu/cmu-ns.html`.

[Rus] R.Russell:*"Linux 2.4 packet filtering HOWTO"*, `http://netfilter.samba.org/documentation/`.

[V02] S.Vigna:*"L'algoritmo di Dijkstra"*, `http://ioi.dsi.unimi.it/dijkstra.pdf`.

Ringraziamenti

In queste poche righe vorrei ricordare non soltanto coloro che mi sono stati d'aiuto per la realizzazione di questa tesi, ma anche, e soprattutto, tutte le persone che mi hanno accompagnato durante questa splendida avventura, perchè è anche grazie a loro se oggi mi ritrovo alla conclusione di questo lungo viaggio.

In primo luogo vorrei ringraziare i compagni di questi sette bellissimi, ma duri anni, trascorsi in via Belvedere...il mio primo compagno di stanza Ste, da sempre un grandissimo vero amico, il mio secondo compagno di stanza Massi (Gianni), con cui ho passato le più belle serate bolognesi, il vecchio Cris (Graziano), che mi ha regalato innumerevoli magie nel corso di questi anni ed infine Massi (Umberto e/o Cagh...), per le fantastiche sfide alla Play e per gli infiniti momenti trascorsi insieme.

Se ho raggiunto questo traguardo è anche merito vostro che mi avete sostenuto nei momenti più bui ed avete condiviso con me le difficoltà di questo percorso.

Un ringraziamento particolare va chiaramente ai miei genitori, per avermi permesso di intraprendere questa strada ed aver creduto in me fino in fondo con sostegno e spirito di sacrificio. Grazie anche a Fede per essere mio fratello.

Vorrei ringraziare il Prof. Antonio Corradi per avermi dato la possibilità di realizzare questo lavoro di tesi e per la costante disponibilità dimostrata nei momenti di bisogno. Un grazie molto sentito, ma direi anche qualcosa di più, va all'Ing. Dario Bottazzi, che più di ogni altro mi ha seguito, con costanza quasi giornaliera, in questi sei mesi ed il quale si è dimostrato essere, più che un correlatore, un'ottimo amico.

Vorrei ringraziare anche tutti i ragazzi del LAB2 che grazie ai loro consigli sono riuscito a superare diversi ostacoli, ed in particolare Fabio, Luca, Ghedo e Andrea.

Per il loro supporto morale e per le gioie condivise da sempre che mi ricordano che non sarò mai solo: Robi, Renfield, Dani, Abi, Bordo, Sbro, Gabo, Giova, Baldu e Bede. Grazie di essere miei amici.

Infine tutti i ragazzi che il primo anno hanno iniziato il corso con me, ma che per vari motivi hanno cambiato tutti, o quasi, indirizzo: l'Ing. Luca, Fabio, l'Ing. Massi, l'Ing. Enri, l'Ing. Enrica e l'Ing. Robi. Esatto...ero praticamente rimasto solo io.

Potrei anche continuare, ma volevo rientrare in un'unica pagina, così, dato che l'ho già superata, termino quì. Non me ne voglia chi si sente escluso da questo elenco, ma ormai sono stanco e potrei dimenticare qualcuno...così...anche a te...grazie, grazie, grazie!!!

...ed ora si salpa per un nuovo, lunghissimo viaggio...