UNIVERSITÀ DEGLI STUDI DI MILANO
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Scienze dell'Informazione
Dottorato di Ricerca in Informatica - X ciclo

PRIMITIVE PER LA COMUNICAZIONE DI GRUPPO AFFIDABILE IN SISTEMI DISTRIBUITI CON STAZIONI MOBILI

Supervisore Esterno: Prof. Mario Gerla

Supervisori Interni: Prof. Francesco Tisato
Prof. Gian Paolo Rossi

Tesi di Dottorato di:
ELENA PAGANI

ai miei genitori . . .

Prefazione

La capacità tecnologica di realizzare reti comprendenti stazioni mobili introduce numerosi problemi originali riguardo alla progettazione di protocolli e applicazioni distribuite. Lo sviluppo di Internet e la diffusione dei computer portatili avevano già consentito agli utenti di lavorare in differenti locazioni, provocando perciò la migrazione nel tempo dei loro dati, ed accedere a dati ed applicazioni remote. I recenti progressi nella tecnologia dei computer portatili e della comunicazione cellulare rendono ora possibile l'emergente paradigma della "*computazione mobile*". I computer portatili hanno attualmente caratteristiche comparabili con quelle delle tradizionali workstation; ciò rende desiderabile la possibilità per gli utenti di lavorare sui loro portatili come farebbero sulla loro più tradizionale stazione di lavoro. La tecnologia della comunicazione wireless consente ai computer portatili di comunicare mantenendo la connessione alla rete anche durante gli spostamenti.

Le caratteristiche delle stazioni mobili e delle reti wireless sono tali da introdurre requisiti per i servizi di supporto al lavoro 'nomade' che sono drasticamente differenti da quelli normalmente considerati per le tradizionali reti fisse. Tali requisiti rendono i protocolli sviluppati per sistemi fissi inadeguati all'uso in reti mobili; si rende perciò necessario individuare una nuova architettura di sistema e progettare i protocolli originali che la costituiscano e che forniscano i servizi necessari per il supporto al lavoro mobile.

Questa tesi concentra l'attenzione su problemi relativi alla gestione di interazioni del tipo "*uno-a-molti*" e "*molti-a-molti*" in gruppi di stazioni mobili cooperanti; in particolare, si prendono in considerazione i problemi di *instradamento multicast*, ovvero di instradamento di messaggi indirizzati a più destinazioni, e di *Reliable Broadcast*, ovvero della consegna affidabile di messaggi a tutte le stazioni del sistema. La soluzione di questi due problemi è fondamentale per la realizzazione di servizi di *comunicazione di gruppo*, poichè essi costituiscono i 'mattoni' necessari alla costruzione di soluzioni per problemi più complessi. Per essi vengono proposti due protocolli originali, di cui si dimostra la correttezza e si fornisce una valutazione delle prestazioni ottenuta mediante simulazioni. I protocolli sono progettati per operare in reti mobili multi-hop, vale a dire reti in cui sono connesse unicamente stazioni mobili.

Organizzazione della tesi

La tesi è organizzata come segue:

- nel Capitolo 1 si presenta una panoramica delle originali problematiche di ricerca poste dalle reti con stazioni mobili. Si descrivono le caratteristiche generali dei sistemi wireless e si introduce la notazione adottata nel resto del lavoro. Si analizza l'impatto delle caratteristiche dei sistemi mobili sulla realizzazione di un'architettura di rete e dei relativi protocolli;

- nel Capitolo 2 si introducono le definizioni dei fondamentali problemi di comunicazione affidabile che i protocolli proposti nella tesi si propongono di risolvere, e si descrive il modello di sistema in cui tali protocolli devono operare. Si dà una breve panoramica dei lavori apparsi in letteratura in cui sono proposte soluzioni a problemi attinenti a quelli considerati;

- nel Capitolo 3 si esaminano in dettaglio le caratteristiche dei protocolli di basso livello dell'architettura nel modello di sistema considerato nella tesi, e si descrive il pacchetto di simulazione impiegato per la valutazione delle prestazioni dei protocolli proposti;

- nel Capitolo 4 si descrive il protocollo per risolvere il problema del Reliable Broadcast nelle reti wireless considerate. Si dimostra la correttezza del protocollo e se ne analizzano la complessità e le prestazioni;

- nel Capitolo 5 si descrive l'algoritmo per la costruzione di una struttura di instradamento multicast, se ne dimostra la correttezza e se ne valutano le prestazioni.

La tesi si conclude con alcune considerazioni finali, riguardanti anche i possibili futuri sviluppi di ricerca, e con le Appendici riportanti la specifica dei protocolli proposti in forma di automi a stati finiti o di pseudo-codice.

Ringraziamenti

Al termine di questo lavoro desidero ricordare tutti coloro che mi hanno fornito il loro aiuto e il loro sostegno.

Un ringraziamento particolare al Prof. Gian Paolo Rossi, per la sua dispnibilità e il suo fondamentale contributo per tutta la durata del corso di dottorato. Desidero inoltre ringraziare il Prof. Mario Gerla e il Prof. Francesco Tisato per la loro collaborazione e i loro preziosi suggerimenti. Un grazie ai miei colleghi di dottorato per la loro amicizia e il loro prezioso sostegno durante questi anni. Infine, ai tesisti del Laboratorio di Telematica, per la loro simpatia e la loro comprensione nel periodo cruciale della stesura della tesi.

Indice

Capitolo 1

I sistemi wireless

1.1 Introduzione

La capacità tecnologica di realizzare reti comprendenti stazioni mobili introduce numerosi problemi originali relativi alla progettazione di protocolli e applicazioni distribuite. Lo sviluppo di Internet e la diffusione dei computer portatili consentivano già agli utenti di lavorare in differenti locazioni, provocando perciò la migrazione nel tempo dei loro dati, e di accedere a dati ed applicazioni remote. I recenti progressi nella tecnologia dei computer portatili e della comunicazione cellulare hanno ora fatto emergere il nuovo paradigma della "*computazione mobile*". La libertà degli utenti di spostarsi nel sistema mantenendo la connessione in rete senza alcun vincolo strutturale permette la costruzione di 'sistemi informativi mobili', destinati principalmente all'uso per applicazioni specializzate, quali ad esempio servizi di informazione, coordinamento di gruppi di lavoro, coordinamento di operazioni di soccorso in situazioni di emergenza o di operazioni su campi di battaglia mobili. La comparsa di questi sistemi rende perciò possibile la realizzazione di nuove classi di applicazioni, ma nel contempo pone numerosi problemi originali riguardanti il supporto alla mobilità. Ad esempio, i protocolli dovrebbero essere in grado di adattare la propria qualità di servizio in accordo alle risorse correntemente disponibili, di trattare radicali cambiamenti nello stato di connettività della rete come eventi normali piuttosto che come fallimenti, e di gestire accuratamente le ridotte risorse di banda disponibili. Poichè si prevede che questi sistemi saranno altamente eterogenei, è desiderabile che i servizi accessibili in rete siano indipendenti dalla locazione dei siti clienti o fornitori, dalle caratteristiche dei canali wireless localmente esistenti e dal tipo di piattaforma. Queste nuove esigenze fanno sì che i protocolli finora realizzati per le reti fisse siano inadeguati all'ambiente mobile; si rende perciò necessario lo sviluppo di una completa architettura di sistema e di un'intera famiglia di protocolli che la compongano, entrambi appositamente progettati per la mobilità.

In questo capitolo si descrivono le caratteristiche generali dei sistemi wireless, che devono essere considerate nella progettazione dell'architettura e dei protocolli mobili. Si illustrano le problematiche introdotte dalla connessione in rete di stazioni mobili. Si discutono i requisiti che i protocolli e l'architettura devono soddisfare, e si individuano le componenti che possono aiutare a soddisfare questi requisiti.

In questo capitolo si introduce inoltre la notazione adottata nel seguito del lavoro. Per la descrizione approfondita della particolare architettura considerata in questo lavoro e dei protocolli di basso livello che ne fanno parte, si rimanda al capitolo 3.

1.2 Struttura generale dei sistemi wireless

A differenza dei sistemi distribuiti con stazioni fisse (SF), i mezzi mobili (SM) sono caratterizzati dalla capacità di mantenere la connessione alla rete di comunicazione anche *durante gli spostamenti*. Questo è possibile grazie all'uso di tecnologie di rete *wireless*, quali ad esempio canali radio o a raggi infrarossi, o reti satellitari, che non impongono vincoli fisici alle stazioni.

Le SM si possono classificare in *dumb terminal* e *walkstation* [46]. Le prime sono macchine senza disco, con risorse di memoria e capacità di elaborazione molto limitate, come ad esempio i palmtop. Possono ricevere dalla rete wireless ma in generale non hanno la capacità di inviare messaggi. Le walkstation sono macchine con capacità di memoria e calcolo paragonabili a quelle di una normale workstation e possono sia ricevere che trasmettere messaggi sulla rete wireless. In questo lavoro considereremo solo queste ultime.

Le reti wireless sono classificabili in due grandi categorie: *single-hop* [46, 45, 8, 27, 49, 56] e *multi-hop* [26, 25, 28, 40, 22, 23, 24]. Le prime (Figura 1(a)) sono caratterizzate dalla presenza di un'infrastruttura fissa, che viene sfruttata per il supporto alla comunicazione tra le stazioni mobili: alcune delle stazioni fisse (le stazioni di supporto ai mobili, SSM) sono dotate di un'interfaccia alla rete wireless. Le SM in grado di ricevere i messaggi di una SSM sono all'interno della *cella* di tale SSM; il raggio di una cella è pari alla massima distanza alla quale un messaggio può essere ricevuto, e dipende dalla tecnologia wireless impiegata. Una SM può comunicare con il resto del sistema solo attraverso lo scambio di messaggi con la propria SSM; anche due SM residenti nella cella della medesima SSM non comunicano direttamente ma solo attraverso la SSM. La trasmissione di un messaggio da parte di una SSM all'interno della propria cella è *broadcast*. I messaggi scambiati tra SM residenti in celle differenti sono instradati sulla porzione fissa della rete.

Le caratteristiche delle reti single-hop, così come i problemi coinvolti nello sviluppo di algoritmi distribuiti e le tecniche usate per la gestione della mobilità in questi sistemi, si possono trovare discussi ad esempio in [10, 46, 8, 48, 9].

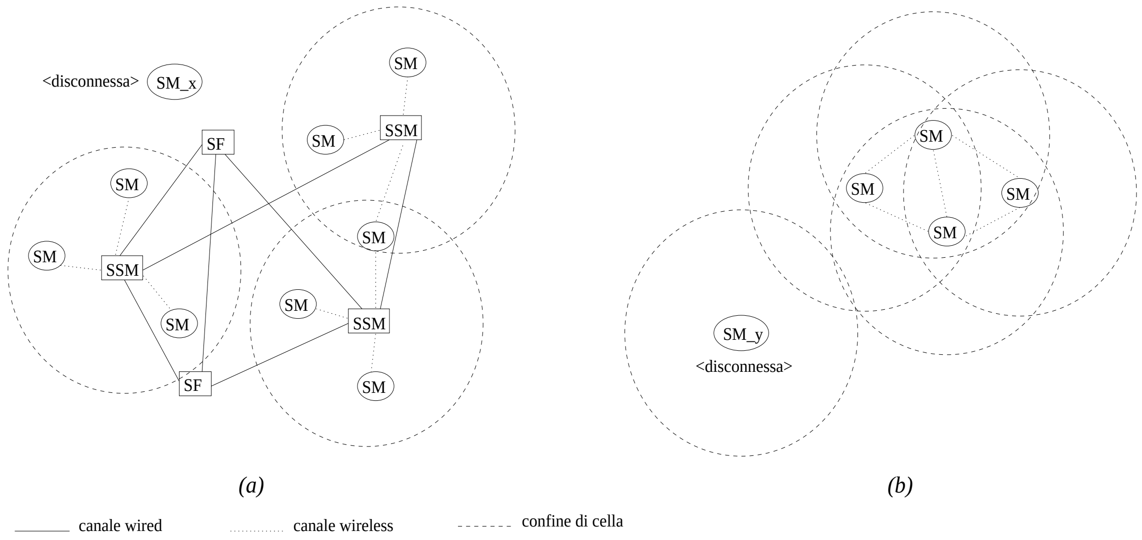


Figura 1: Esempi di reti (a) single-hop e (b) multi-hop.

Nell'ambito di questa tesi ci occuperemo soltanto delle reti multi-hop (Figura 1(b)), o reti mobili *ad-hoc*. Esse sono caratterizzate dall'assenza di qualsiasi infrastruttura fissa; ovvero tutte le macchine connesse sono SM. Per questa ragione i pacchetti possono compiere più di un hop sulla rete wireless per giungere alla propria destinazione, da cui la denominazione. Le reti multi-hop possono essere rapidamente installate in qualsiasi situazione, e sono perciò particolarmente adatte all'utilizzo in situazioni di emergenza (ad esempio, soccorso in caso di disastro), o quando non sia disponibile la connessione ad una rete fissa (ad esempio, campo di battaglia mobile).

Una SM può comunicare direttamente con tutte le SM che sentono il suo segnale, ovvero con le SM nella sua cella, perchè il canale di comunicazione è *broadcast*. Questo comporta dei problemi di interferenza tra trasmissioni contemporaneamente generate da SM nella stessa cella (come ad esempio le 4 stazioni connesse in Figura 1(b)), con conseguente *collisione* tra i messaggi.

In letteratura sono state studiate diverse tecniche per gestire il problema delle collisioni. Una delle possibilità è quella di costruire una struttura *gerarchica* sovrapponendo alla rete un insieme di *cluster*, possibilmente su più livelli. Il raggruppamento dei nodi in cluster, e possibilmente dei cluster di un livello in cluster di livello superiore (come in [62]) consente:

1. la riduzione della probabilità di collisione, se tra cluster differenti si usano codici ortogonali [40];

2. il riutilizzo di codici o frequenze tra cluster differenti, e perciò un più efficiente uso della banda disponibile;

3. una migliore scalabilità su un grande numero di SM, se si confina il trattamento dei comportamenti anomali al più basso livello della gerarchia che ne è interessato [62].

In questo lavoro si considerano reti con un'architettura gerarchica su un solo livello come quella ottenibile con gli algoritmi descritti in [40, 22]: le SM sono raggruppate in cluster, all'interno di ognuno dei quali una SM è eletta *clusterhead* (Figura 2). I clusterhead hanno il compito di coordinare la comunicazione fra SM nello stesso cluster e fra cluster differenti.
Gli algoritmi proposti in [40, 22] per costruire l'architettura gerarchica soddisfano la seguente proprietà.

Proprietà 1.2.1 [Clustering] *L'architettura a cluster è tale che:*

1. *Ogni SM all'interno di un cluster è ad un hop dal clusterhead e al più a due hop da ogni altra SM nello stesso cluster.*

2. *Due clusterhead non possono mai essere connessi direttamente.*

Due SM nello stesso cluster che non comunicano direttamente scambiano messaggi tramite il clusterhead.

In Figura 2 è mostrato un esempio di architettura gerarchica. Le stazioni che si trovano all'intersezione tra due o più cluster (ovvero, che sono a un hop da due o più clusterhead, come la SM p in Figura 2) sono dette *gateway*: i gateway si incaricano di gestire lo scambio di messaggi tra cluster. Un gateway *distribuito* è costituito da due SM che si trovano in cluster differenti (quali ad esempio p' e p'' in Figura 2) ma che possono scambiarsi messaggi. Due cluster possono anche essere indirettamente connessi via altri cluster (come i cluster C_A e C_D in Figura 2; oppure possono essere partizionati (come i cluster C_A e C_E), nel caso esista una partizione della rete.

Il principale vantaggio della struttura gerarchica scelta è quello di permettere il riutilizzo delle frequenze o dei codici tra cluster differenti, mentre all'interno di uno stesso cluster la gestione delle collisioni è affidata al protocollo di livello MAC.

1.2.1 Notazione

Poichè nelle reti wireless multi-hop ogni SM è anche router, nel seguito per indicare una SM useremo anche indifferentemente i termini di nodo e host.

Una rete multihop può essere rappresentata come un grafo $G = \langle N, E \rangle$ tale che l'insieme N dei nodi è costituito dalle macchine del sistema. Indicheremo con n la

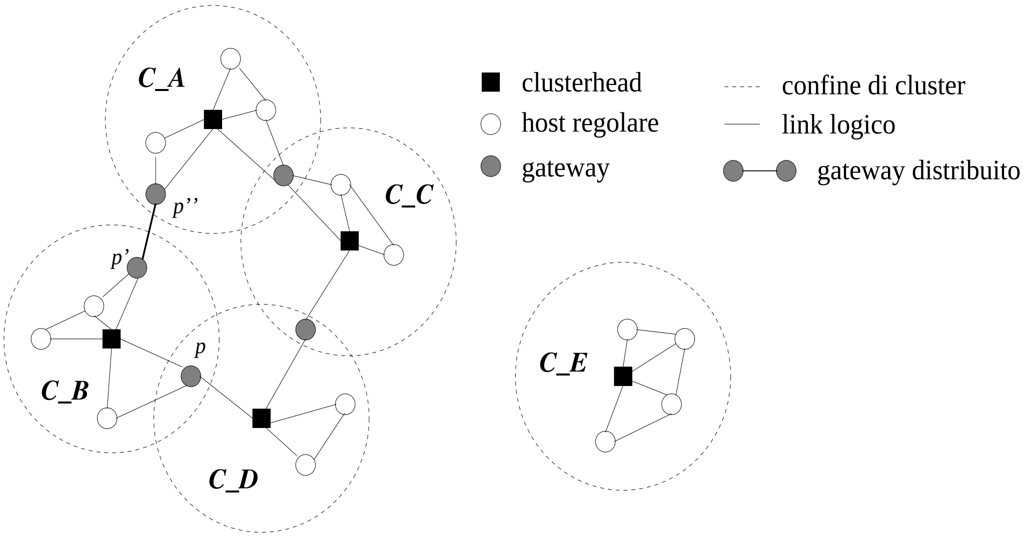


Figura 2: Esempio di architettura a cluster.

cardinalità dell'insieme N. Nel seguito assumiamo che ogni SM abbia un identificatore univoco nel sistema, anche se questo non è necessariamente vero per tutti i sistemi; indicheremo le SM come $p_1, \ldots, p_n$. Si possono immaginare due SM che comunicano reciprocamente come connesse da un link *logico*; l'insieme dei link logici costituisce l'insieme E di archi del grafo G. Diciamo che due SM sono *vicine* (*one-hop*) se esse sono connesse da un link logico.

Per quanto riguarda la struttura a cluster, nel seguito indicheremo con ch_A il clusterhead di un cluster C_A. Con gw_{AB} si indica un gateway (possibilmente distribuito) tra i cluster C_A e C_B. Indicheremo i nodi che non sono nè clusterhead nè gateway come *regular host* (rh); gli host regolari in C_A si indicheranno con rh_A^1, rh_A^2 e così via. Diremo che due cluster sono *adiacenti* se sono connessi da un gateway, anche distribuito.

Per quanto riguarda il problema del multicast, indicheremo con $\mathcal{G}$ il gruppo multicast, di cardinalità g; con $\mathcal{G}$ si indicherà anche l'identificatore del gruppo, eventualmente usato per l'indirizzamento multicast. Ipotizziamo che ogni processo risieda in un differente nodo della rete (su un differente processore). Per convenienza di rappresentazione, non distingueremo tra un nodo e il suo processo residente.

1.3 Computazione mobile: problematiche

Nella progettazione di protocolli per sistemi wireless, oltre a parametri di sistema abituali quali l'ampiezza di banda disponibile, il tasso di errore, la disponibilità di memoria, è necessario prendere in considerazione anche quelle caratteristiche peculiari delle SM e dei canali wireless che rendono questi sistemi completamente differenti dalle tradizionali reti fisse. In questa sezione si discutono l'impatto di tali caratteristiche sullo sviluppo di protocolli, e le nuove problematiche da affrontare nello studio di questi sistemi, che non si ponevano nei sistemi fissi.

1.3.1 Mobilità

Lo spostamento delle SM all'interno del sistema provoca il cambiamento della topologia della rete wireless anche in assenza di guasti. Questo rende più difficoltosa la consegna a destinazione dei messaggi poichè il loro instradamento deve variare dinamicamente. In particolare, se una SM si muove rapidamente, è teoricamente possibile che un messaggio ad essa destinato non possa mai essere consegnato: infatti, il messaggio potrebbe essere instradato in modo da arrivare ad una locazione della SM sempre dopo che essa si è spostata in una nuova locazione. Questa eventualità evidentemente può impedire la terminazione di qualunque algoritmo distribuito. Per questa ragione, viene usualmente fatta una *assunzione sulla mobilità* [1, 2, 28] per assicurare la vivezza del sistema, consistente in generale nell'ipotizzare che lo spostamento delle SM nel sistema avvenga in modo sufficientemente lento da permettere il completamento di alcune operazioni critiche per il protocollo nell'intervallo tra due successivi cambiamenti di topologia. Frequentemente tali operazioni consistono in una procedura di *handoff* [2] avente come scopo lo scambio di informazioni di stato rilevanti tra la vecchia e la nuova SSM di una SM (nei sistemi single-hop) o tra due SM che sono diventate vicine (nei sistemi multi-hop). Nei capitoli successivi si stabilirà più precisamente l'assunzione fatta nel caso dei protocolli descritti in questo lavoro.

Come ulteriore conseguenza della mobilità, i protocolli di instradamento sono chiamati più frequentemente ad aggiornare le proprie tabelle rispetto a quanto avviene nei sistemi fissi, il che impone che tali algoritmi abbiano un basso costo in termini di utilizzo di banda, computazione ed occupazione di memoria richieste.

1.3.2 Disconnessioni e *doze mode*

Durante i loro spostamenti all'interno del sistema, le SM possono trovarsi in zone non coperte dalla rete wireless, ovvero senza alcun vicino one-hop; è il caso ad esempio della SM_x in Figura 1(a) e della SM_y in Figura 1(b). In tal caso, la SM è *disconnessa* dal resto del sistema, ovvero non esiste alcun cammino che consenta di inviare ad essa messaggi. Una SM può anche disconnettersi per il proprio spegnimento, a causa dell'esaurimento della carica delle proprie batterie o di un guasto. La disconnessione di una SM ha durata indeterminata; al termine la SM può riconnettersi in una locazione diversa dalla precedente.

La disconnessione di una SM è simile al fallimento crash di una macchina nei sistemi tradizionali, ma si differenzia da esso per la possibilità che la macchina continui ad eseguire computazioni su dati mantenuti localmente (ad esempio nel caso in cui sia disconnessa perchè troppo lontana da qualunque altra stazione nel sistema). Dal punto di vista della tolleranza ai fallimenti, questo crea dei problemi ad esempio per il mantenimento della consistenza tra gli stati di stazioni coinvolte nella gestione di dati su di esse replicati. In [9] si distingue tra disconnessioni *pianificate* ed *improvvise*. Le prime sono in qualche modo prevedibili: prima che la SM si disconnetta, il resto del sistema può essere avvertito del prossimo distacco della stazione dal sistema, attraverso una procedura di handoff, ed agire opportunamente in accordo ai requisiti dei servizi forniti. Ad esempio, se la SM possiede una copia di una base di dati replicata, può decidere di rinunciare ad essa prima della disconnessione: tutte le altre stazioni coinvolte nell'applicazione sapranno che quella SM non parteciperà da quel momento in poi al protocollo per l'esecuzione delle transazioni [13, 15, 16]. Al contrario, la scomparsa inaspettata di una SM può dover essere trattata come un fallimento: nell'esempio precedente, se si sta adottando un meccanismo di locking per garantire la correttezza delle transazioni, è necessario raggiungere un accordo riguardo all'uscita dal sistema della SM per poter rimuovere suoi eventuali lock ed evitare il blocco dell'intero sistema.

Una SM può anche entrare in *doze-mode*, quando non è coinvolta nello scambio di messaggi: allo scopo di risparmiare energia sospende tutte le proprie funzioni[1], e mantiene attiva solo l'interfaccia hardware alla rete wireless *in ricezione*. Non appena viene ricevuto un messaggio indirizzato alla macchina, questa torna al normale modo di operazione. Evidentemente, il doze-mode non è una vera e propria disconnessione, perchè la SM resta comunque raggiungibile ed in grado di comunicare con il resto del sistema.

Sia il doze mode che le disconnessioni pianificate possono essere utili per prolungare la durata della carica delle batterie; infatti la comunicazione è la funzione più dispendiosa dal punto di vista del consumo di energia.

1.3.3 Canali wireless

La comunicazione sui canali wireless è maggiormente inaffidabile rispetto ai canali fisici. I pacchetti possono essere persi o danneggiati. Il ritardo di trasmissione è altamente variabile a causa della mobilità, della possibile coesistenza di diverse tecnologie wireless nello stesso sistema, e della variabilità delle condizioni di traffico. Di conseguenza, i sistemi wireless possono essere assimilati a sistemi *asincroni*, cioè sistemi in

[1]Ad esempio, ferma il disco. Si veda [69] per maggiori approfondimenti sulle tecniche per il risparmio di energia.

cui non esistono garanzie sulla sincronizzazione fra i clock dei processori, nè limiti al massimo ritardo di trasmissione dei messaggi [36, 44].

1.3.4 Problematiche inerenti lo sviluppo di protocolli

Le caratteristiche delle SM descritte nelle precedenti sezioni pongono indirettamente altri problemi che devono essere affrontati nello sviluppo di protocolli per reti mobili [46, 9, 51, 53]; discutiamo nel seguito i principali.

Locazione. I protocolli devono risolvere il problema della *locazione* degli utenti nel sistema, ovvero della determinazione della posizione corrente di una SM nel momento in cui deve essere inviato un messaggio ad essa. Inoltre deve essere nota la locazione delle risorse condivise, ovvero da quale SM esse siano gestite ad un dato momento. In letteratura sono state proposte diverse tecniche per la manutenzione dei dati aggiornati riguardo all'attuale posizione delle SM connesse in rete [10, 47, 38, 52].

Tolleranza ai fallimenti. Come osservato, mobilità e disconnessioni possono essere interpretate come fallimenti apparenti delle SM. Si è valutato che queste caratteristiche delle SM produrranno un tasso di fallimento di un ordine di grandezza superiore a quanto osservato nei sistemi fissi (vale a dire, per quanto riguarda lo scambio di messaggi, l'errore di un bit ogni 10^5 circa). I meccanismi di gestione dei fallimenti adottati dagli algoritmi per la comunicazione di gruppo in sistemi fissi (si vedano ad esempio [17, 6]) hanno un overhead non trascurabile, che può essere accettabile quando i malfunzionamenti si presentino raramente. Nel caso dei sistemi mobili l'uso di simili meccanismi sarebbe più frequente ed il loro costo diverrebbe inaccettabile. Ciò comporta la necessità di individuare tecniche innovative per la gestione dei comportamenti "anomali" nelle reti wireless. Una possibile soluzione consiste nel delegare all'applicazione la gestione della composizione del gruppo dei partecipanti e l'identificazione dei processi falliti in modo da soddisfare i propri specifici requisiti; questa è la scelta fatta nel caso dei protocolli proposti nei capitoli 4 e 5.

Amministrazione delle risorse. Nonostante le walkstation abbiano prestazioni comparabili con quelle delle workstation, esse sono comunque vincolate ad una fonte di energia di capacità limitata (le batterie), almeno durante gli spostamenti. Allo scopo di aumentarne la durata, è indispensabile costruire protocolli con basso overhead sia per quanto riguarda la computazione richiesta alle SM che per quanto riguarda lo scambio di messaggi[2]. Se, come nel caso delle reti single-hop, sono presenti nel sistema

[2]In [14] si valuta che il funzionamento dell'interfaccia hardware alla rete wireless *in sola ricezione* riduca del 25% la durata delle batterie di una SM rispetto al caso in cui la SM non sia equipaggiata con tale interfaccia. L'invio di messaggi è ancora più costoso in termini di dispendio di energia.

delle SF o comunque stazioni con maggiori risorse di altre, è naturale progettare protocolli che sfruttino tali caratteristiche imponendo un carico di lavoro maggiore alle macchine più potenti. Nel caso contrario (considerato in questo lavoro) i protocolli dovranno essere a *controllo distribuito* così che nessuna delle SM abbia un ruolo preponderante e possa costituire un collo di bottiglia.

La banda dei canali wireless è significativamente più bassa di quella disponibile nelle reti fisse; di conseguenza i protocolli dovranno avere un basso costo in termini di quantità di messaggi generati e dovranno includere meccanismi per il controllo della *congestione*.

1.4 Protocolli per reti mobili ad-hoc

Nelle sezioni precedenti si è data una descrizione generale delle caratteristiche dei sistemi wireless multi-hop e si sono esaminati gli effetti che queste caratteristiche hanno sullo sviluppo di protocolli e le nuove problematiche che è necessario affrontare. I requisiti che i protocolli dovranno soddisfare dipendono inoltre dalle potenziali applicazioni delle reti mobili. La tecnologia delle reti mobili ad-hoc può essere impiegata per la realizzazione di servizi di comunicazione personale (Personal Communication Systems, PCS) o per consentire l'accesso a Internet ad utenti mobili. Si prevede però che sarà utile principalmente per la realizzazione di applicazioni specializzate in situazioni in cui non sia disponibile un'infrastruttura di comunicazione fissa. Esempi di queste applicazioni riguardano il coordinamento di gruppi di lavoro, il coordinamento delle operazioni di soccorso in situazioni di emergenza o di operazioni militari, la gestione del movimento di veicoli.

In questa sezione si discutono le caratteristiche ed i servizi che dovrebbero essere offerti da un'architettura di riferimento per reti ad-hoc; si vedano [46, 53, 51, 42] per un approfondimento su questi temi.

Criteri di progettazione

Oltre a risolvere i problemi discussi in sezione 1.3, i protocolli per reti wireless ad hoc devono soddisfare ulteriori requisiti. Già ora esistono differenti tecnologie wireless, e potrebbe essere addirittura desiderabile equipaggiare le SM con interfacce a diversi tipi di rete, così da migliorare le comunicazioni scegliendo di volta in volta la rete che offre la migliore qualità di servizio. L'**indipendenza** dei protocolli di più alto livello dalle caratteristiche della rete è di conseguenza un requisito desiderabile per garantire la possibilità di operare nelle diverse infrastrutture in cui una SM può trovarsi connessa durante i propri spostamenti. Un aspetto negativo di questa caratteristica è che una più stretta relazione con i protocolli di livello inferiore potrebbe permettere di ottenere migliori prestazioni.

Dato che le diverse tecnologie offrono diverse disponibilità di banda e qualità di servizio, i protocolli dovrebbero essere **adattabili**, ovvero adeguare il proprio modo di operare in funzione delle caratteristiche della rete nella locazione corrente. Questa proprietà rende i protocolli più *robusti* ai cambiamenti che possono verificarsi nella rete wireless.

La locazione dei dati nel sistema può cambiare dinamicamente a causa della mobilità; inoltre, frequentemente copie dei dati potranno essere elaborati localmente nella cache delle SM. In conseguenza di questo, dovranno essere affrontati problemi correlati alla **replicazione dinamica** dei dati sulle SM e i concetti base nella gestione dei dati, come la loro integrità o la possibilità di recupero dai fallimenti, dovranno essere ridefiniti [15]. Probabilmente si farà ricorso alla replicazione delle informazioni anche per ottenere buone prestazioni ed alta disponibilità dei dati nonostante la mobilità delle stazioni o la possibile congestione di alcune zone della rete wireless.

I vincoli sulla disponibilità di energia delle SM rendono desiderabile che i protocolli operino **on demand**, vale a dire solo quando è necessario fornire il proprio servizio. Analogamente, i protocolli dovrebbero coinvolgere solo le SM effettivamente interessate al servizio, permettendo perciò alle altre SM di restare inattive (doze) e conservare la carica delle batterie.

La **scalabilità** è un'altra caratteristica fondamentale per i protocolli, in quanto si stima che il numero di macchine coinvolte nelle applicazioni distribuite sarà superiore nei sistemi wireless rispetto ai sistemi fissi.

Servizi e algoritmi fondamentali

Oltre ai servizi comunemente presenti nelle reti fisse, altri sono necessari come deriva dalle considerazioni presentate nelle sezioni precedenti; è il caso ad esempio dei protocolli per la manutenzione delle informazioni di locazione delle SM e dei meccanismi per il controllo della congestione.

Per quanto riguarda il campo di ricerca di questa tesi, si è precedentemente notato come una caratteristica essenziale dei protocolli per reti wireless sia la distribuzione del controllo [53] e, parzialmente correlata a quest'ultima, la capacità di gestire dati replicati a tutti i livelli dell'architettura [51]. Inoltre, la maggior parte delle applicazioni specifiche per cui si prevede l'utilizzo di reti mobili ad hoc prevede la cooperazione e il coordinamento tra gruppi di utenti nomadi. Questi fattori fanno sì che *servizi di supporto alla comunicazione di gruppo* diventino di importanza fondamentale in questi sistemi, come rilevato in [42]. Tali servizi consistono in particolare nella realizzazione di strutture di instradamento multicast, nella trasmissione affidabile di messaggi multicast e, in futuro, nel supporto al traffico con qualità di servizio.

Parametri del sistema

Le prestazioni dei protocolli per sistemi wireless possono essere misurate facendo riferimento agli indici usuali quali throughput, ritardo e costo in termini di numero di messaggi generati. Le prestazioni possono essere ottimizzate prendendo in considerazione le caratteristiche delle reti in cui i protocolli dovranno operare. Esempi di tali parametri sono la taglia della rete (numero di SM), la connettività della rete e la frequenza di cambiamento della topologia, la banda dei canali wireless, la probabilità di spostamento e disconnessione delle SM e la distribuzione del traffico, determinata dal numero e dall'identità delle SM sorgenti del traffico e dalla frequenza con cui esse generano messaggi. Nel capitolo 3 e in parte nei capitoli 4 e 5 si definirà con precisione il tipo di ambiente per cui sono stati progettati e in cui sono stati valutati i protocolli presentati in questa tesi, con attenzione anche ai protocolli di basso livello utilizzati, le cui caratteristiche influiscono sul comportamento dei protocolli di livello superiore.

1.5 Conclusioni

In questo capitolo è stata data una descrizione generale delle caratteristiche dei sistemi distribuiti comprendenti stazioni mobili. È stato discusso l'impatto di tali caratteristiche sullo sviluppo di protocolli per questi sistemi e si sono analizzati i requisiti che i protocolli e l'architettura dovrebbero soddisfare. Si è osservato come, date le innovative caratteristiche dei sistemi mobili, i protocolli finora proposti per reti fisse non sono adeguati all'impiego in ambito mobile. Ciò rende necessario sviluppare protocolli originali appositamente progettati per operare in reti wireless. Dei servizi essenziali da realizzare per questi sistemi fanno parte i servizi di supporto alla comunicazione di gruppo, a causa anche della necessaria distribuzione del controllo e conseguente replicazione dei dati. In questa tesi si propongono due protocolli aventi lo scopo di realizzare due fondamentali servizi per la comunicazione di gruppo, ovvero il servizio di instradamento multicast ed il servizio di consegna affidabile di messaggi broadcast. Nei prossimi capitoli si definiranno precisamente i problemi risolti da questi servizi e l'architettura di riferimento considerata, e si descriveranno in dettaglio i due protocolli citati.

Capitolo 2

Problemi di comunicazione di gruppo

2.1 Introduzione

La realizzazione di servizi di supporto alla comunicazione all'interno di gruppi di processi cooperanti impone la soluzione di numerosi problemi, connessi sia all'amministrazione delle risorse della rete necessarie per la gestione di un traffico uno-a-molti, sia alla qualità del servizio fornito alle applicazioni. In [32] i problemi relativi alla comunicazione di gruppo vengono classificati in tre categorie:

1. a livello '*hop-by-hop*': problemi di indirizzamento di gruppo, instradamento multicast, gestione della composizione del gruppo e controllo di congestione;

2. a livello '*end-to-end*': problemi di trasmissione affidabile dei messaggi e qualità di servizio;

3. a livello '*applicazioni*': problemi di consegna ordinata dei messaggi e di sincronizzazione fra processi, sicurezza.

In questa tesi ci si occuperà dei due servizi fondamentali tra quelli citati: il servizio di *Reliable Broadcast* ed il servizio di *instradamento multicast*, che si collocano rispettivamente al livello end-to-end e al livello hop-by-hop.

Reliable Broadcast. In [44] è descritta una gerarchia di problemi di comunicazione di gruppo *affidabile*. In questo lavoro ci concentriamo sul problema più semplice della gerarchia, ovvero quello di *Reliable Broadcast*. Un protocollo per il Reliable Broadcast è la base per la costruzione di protocolli che risolvano i problemi più complessi della gerarchia. Tali problemi, oltre a richiedere la consegna affidabile dei messaggi alle destinazioni, impongono che i destinatari consegnino in un ordine consistente tutti i messaggi generati dalla stessa sorgente (FIFO *Broadcast*), i messaggi legati da una relazione di dipendenza causale (*Causal Broadcast*), o tutti i messaggi inviati alle destinazioni in accordo ad un ordinamento globale (*Atomic Broadcast*). In tutti i citati problemi di broadcast l'insieme delle destinazioni di un messaggio è l'insieme N di tutti i nodi del sistema. Come osservato in [32, 37], la scelta dell'ordinamento da imporre ai messaggi dovrebbe essere lasciata all'applicazione in funzione della semantica del servizio richiesto, in accordo allo *end-to-end argument*; per questa ragione, si è scelto di occuparsi inizialmente solo dell'affidabilità della trasmissione. Mobilità e disconnessioni rendono necessario modificare la definizione del problema del Reliable Broadcast usualmente considerata in letteratura; in sezione 2.2 si dà una nuova formulazione del problema, per cui il protocollo descritto nel capitolo 4 rappresenta una possibile soluzione.

Instradamento multicast. Il servizio di instradamento multicast ha lo scopo di individuare lungo quali cammini devono essere inviati i messaggi multicast, così da contenere il consumo di banda. Il modo più semplice di instradare un messaggio ad un gruppo $\mathcal{G}$ di destinatari è quello di generare tante copie quanta è la cardinalità g di $\mathcal{G}$ e inviarle separatamente alle destinazioni usando il servizio di instradamento unicast. Tale tecnica comporta uno spreco di banda, di maggiore rilevanza nelle reti mobili, e non sfrutta la natura broadcast del canale wireless; inoltre, i messaggi non sono indirizzati a $\mathcal{G}$ ma ai singoli destinatari, obbligando perciò le sorgenti a conoscere la composizione di $\mathcal{G}$. I cammini individuati dal servizio di instradamento multicast devono avere basso costo e devono essere mantenuti aggiornati nel caso di cambiamenti della topologia. In sistemi wireless, ciò significa che il servizio di instradamento deve far fronte, oltre che ai fallimenti, anche ai problemi legati alla mobilità delle SM destinatarie dei messaggi multicast e alla loro possibile disconnessione dalla rete. In sezione 2.3 si definisce il problema di instradamento multicast in reti mobili ad-hoc che ci si propone di risolvere con il protocollo proposto nel capitolo 5.

Allo scopo di valutare la correttezza di entrambi i protocolli nel risolvere i problemi introdotti, è necessario formalizzare il modello del sistema in cui i protocolli operano; questo è fatto in sezione 2.4. Infine, in sezione 2.5 viene data una breve panoramica dei lavori proposti in letteratura per la soluzione dei problemi considerati.

2.2 Il problema del Reliable Broadcast

Nei sistemi fissi la soluzione del problema di Reliable Broadcast richiede che: (i) se un processo corretto genera un messaggio, tutti i processi corretti lo ricevono, e (ii) se un processo corretto consegna[1] un messaggio tutti i processi corretti lo consegnano. Per processo *corretto* si intende un processo non soggetto ad alcun fallimento; in [44] sono descritti i modelli di fallimento dei sistemi fissi considerati in letteratura.

Questa definizione non è adatta all'ambiente mobile, in cui un processo corretto (*attivo*) ma disconnesso potrebbe non ricevere mai un messaggio. Per superare questo ostacolo, nel seguito si farà riferimento alla seguente definizione del problema, leggermente modificata rispetto a quella usualmente adottata in letteratura:

Definizione 2.2.1 [Problema del Reliable Broadcast] *Data una rete di n stazioni in cui ogni stazione, ad ogni istante, può generare un messaggio broadcast, i messaggi sono consegnati a tutte le destinazioni in modo tale che le seguenti proprietà siano soddisfatte:*

Validità e Accordo: *se una stazione fa broadcast di un messaggio m allora tutte le stazioni nella rete ricevono m (consegna at-least-once).*

Integrità: *per ogni messaggio m, tutte le stazioni consegnano m al più una volta e solo se m è stato inviato broadcast da qualche stazione della rete.*

Terminazione: *se una stazione p fa broadcast di un messaggio m, allora le proprietà di Validità, Accordo e Integrità sono soddisfatte per m entro un tempo finito.*

In accordo alla definizione data, la consegna dei messaggi è garantita a *tutte* le SM nella rete, e il requisito di affidabilità è esteso a *tutti* i messaggi broadcast generati. Dalla definizione, il servizio di Reliable Broadcast fornisce una semantica di consegna *exactly-once*.

Per soddisfare il requisito generale di risparmio delle risorse [46], noi imponiamo un'ulteriore proprietà riguardo ai messaggi diffusi:

Proprietà 2.2.1 [Quiescenza eventuale] *Per ogni messaggio m inviato broadcast, prima o poi non sono più generati messaggi riguardanti m e m può essere cancellato dalla memoria delle SM.*

Questa proprietà garantisce che tutte le risorse di computazione, memoria e banda utilizzate per la diffusione affidabile di un messaggio m sono rilasciate.

2.3 Il problema dell'instradamento multicast

In letteratura sono stati proposti algoritmi per la costruzione di strutture di instradamento multicast in sistemi fissi. Una struttura usata frequentemente è quella ad *albero*, costruita ad esempio dagli algoritmi *Core-Based-Tree* [12], *PIM-SM* e *PIM-DM* [31] proposti per Internet. L'uso di una struttura ad albero fa sì che:

[1] Per *consegna* di un messaggio si intende la sua notifica ai livelli superiori dell'architettura.

1. una copia del messaggio sia creata solo ai nodi corrispondenti a biforcazioni nell'albero (ovvero, alle radici di sottoalberi);

2. il massimo ritardo di propagazione al nodo foglia più lontano dalla radice cresca meno che linearmente con g.

Nel capitolo 5 si propone un algoritmo per la costruzione di una struttura di instradamento ad albero *condiviso* in reti wireless multi-hop.

Algoritmi di questo tipo, oltre a costruire la struttura di instradamento, hanno il compito di mantenerla a fronte di cambiamenti della topologia. Nelle reti wireless la frequenza di tali cambiamenti è molto superiore a quanto osservato nelle reti fisse; a meno di assunzioni sulla mobilità, la topologia è in continua mutazione. Questo ha principalmente due conseguenze sul problema da risolvere:

1. a causa della variazione dinamica della topologia, la terminazione dell'algoritmo può essere garantita solo se la connettività resta invariata per un tempo "sufficientemente" lungo[2];

2. l'albero ha alta probabilità di partizionarsi. Per favorire il risparmio delle risorse delle SM, ed evitare spreco di banda dei canali wireless, i rami obsoleti devono essere rapidamente rimossi.

In accordo a queste considerazioni, il problema che l'algoritmo presentato in questa tesi si propone di risolvere si può definire come segue, analogamente a [28]:

Definizione 2.3.1 [Problema dell'instradamento multicast] *Un protocollo $\mathcal{P}$ risolve correttamente il problema dell'instradamento multicast per un gruppo $\mathcal{G}$ di stazioni se soddisfa le seguenti proprietà:*

Liveness(1): *data una rete wireless con topologia statica arbitraria, il protocollo garantisce che entro un tempo finito tutte le destinazioni in $\mathcal{G}$ non partizionate dal nodo radice dell'albero sono connesse all'albero.*

Liveness(2): *il protocollo è esente da deadlock.*

Safety(1): *durante periodi di cambiamenti arbitrari della topologia, la struttura mantenuta dal protocollo è priva di cicli.*

Safety(2): *in caso di cambiamenti arbitrari di topologia, entro un tempo finito i cammini dell'albero non più validi sono cancellati.*

[2] Ovvero, per il tempo necessario all'esecuzione dell'algoritmo.

Il primo punto della definizione garantisce che il protocollo termina se prima o poi la topologia della rete resta stabile per un tempo "sufficientemente" lungo; tale ipotesi è affine all'assunzione sulla mobilità introdotta in sezione 1.3.1. Il terzo punto garantisce che la struttura prodotta è aciclica; questa proprietà è importante per garantire che non si creino duplicati di messaggi, che sprecherebbero inutilmente banda, e che un messaggio non resti in circolazione indefinitamente. Il quarto punto garantisce che un messaggio non viene instradato su cammini obsoleti, e che i nodi appartenenti a tali cammini entro un tempo finito non sono più coinvolti nel processo di instradamento, evitando perciò uno spreco sia di banda che di risorse delle SM.

2.4 Modello di fallimento

In questa sezione si formalizza il modello di fallimento considerato per la soluzione dei due problemi precedentemente definiti. I modelli di fallimento tradizionalmente adottati in letteratura [44] non descrivono le peculiari caratteristiche dei sistemi mobili. In tali modelli tutti i malfunzionamenti sono imputati ai processori, mentre i canali di comunicazione vengono assunti affidabili. Inoltre, un processore che si discosti ad un certo momento dalla corretta esecuzione del protocollo, ad esempio omettendo di inviare o ricevere un messaggio, da quel momento in poi è ritenuto *fallito*. Le definizioni usuali dei problemi di comunicazione di gruppo [44] non impongono vincoli sul comportamento dei processori falliti e non garantiscono che per essi il problema sia correttamente risolto. Frequentemente, per garantire o facilitare la terminazione di un protocollo distribuito viene sfruttato un servizio di *group membership* [19, 21, 33, 54, 20, 7]. Tale servizio ha lo scopo di individuare i processori falliti ed eliminarli dal gruppo dei partecipanti al protocollo, mantenendo l'accordo sulla corrente composizione del gruppo tra i processori rimanenti.

L'adozione di uno di questi modelli per descrivere il comportamento dei sistemi mobili presenta diversi aspetti negativi:

1. una SM che si disconnette temporaneamente dalla rete wireless può essere sospettata di fallimento crash, e pertanto non è garantita terminare correttamente il protocollo nonostante non sia realmente malfunzionante;

2. la rete wireless ha un alto tasso di fallimento. L'imputazione dei suoi errori alle SM può far considerare fallite stazioni che operano correttamente, con le medesime conseguenze di cui al punto precedente;

3. il ricorso alla group membership sarebbe più frequente non solo per la gestione dei fallimenti apparenti di cui ai punti precedenti, ma possibilmente anche per la reintegrazione nel gruppo dei partecipanti al protocollo delle SM erroneamente escluse, il che è oneroso dal punto di vista del consumo di risorse.

Il modello di sistema che si considera in questo lavoro, e rispetto al quale verrà dimostrata la correttezza dei protocolli proposti, descrive sia i fallimenti delle SM, sia quelli dei canali di comunicazione, sia gli effetti di mobilità e disconnessioni sulla comunicazione affidabile tra processi, e viene definito come segue [60]:

Definizione 2.4.1 [Modello di fallimento mobile]

1. *le SM possono disconnettersi o fallire temporaneamente. Non si fanno ipotesi sulla durata di una disconnessione o di un malfunzionamento;*

2. *la rete può partizionarsi per un tempo indeterminato;*

3. *i canali di comunicazione possono smarrire i messaggi.*

Si noti che non viene data un'esplicita definizione di SM *fallita*; analogamente, nelle definizioni 2.3.1 e 2.2.1 dei problemi risolti dai protocolli proposti nei prossimi capitoli, non si distingue tra il comportamento delle SM corrette e fallite. D'altra parte, si impone che i fallimenti siano solo *temporanei*; questa è anche una conseguenza della scelta architetturale di non realizzare momentaneamente un servizio di group membership.

Per garantire la terminazione eventuale dei protocolli proposti nei capitoli successivi, si potranno fare delle assunzioni sulla mobilità che porranno alcune restrizioni sul modello generale di fallimento ora definito; queste assunzioni verranno introdotte e discusse nel seguito del lavoro.

2.5 Stato dell'arte

In questa sezione diamo una breve panoramica dei lavori esistenti in letteratura riguardanti i problemi di diffusione affidabile di messaggi a utenti su SM, e di instradamento di messaggi nel caso di comunicazione "*uno-a-molti*" e "*uno-a-tutti*".

2.5.1 Protocolli per sistemi single-hop

I protocolli finora proposti in letteratura per risolvere il problema del Reliable Broadcast (o Multicast) in sistemi con SM considerano solo il caso di reti single-hop[3].

In [1, 2] si presenta un protocollo che fornisce i servizi di Reliable Broadcast e Reliable Multicast con semantica di consegna *exactly-once*. La computazione è affidata pesantemente alle SSM: esse devono preoccuparsi di mantenere i messaggi e lo stato di consegna degli stessi alle SM nelle proprie celle, e di diffondere i messaggi a tutte le destinazioni. I fallimenti possono riguardare solo la componente wireless

[3] Per approfondimenti sul problema della gestione di dati replicati su SM si veda anche [15, 16].

del sistema; essi vengono superati per mezzo di ritrasmissioni, e la corretta ricezione dei messaggi viene confermata dalle SM mediante l'invio di acknowledgement. La mobilità di una SM viene gestita mediante lo scambio delle informazioni riguardanti il suo stato tra la sua vecchia e la nuova SSM durante la procedura di *handoff*.

In [27] si ipotizza che le SM non abbiano risorse sufficienti all'esecuzione di un protocollo di comunicazione di gruppo. Per questa ragione l'intera elaborazione viene affidata alle SF, che eseguono i protocolli offerti da ISIS [17]; una SM semplicemente invia i messaggi da diffondere alla propria SSM. Analogamente a [1, 2], le SSM si incaricano di mantenere le informazioni riguardanti lo stato delle proprie SM. Per garantire un più efficiente supporto alla mobilità, queste informazioni di stato sono replicate sulle SSM vicine, così da aumentare la probabilità che, quando una SM si sposta, entri nella cella di una SSM che già possiede il suo stato. L'insieme di SSM in possesso dello stato di una SM varia dinamicamente in funzione degli spostamenti di quest'ultima; la consistenza tra le SSM viene garantita dall'uso di ISIS.

Entrambi questi approcci fondano la propria correttezza sull'assunzione che le SF e la rete fissa siano affidabili, perciò nessun messaggio può essere smarrito; se questa ipotesi viene meno, gli algoritmi citati possono comportarsi in modo scorretto. Questa è la ragione principale per cui il loro uso non è possibile in reti multi-hop.

2.5.2 Protocolli per sistemi multi-hop

Per i sistemi multi-hop sono stati ideati alcuni algoritmi di instradamento di supporto alla comunicazione *uno-a-molti*. Si possono adottare due approcci principali per la costruzione di un'infrastruttura di instradamento multicast:

1. si può determinare l'albero dei cammini più brevi (*Shortest Path Tree*, SPT) dalla sorgente di ogni messaggio a tutti i membri del gruppo destinatario (come in DVMRP [30] e MOSPF [55]);

2. si può usare un albero condiviso (*Single Shared Tree*, SST) utilizzato da tutte le sorgenti di messaggi (come nel protocollo CBT [12]).

La scelta di ricorrere ad un albero condiviso presenta numerosi vantaggi in termini di efficienza sia per quanto riguarda la quantità di memoria e computazione richiesta ai processori, sia riguardo all'uso della banda della rete:

- se indichiamo con g la cardinalità del gruppo multicast $\mathcal{G}$ e con S il numero di processori sorgenti di messaggi ad esso indirizzati, un algoritmo che mantiene uno spanning tree da ogni sorgente ad ogni elemento del gruppo ha costo proporzionale a $S \times g$. Al contrario, gli algoritmi che costruiscono un albero condiviso hanno costo proporzionale a g;

- il guasto di un processore o di un link rende necessario riconfigurare tutti gli spanning tree che fanno uso del nodo o link guasto, ovvero S nel caso peggiore, mentre con l'approccio SST si deve mantenere un solo albero;

- un nodo deve memorizzare tutti gli spanning tree di cui fa parte (potenzialmente, S), e partecipare alle procedure per il loro mantenimento;

- tutti i nodi che si trovano sul cammino più breve tra una sorgente ed uno dei membri del gruppo sono inclusi nello spanning tree di quella sorgente. Al contrario, in un albero condiviso sono coinvolti solo i nodi che si trovano sul cammino più breve tra la radice ed una delle altre destinazioni, mentre i nodi sul cammino più breve tra una sorgente e la radice sono chiamati solo ad instradare verso quest'ultima dei messaggi unicast, quando necessario.

È evidente che le caratteristiche elencate rendono l'approccio ad albero condiviso attraente nel caso di sistemi wireless. Inoltre, questo è forse l'unico approccio possibile per ottenere l'interoperabilità nel caso di interconnessione tra una sottorete mobile e sottoreti comprendenti stazioni fisse, grazie all'esistenza di un unico nodo che agisce come agente per le SM. D'altra parte, l'utilizzo di un albero condiviso presenta anche alcuni svantaggi; in particolare: (i) la centralità del nodo radice che lo rende un potenziale collo di bottiglia ed il cui fallimento lascia completamente partizionato l'albero; (ii) la concentrazione del traffico su un sottoinsieme dei canali della rete, ovvero quelli che connettono i membri del gruppo; (iii) una degradazione delle prestazioni, perchè l'albero condiviso può non fornire il cammino migliore tra le sorgenti e le destinazioni. Questi punti sono ancora parzialmente veri in ambiente mobile. Se i riceventi si muovono in modo solidale, effettivamente il traffico multicast interessa lo stesso insieme di canali; ma quando le SM si muovono in modo casuale (ad esempio, nel caso di applicazioni per Personal Communication o per soccorso in caso di disastro) il traffico tende a distribuirsi uniformemente nella rete in conseguenza della dinamica del gruppo, perciò interessando tutti i canali wireless.

Noi abbiamo scelto l'approccio SST perchè riteniamo che nell'ambiente mobile i benefici siano superiori agli svantaggi. Le precedenti considerazioni indicano che questo è vero in particolare nel caso di sorgenti con basso tasso di generazione di messaggi; in caso contrario, è necessario prevedere un meccanismo di controllo della congestione al nodo radice dell'albero, come è stato fatto nel caso del protocollo descritto nel capitolo 5.

Finora, poche soluzioni sono state proposte per reti mobili multi-hop. In [28] Corson e Ephremides propongono un algoritmo che costruisce alberi di instradamento orientati alle destinazioni. Una SM che voglia mandare messaggi ad un gruppo di host invia ad ognuno di essi un'interrogazione. Ogni destinazione fa il flooding della risposta; il flooding traccia tutti i cammini possibili tra la destinazione ed ogni altro nodo del sistema, ed in particolare i cammini al sender. I nodi che vengono attraversati dai messaggi di risposta ed il sender stesso memorizzano i link da cui questi messaggi sono stati ricevuti, che sono i link lungo i quali vanno inoltrati i messaggi per la destinazione. La risposta all'interrogazione di un sender può essere anche inviata da un altro nodo che già possiede un cammino alla destinazione cercata. L'algoritmo è dotato di meccanismi per cancellare i cammini che diventano obsoleti a causa dei cambiamenti di topologia dovuti a fallimenti o mobilità. I cammini vengono ricostruiti *on-demand*, ovvero solo alla successiva spedizione di un messaggio.

La tecnica di costruzione dei cammini a partire dalla destinazione è vantaggiosa in quanto non richiede l'utilizzo di ack per controllare che la costruzione sia avvenuta con successo (o meglio, l'ack coincide con il messaggio che costruisce il cammino). Nelle reti wireless però un canale tra due nodi può avere costi differenti nelle due direzioni, o addirittura essere *unidirezionale*; in questo caso, la scelta di costruire i cammini nella direzione opposta a quella in cui verranno trasmessi i messaggi può produrre cammini subottimi o addirittura inesistenti. Inoltre l'algoritmo non è effettivamente orientato alla comunicazione di gruppo, perchè non supporta l'indirizzamento multicast: il sender deve conoscere l'identità di ogni destinazione a cui desidera inoltrare il messaggio, e trovare un cammino ad essa. La trasparenza della composizione del gruppo è ritenuta uno dei requisiti fondamentali che caratterizzano un servizio multicast [29]. L'algoritmo, sotto opportune assunzioni sulla mobilità, è stato formalmente dimostrato corretto, ovvero produce una struttura connessa e senza loop entro tempo finito. L'algoritmo *non* garantisce la consegna affidabile dei messaggi: una volta costruiti i cammini, se un link fallisce mentre un messaggio è in transito, l'algoritmo è in grado di trovare un cammino alternativo (a meno di partizioni nella rete) ma il messaggio è irrimediabilmente perso.

In [22] Chiang e Gerla propongono un algoritmo per la costruzione di un albero di instradamento *condiviso* basato sul protocollo *CBT* ideato per Internet [12, 66], adattato alle reti multi-hop. Questo protocollo è *hard-state*, vale a dire, costruisce l'albero e lo mantiene continuamente a fronte di ogni cambiamento nella topologia. In esso, una destinazione richiede al nodo radice dell'albero di essere agganciata all'albero, o quando entra nel gruppo o quando ha perso il proprio precedente padre nell'albero a causa dei fallimenti o della mobilità. Il ramo dell'albero è tracciato dal percorso del messaggio di risposta, inviato dal primo nodo appartenente all'albero che riceve la richiesta.

In [23] è proposta una variante *soft-state* dell'algoritmo precedente, in cui l'albero non viene mantenuto continuamente, ma le informazioni sulla sua struttura vengono periodicamente aggiornate dai nodi che ne fanno parte. Questo protocollo è ispirato al protocollo *PIM-SM* [31] proposto per Internet: periodicamente ogni nodo nell'albero manda un messaggio di *refresh* al proprio genitore; se il genitore non riceve tale messaggio entro la scadenza di un timeout, cancella il proprio discendente e non instrada più verso di esso i messaggi multicast. La durata del timeout dipende dalla mobilità.

Infine, in [24] si descrive il protocollo *FGMP* per la costruzione di un albero di instradamento soft-state, che si basa sullo sfruttamento della natura broadcast del canale wireless per ottimizzare le proprie prestazioni. Il protocollo può essere orientato al sender o alle destinazioni. Nel primo caso, i membri del gruppo destinatario fanno periodicamente flooding delle proprie identità; ogni host che desidera mandare messaggi al gruppo prende nota dei suoi appartenenti. Periodicamente un sender ricostruisce interamente il proprio albero: esso costruisce una *Forwarding Table* contenente per ogni destinazione il proprio next-hop sul cammino migliore per essa, e la invia broadcast. Ogni nodo indicato come next-hop ricorda di essere nell'albero per quel sender e ne continua la costruzione usando il medesimo procedimento; da quel momento in poi farà broadcast di ogni messaggio multicast che riceve per quel gruppo dal sender. Un nodo nell'albero che non riceve una Forwarding Table entro la scadenza di un timeout, dimentica di appartenere all'albero e smette di instradare i messaggi multicast; la dimensione del timeout dipende soltanto dalla frequenza con cui la composizione del gruppo cambia. In [24] il protocollo proposto è confrontato mediante tecniche di simulazione con il flooding e il protocollo *DVMRP* [29] proposto per Internet ed opportunamente adattato alle reti wireless.

Gli ultimi tre protocolli citati, a differenza di quello descritto in [28], supportano l'indirizzamento multicast, ma come in [28] *non* si garantisce la consegna affidabile dei messaggi.

Dal punto di vista dell'approccio utilizzato, la tecnica hard-state può essere dispendiosa, perchè la mobilità delle SM può richiedere frequenti ricostruzioni di parte dell'albero di instradamento. Se la frequenza di generazione dei messaggi multicast non è elevata, l'albero può essere riconfigurato più volte senza essere mai utilizzato nel frattempo. All'opposto, la tecnica soft-state rende indipendente l'esecuzione dell'algoritmo dalla mobilità: la struttura dell'albero viene memorizzata dai nodi che ne fanno parte per un tempo massimo predefinito; se le informazioni non vengono rinfrescate entro questo tempo, esse vengono cancellate e l'albero smette di esistere. Se la frequenza di aggiornamento è superiore al tasso di generazione dei dati, allora questo approccio presenta lo stesso inconveniente precedentemente descritto a proposito dell'approccio hard-state. Se al contrario la frequenza è inferiore al tasso di generazione dei dati, alcuni messaggi possono essere inoltrati lungo cammini obsoleti, che non portano ad alcuna destinazione, causando così delle trasmissioni ridondanti ed uno spreco di banda. Probabilmente la tecnica soft-state offre le prestazioni migliori quando la frequenza di riconfigurazione dell'albero sia uguale al tasso di generazione dei messaggi, e la riconfigurazione venga fatta appena prima della loro trasmissione.

Su questa idea si basa l'approccio utilizzato per il protocollo descritto nel capitolo 5, che ha portato alla definizione di una tecnica originale: l'approccio *ibrido*.

L'approccio ibrido è fondamentalmente un approccio soft-state in cui la rigenerazione dell'albero è attivata *on-demand* quando il nodo radice ha un pacchetto da inviare multicast. Ciò significa che il periodo di rigenerazione è adattato dinamicamente in accordo alla frequenza di arrivo dei messaggi alla radice.

2.6 Conclusioni

In questo capitolo si sono introdotte le definizioni dei problemi che i protocolli proposti in questo lavoro si prefiggono di risolvere. Si è discussa la differenza tra le definizioni date e quelle usualmente considerate in letteratura a proposito dei sistemi fissi, motivando gli adattamenti fatti. Analogamente, si è definito il modello adottato nel resto del lavoro per catturare i comportamenti anomali e i malfunzionamenti che si possono verificare nelle reti multi-hop, mettendone in luce le differenze rispetto ai modelli di fallimento esistenti per i sistemi fissi.

Infine, si sono esaminati alcuni protocolli di supporto alla comunicazione di gruppo in reti wireless. Si è osservato che i protocolli per il broadcast affidabile finora proposti in letteratura sono stati progettati per reti single-hop e non possono essere portati in reti multi-hop. Il protocolli descritto nel capitolo 4 è perciò una soluzione originale per reti multi-hop.

Per quanto riguarda il problema dell'instradamento "uno-a-molti", i protocolli disegnati per reti multi-hop non garantiscono anche la consegna affidabile, e talvolta non supportano l'indirizzamento di gruppo. Il protocollo descritto nel capitolo 5 permette l'indirizzamento di gruppo dei messaggi; esso inoltre adotta un approccio originale al problema che cerca di superare alcune debolezze presentate dalle altre soluzioni.

Capitolo 3

Architettura del sistema wireless

3.1 Introduzione

In questo capitolo si presenta l'architettura di riferimento considerata nel corso di questo lavoro e si descrivono in dettaglio i protocolli che ne costituiscono i livelli. Inoltre si descrive il simulatore usato per la valutazione delle prestazioni dei protocolli per il Reliable Broadcast e per l'instradamento multicast inaffidabile proposti nei prossimi capitoli.

3.2 Schema dell'architettura

L'architettura di riferimento considerata nel seguito del lavoro è mostrata in Figura 3.

L'interfaccia hardware alla rete wireless fornisce l'accesso fisico alla rete e filtra i messaggi in ricezione in base all'indirizzo del destinario; tale indirizzo può anche essere *multicast* (ovvero l'identificatore di un gruppo) o *broadcast* (indicato dal valore speciale `any`).

Il protocollo di livello data link si occupa dello scambio periodico di messaggi (*beacon* [2]) tra nodi vicini per mantenere aggiornate le informazioni di vicinanza. Queste informazioni sono usate dal protocollo di clustering per produrre la struttura gerarchica della rete. Il protocollo di livello MAC sfrutta la struttura gerarchica della rete per l'accesso a canale, riducendo le collisioni mediante l'uso di codici ortogonali in cluster adiacenti e l'uso di una politica a divisione di tempo per l'accesso al canale da parte dei nodi appartenenti allo stesso cluster [40]. Al di sopra del livello MAC risiede il protocollo per la costruzione delle tabelle di instradamento unicast.

Il protocollo di *Reliable Broadcast* descritto nel capitolo 4 costruisce una propria struttura di instradamento, indipendente dall'instradamento unicast, per la diffusione

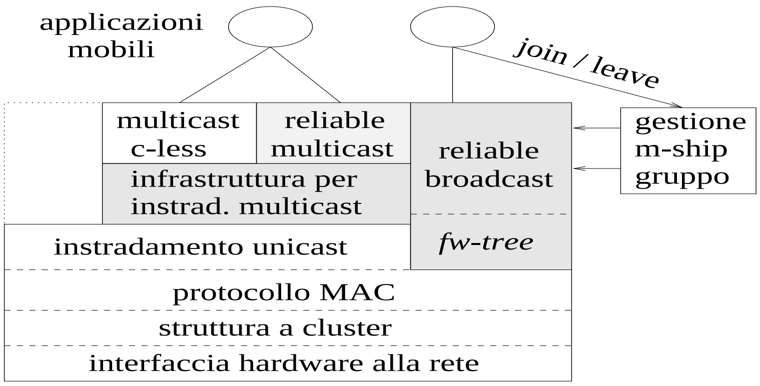


Figura 3: Architettura di riferimento.

dei messaggi (questa scelta è motivata nei capitoli 1 e 4). Al contrario, il protocollo per la costruzione di una struttura di instradamento multicast descritto nel capitolo 5 sfrutta le tabelle prodotte dall'instradamento unicast. I servizi dell'instradamento multicast inaffidabile possono essere sfruttati da sovrastanti protocolli per il *Reliable Multicast* o per il multicast inaffidabile.

La gestione dei fallimenti di membri del gruppo multicast o di richieste di ingresso ed uscita volontaria dal gruppo possono essere gestite da un modulo che fornisca il servizio di *membership*. Per questioni di semplicità, non ci occupiamo di problematiche attinenti alla manutenzione della membership dei gruppi multicast in questo lavoro.

Le applicazioni che eseguono su SM possono sfruttare i servizi non solo di comunicazione di gruppo, ma anche di *locazione* di dati, servizi e risorse all'interno del sistema.

3.2.1 Caratteristiche dei sistemi esistenti

I sistemi wireless si differenziano notevolmente tra loro per quanto riguarda la banda disponibile ed il raggio di comunicazione, in funzione della tecnologia adottata per i canali wireless. Ad esempio, le reti WAN che usano canali radio come nel caso della telefonia cellulare, hanno frequenze di trasferimento dati basse, dell'ordine di 14.4 - 64 Kbit/sec., e portata del segnale intorno ai 3 - 4 Km. [64]; recentemente si sono iniziati a sviluppare servizi con canali a 100 Kbps.

In questo lavoro prendiamo in considerazione solo reti LAN multi-hop; in accordo allo standard IEEE 802.11 [35] proposto per queste reti, abbiamo assunto che la velocità di trasferimento dei dati sia di 2 Mbps, e che la portata del segnale sia intorno ai 300 piedi (circa 100 metri). Queste misure sono indicative, poichè il segnale radio nei sistemi reali è soggetto all'influenza di una varietà di fattori, quali la presenza di ostacoli fisici sul terreno o l'interferenza con il segnale di altre sorgenti di onde radio; tuttavia, in questo lavoro non prenderemo in considerazione questi fattori.

3.3 Protocolli di basso livello

I protocolli proposti in questa tesi risiedono sopra i livelli di clustering, di accesso a canale e di instradamento unicast. In letteratura sono state proposte diverse alternative per realizzare i servizi di questi livelli [40, 22, 35]. Poichè, come si vedrà successivamente, le prestazioni degli algoritmi proposti dipendono anche dalle caratteristiche dei livelli su cui essi si basano, in questa sezione si descrivono i protocolli di basso livello adottati nel resto del lavoro.

3.3.1 Protocollo di clustering

Esistono in letteratura vari algoritmi per la costruzione di un'architettura a cluster avente le caratteristiche descritte in sezione 1.2. Essi si differenziano per la tecnica adottata nell'elezione del clusterhead; in particolare in [40, 22] sono state proposte le seguenti tecniche:

Lowest ID: periodicamente ogni SM invia broadcast ai propri vicini gli identificatori dei nodi da cui riceve, incluso se stesso. Un nodo è clusterhead se ha l'identificatore più basso di quelli di tutti i suoi vicini. Il clusterhead di un nodo p è il nodo p' con l'identificatore più basso tra quelli da cui p riceve, a meno che p' non rinunci al ruolo di clusterhead[1].

Highest connectivity: periodicamente ogni SM invia broadcast ai propri vicini gli identificatori dei nodi da cui riceve, incluso se stesso. Indichiamo un nodo come *scoperto* se non ha ancora eletto il proprio clusterhead, e *coperto* altrimenti. Un nodo è eletto clusterhead se ha la più alta connettività tra tutti i suoi vicini scoperti e non ha ancora eletto il proprio clusterhead.

Eletti i clusterhead, un nodo vicino a due o più di essi è un gateway, altrimenti è un nodo regolare.

L'algoritmo di clustering viene rieseguito nel caso di cambiamenti topologici. Poichè i clusterhead coordinano lo scambio di messaggi tra i nodi locali, una misura di efficienza di un algoritmo di clustering è la stabilità dei clusterhead eletti in accordo all'algoritmo. In [40] la politica *Lowest ID* è dimostrata essere la più efficiente in base a misure comparative effettuate mediante tecniche di simulazione.

[1] Questo può accadere se p' riceve da un altro clusterhead, per cui se assumesse tale ruolo verrebbe infranta la regola secondo cui due clusterhead non possono essere connessi.

Con gli algoritmi descritti, la rielezione di un clusterhead viene effettuata ogni volta che la composizione di un cluster cambia. In [22] è proposta la tecnica di *Least Clusterhead Change* (LCC) per aumentare la stabilità dei clusterhead, in accordo alla quale la rielezione viene effettuata solo quando: (i) un nodo si disconnette da qualunque cluster; oppure (ii) due clusterhead con il movimento diventano vicini. La rielezione viene effettuata con uno dei due algoritmi precedentemente descritti.

I risultati di simulazione riportati in [22] dimostrano che il LCC è efficace nel ridurre il numero di riconfigurazioni dell'infrastruttura dei clusterhead. In questo lavoro è stato di conseguenza usato il LCC in combinazione con il *Lowest ID*.

3.3.2 Protocollo di accesso a canale

Nella misura delle prestazioni dei protocolli proposti in questo lavoro è stato utilizzato il protocollo di accesso a canale descritto in [40]. Per ridurre la probabilità di collisione vengono utilizzati codici differenti nei diversi cluster in accordo alla politica CDMA [41, 50]; in particolare, si usano codici ortogonali in cluster adiacenti. All'interno di un cluster tutti i nodi usano lo stesso codice, e la banda è allocata a contesa in accordo alla politica TDMA [40]. Ogni gateway si sintonizza alternativamente sul codice di ogni clusterhead ad esso adiacente. L'unità di tempo del sistema è il *frame*: un frame

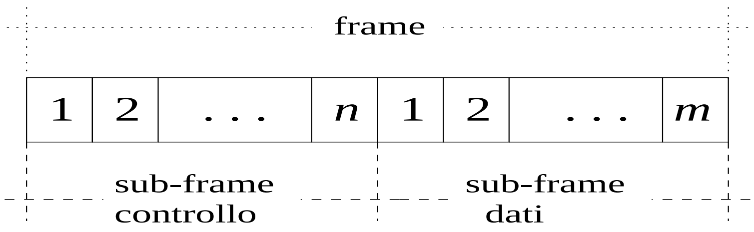


Figura 4: Struttura di un frame.

è costituito da un sub-frame di *controllo* e un sub-frame *dati* (Figura 4). Il sub-frame dati è costituito da m *data slot*, con $m < n$, ognuno di durata sufficiente per consentire la trasmissione di un pacchetto dati. I data slot sono assegnati a contesa ai nodi che desiderano trasmettere. Se due nodi nello stesso cluster trasmettono un pacchetto dati nel medesimo slot, i due pacchetti *collidono* in modo *distruttivo*, ovvero nessuno degli host adiacenti li riceve. In Figura 5, gli host p_1 e p_3 nel cluster C_A trasmettono due pacchetti dati nel medesimo slot; il clusterhead ch_A e la SM p_2 che sono nel raggio di comunicazione di entrambi i trasmettitori non ricevono nessuno dei due pacchetti a causa della collisione. La SM p_4 è vicina solo a p_3 e pertanto ne riceve il pacchetto; analogamente, p_5 riceve il pacchetto di p_1.

Diciamo che si verifica una collisione anche quando un pacchetto viene trasmesso ad un nodo p e (i) nello stesso slot p invia un proprio pacchetto; oppure (ii) p è gateway ed in quello slot è sintonizzato sul codice di un altro dei clusterhead ad esso adiacenti. In entrambi i casi, il pacchetto trasmesso a p non viene da esso ricevuto.

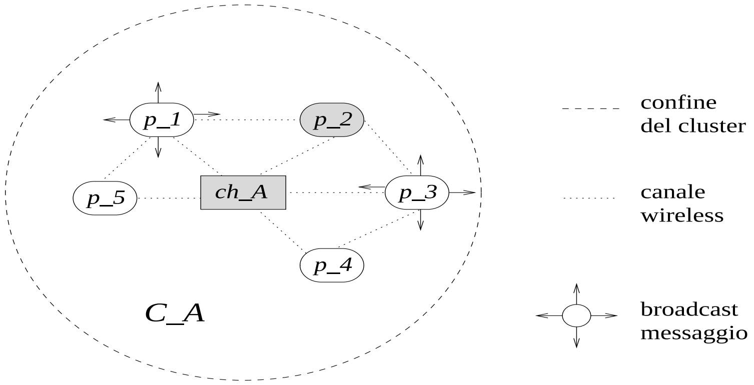


Figura 5: Esempio di collisione tra pacchetti dati.

Il sub-frame di controllo è costituito da *n control slot*, ognuno di durata sufficiente alla trasmissione di un pacchetto di controllo. Ogni SM_i nel sistema manda un pacchetto di controllo ad ogni frame, nell'i-esimo control slot del frame. In accordo a [40], i pacchetti di controllo contengono gli identificatori dei vicini del nodo sorgente, la sua tabella di instradamento, l'identificatore del suo clusterhead, e lo stato dei data slot. Quest'ultima informazione è usata per scoprire se i pacchetti dati trasmessi durante il frame precedente sono stati ricevuti dai vicini destinatari. In caso contrario, il protocollo provvede ad eseguire la ritrasmissione dei pacchetti collisi. A differenza del protocollo descritto in [40], il protocollo MAC usato in questo lavoro fornisce anche un servizio di trasmissione broadcast o multicast inaffidabile; vale a dire, un messaggio può essere indirizzato ad un solo vicino, a tutti o ad un sottoinsieme di essi. Le ritrasmissioni vengono effettuate fino alla ricezione con successo del messaggio da parte di tutti i destinatari vicini al sender.

Si noti che in ogni cluster il numero di control slot è pari al numero di SM nell'intero sistema, benchè in generale il numero di SM all'interno di un cluster sia minore; ciò implica che parte dei control slot sarà inutilizzata con conseguente spreco di banda. D'altra parte questa scelta è forzata dal fatto che i control slot sono preassegnati, mentre la composizione dei cluster varia con la mobilità. Si approfondirà ulteriormente questo argomento nel capitolo 5.

3.3.3 Protocollo per l'instradamento unicast

Per costruire le tabelle di instradamento unicast sopra l'architettura a cluster individuata ai livelli inferiori è possibile usare gli algoritmi di instradamento progettati per Internet, quale il Destination Sequenced Distance Vector (DSDV) [61]. In questo lavoro si è preferito usare l'algoritmo *Clusterhead-Gateway Switch Routing* (CGSR) proposto in [22] per reti multi-hop con struttura a cluster e basato su DSDV. Questo algoritmo individua dei cammini che sono formati da una sequenza di clusterhead e gateway. Ogni nodo mantiene una tabella in cui ricorda per ogni nodo destinatario il rispettivo clusterhead; queste tabelle sono periodicamente scambiate tra nodi vicini e così mantenute aggiornate. Ogni nodo costruisce, usando l'algoritmo DSDV, le tabelle di instradamento per i clusterhead del sistema, ed usa queste informazioni per raggiungere il clusterhead del nodo destinatario di un messaggio. Il destinatario riceve il messaggio dal proprio clusterhead.

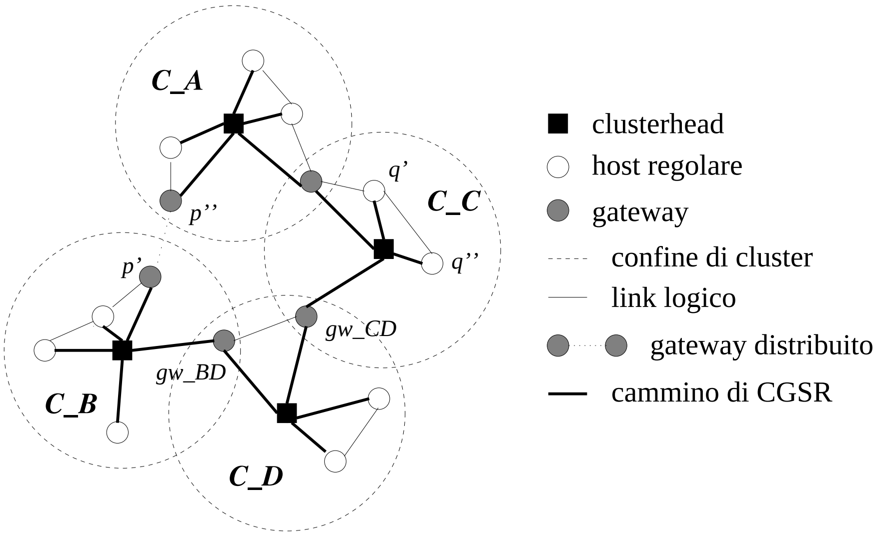


Figura 6: Esempio di instradamento unicast con l'algoritmo CGSR.

La differenza del CGSR rispetto al DSDV è che i cammini individuati evitano le scorciatoie gateway-gateway: ad esempio, in Figura 6 non è possibile usare il link tra gw_{BD} e gw_{CD} per connettere C_B e C_C, ma bisogna percorrere il cammino gw_{CD} - ch_D - gw_{BD}. Questo intuitivamente sembrerebbe diminuire la connettività della rete e far aumentare la lunghezza dei cammini. Osservando ancora la Figura 6, si nota che, poichè l'algoritmo CGSR non considera l'esistenza di gateway distribuiti, i cluster C_A e C_B non sono direttamente collegati (attraverso i nodi p' e p''), ma il cammino che li connette passa attraverso C_C e C_D. I nodi q' e q'' nel cluster C_C non possono comunicare direttamente, nonostante fra essi esista un link logico, ma solo attraverso il comune clusterhead ch_C. Gli esperimenti descritti in [22] dimostrano invece che CGSR si comporta meglio di DSDV perchè elimina la necessità di dover attendere che due gateway si sintonizzino sullo stesso codice prima di far avanzare un messaggio.

3.4 Il simulatore Netsimpro

I protocolli descritti nei prossimi capitoli sono stati implementati sopra un simulatore per reti wireless multi-hop con architettura a cluster, allo scopo di valutarne le prestazioni mediante tecniche di simulazione. Si è usato il simulatore Netsimpro [67] sviluppato presso la *University of California at Los Angeles* (UCLA) in linguaggio *Maisie* [11], che realizza in dettaglio i livelli inferiori dell'architettura di riferimento precedentemente descritta, consentendo così un'accurata analisi del comportamento dei protocolli. Nelle sezioni successive si presentano brevemente il linguaggio Maisie e l'ambiente di simulazione, rimandando la specifica dei parametri e delle condizioni di simulazione alle sezioni in cui sono presentati gli esperimenti condotti con i protocolli proposti in questo lavoro.

Il linguaggio Maisie

Maisie è il linguaggio di simulazione parallelo basato sul C, sviluppato presso la UCLA. Maisie consente di specificare il codice dell'algoritmo che un processo deve eseguire, e di creare multiple istanze del processo (*entità*), ognuna operante su una propria copia delle strutture dati. Il linguaggio è esteso rispetto al C con due classi di istruzioni:

- istruzioni per la gestione dei processi: consentono di creare un'entità, di distruggerla, di sospendere l'esecuzione di un'entità fino alla ricezione di un messaggio o per un certo tempo di simulazione;
- istruzioni per la comunicazione fra entità: consentono lo scambio di messaggi, di struttura specificata dal programmatore, tra entità, e permettono di decidere il ritardo (in termini di tempo di simulazione) di consegna di un messaggio all'entità destinataria.

Il supporto a run-time del linguaggio gestisce la concorrenza fra le entità e l'avanzamento del tempo di simulazione. Il tempo di simulazione avanza di una unità quando ogni *evento* (consistente nell'esecuzione di una porzione di codice di un'entità) in lista per il tempo corrente è stato eseguito. Il tempo è globale nel sistema, ovvero tutte le entità sono sincronizzate su di esso.

Struttura del simulatore

In Figura 7 è mostrato uno schema della struttura del simulatore. L'entità DRIVER genera tutte le altre entità e fa partire la simulazione, al termine della quale scarica su file i risultati ottenuti. L'entità SYNCHRONIZER gestisce il tempo di simulazione; nel

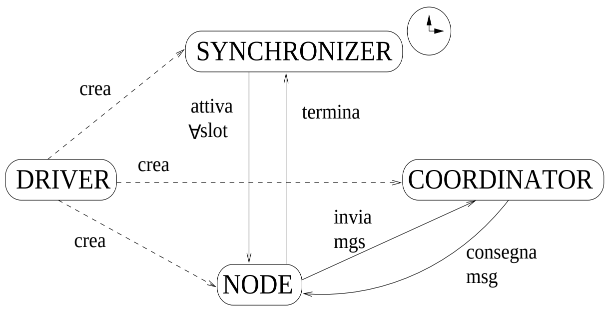


Figura 7: Struttura del simulatore NETSIMPRO.

nostro caso, l'unità di tempo corrisponde ad un frame. L'entità NODE include il codice dei protocolli dei livelli bassi dell'architettura di riferimento; vengono create tante entità NODE quante sono le SM nel sistema simulato. Ad ogni slot di controllo SYNCHRONIZER attiva l'entità NODE a cui compete quello slot, per l'invio del pacchetto di controllo. La struttura del messaggio scambiato tra NODE e COORDINATOR negli slot di controllo contiene tutte le informazioni di controllo stabilite dall'algoritmo MAC descritto in [40] e in sezione 3.3.2, permettendo così di realizzare gli algoritmi di clustering e di instradamento unicast precedentemente illustrati. L'entità COORDINATOR provvede a simulare lo scambio di pacchetti tra le SM, risvegliando le entità NODE destinatarie di ogni pacchetto e passando loro il pacchetto. Al termine degli slot di controllo, le medesime interazioni tra SYNCHRONIZER, COORDINATOR e NODE sono ripetute per ogni slot dati. Nel caso dei pacchetti dati, COORDINATOR si occupa anche di rilevare l'occorrenza di collisioni e di conseguenza consegna solo i pacchetti non collisi ai rispettivi NODE destinatari. Al termine degli slot dati, SYNCHRONIZER fa avanzare il tempo di simulazione al frame successivo. Altre entità sono incluse nel simulatore, che hanno il compito di generare il movimento dei nodi e i pacchetti del traffico dati.

Il protocollo di Reliable Broadcast descritto nel capitolo 4 è incluso in una nuova entità del simulatore che opera in stretta relazione con l'entità NODE. Al contrario, il codice del protocollo per l'instradamento multicast descritto nel capitolo 5 è stato inserito direttamente nell'entità NODE. Alcune caratteristiche dell'ambiente simulato sono comuni ad entrambi i casi. Si è assunto un modello di movimento per cui una SM che raggiunge il perimetro dell'area considerata viene riflessa. Due SM possono comunicare tra loro se sono a distanza al più pari al raggio di un cluster; per semplicità di caratterizzazione del sistema e per evitare un'eccessiva durata delle simulazioni, non è stato considerato il caso in cui un pacchetto possa essere distrutto a causa di rumore o attenuazione dovuta alle caratteristiche del terreno. I pacchetti vengono persi solo in caso di collisione; si assume che le collisioni siano distruttive. Una collisione si verifica quando due SM vicine mandano un pacchetto nello stesso slot dati o quando un clusterhead invia un pacchetto ad un gateway che è in ascolto sul codice di un altro clusterhead. Per superare collisioni e fallimenti di link, il protocollo MAC esegue ritrasmissioni. Si assume che la larghezza di banda sia di 2 Mbps. Ipotizziamo che la taglia di un pacchetto di controllo sia 3K e quella di un pacchetto dati sia 6K; di conseguenza, i control slot avranno durata pari a 1.5 msec. e i data slot a 3 msec.

3.5 Conclusioni

In questo capitolo si sono discussi in dettaglio l'architettura di riferimento che verrà presa in considerazione nel seguito del lavoro, e il funzionamento dei protocolli di basso livello che ne fanno parte. Questo permette di definire precisamente quali servizi siano a disposizione dei protocolli proposti in questa tesi. Si è data una rapida descrizione del pacchetto di simulazione che è stato usato per implementare gli algoritmi proposti in questa tesi allo scopo di verificarne il funzionamento e valutarne le prestazioni; tale pacchetto implementa i suddetti protocolli di basso livello.

Capitolo 4

Wireless Reliable Broadcast

4.1 Introduzione

In questo capitolo si presenta un protocollo originale per la soluzione del problema del Reliable Broadcast in reti wireless multi-hop. Il modello di sistema considerato è quello descritto nel capitolo 1; il problema che ci si propone di risolvere è formalmente definito nella sezione 2.2.

Come si è visto in sezione 2.5, in letteratura sono stati presentati recentemente alcuni protocolli per la consegna affidabile di messaggi in sistemi single-hop, che sfruttano l'esistenza di stazioni fisse [1, 2, 27]. Il protocollo proposto in questo capitolo è stato progettato per l'uso in reti multi-hop; esso può operare direttamente sopra il livello MAC, in quanto non necessita di alcun servizio di instradamento. Il servizio che il protocollo fornisce garantisce che *tutte* le SM nella rete *consegnino* lo stesso insieme di messaggi al livello superiore dell'architettura. Lo scopo di tale forte requisito è quello di garantire l'affidabilità e la tolleranza ai fallimenti a dispetto della mobilità, che può influenzare negativamente la possibilità di mantenere strutture di instradamento basate sulla conoscenza (totale o parziale) della topologia della rete.

Si considera un sistema in cui la frequenza dei cambiamenti della topologia è difficile da prevedere e può variare in un ampio intervallo. In questo contesto, il protocollo presentato si propone come un tentativo di ottenere il servizio di Reliable Broadcast come una primitiva di basso livello nello stack di protocolli dell'architettura, con un meccanismo più affidabile ed efficiente del flooding senza la necessità di fare assunzioni sulla mobilità. L'importanza di avere un servizio di broadcast di basso livello con la garanzia di consegna affidabile può avere diverse motivazioni:

1. riduce drasticamente la difficoltà di ottenere un servizio multicast ad alto livello con più forti garanzie sull'ordine di consegna dei messaggi (ad esempio, causale o totale [44]), indipendentemente dal fatto che tali servizi siano forniti dalla pila di protocolli o a livello applicazione;

2. è in linea con lo *end-to-end argument*, in accordo al quale ogni livello dell'architettura deve fornire solo i servizi fondamentali per quel livello (in accordo alle sue funzionalità descritte dal modello ISO/OSI [68]), e demandare ai livelli superiori la realizzazione di servizi più complessi, le cui caratteristiche dipendono dalla loro semantica [37];

3. può beneficiare dei servizi del protocollo MAC per ottenere migliori prestazioni, controllare la crescita della dimensione dei messaggi e permettere l'implementazione di una pila integrata di protocolli per reti ad-hoc;

4. può rappresentare un'alternativa al flooding e facilitare la costruzione di altri servizi di base, quale ad esempio un protocollo di instradamento basato sul *link state*, come quello usato in [62].

Il protocollo descritto in questo capitolo è costituito da due fasi: la fase di *diffusione*, durante la quale il messaggio è distribuito a tutte le destinazioni, e la fase di *raccolta*, nella quale vengono ottenuti gli acknowledgement (*ack*) da tutti i destinatari. La struttura di instradamento sfruttata (il *forwarding tree*, FT) è costruita durante la fase di diffusione e non richiede alcun meccanismo per la manutenzione dei cammini. La durata di vita di questa struttura è pari al massimo al tempo necessario per distribuire un messaggio broadcast e raccogliere gli acknowledgement. Per garantire la consegna affidabile è stato adattato alle reti multi-hop il medesimo meccanismo di recupero impiegato in [1, 2] per superare lo smarrimento di un pacchetto. Il meccanismo di acknowledgement è tale per cui, in presenza di mobilità, il protocollo interviene rapidamente nel luogo in cui si è verificato il cambiamento di topologia, evitando quando possibile di coinvolgere più ampie regioni della rete e risparmiando quindi risorse. L'uso di un meccanismo distribuito per superare i fallimenti e del FT permette di limitare il numero di messaggi generati. Quando la frequenza di cambiamenti della topologia è molto elevata così da distruggere anzitempo il FT, il protocollo ricorre al flooding (se non è disponibile un protocollo di instradamento unicast), limitatamente alle zone in cui l'albero è stato danneggiato. In tal modo il protocollo è in grado di adattarsi autonomamente alle condizioni incontrate in diverse regioni della rete.

Il capitolo è organizzato come segue: nella prossima sezione introduciamo le assunzioni sul sistema che garantiscono la correttezza del protocollo rispetto alla definizione del problema data. Quindi descriviamo in dettaglio il funzionamento del protocollo e ne dimostriamo la correttezza. In sezione 4.5 si analizzano le prestazioni del protocollo valutate sia mediante tecniche analitiche sia mediante tecniche di simulazione[1]. Il contenuto di questo capitolo è tratto da [60, 58]. In Appendice A sono riportati

[1] L'ambiente di simulazione usato è descritto in sezione 3.4.

gli automi che descrivono il funzionamento del protocollo per i diversi ruoli delle SM: clusterhead, gateway o host regolare.

4.2 Assunzioni sul sistema

Si considera un sistema che si comporta in accordo al modello di fallimento descritto dalla definizione 2.4.1, con l'assunzione aggiuntiva che tutti i malfunzionamenti sono temporanei, benchè non sia specificato un limite superiore alla loro durata. Analogamente, le disconnessioni delle stazioni e le partizioni della rete, come nel caso del cluster C_E in Figura 2, sono tollerate sotto l'ipotesi che vengano prima o poi superate. Queste assunzioni sono adatte per modellare la mobilità delle SM e i fallimenti transienti dei canali wireless, e si basano su una precisa scelta progettuale volta a limitare la complessità del protocollo a questo livello dell'architettura. Infatti, grazie a queste ipotesi non è necessario adottare un meccanismo per la gestione della *membership* [7], che permetta ad un insieme di SM di raggiungere un accordo riguardo alla composizione del sistema che tenga conto dei cambiamenti che si possono verificare nel tempo. Ad esempio, la composizione del sistema può cambiare se una SM fallisce crash o, nel caso del problema del *multicast*, se una SM volontariamente entra o esce dal gruppo delle destinazioni. In conseguenza delle nostre assunzioni, la membership può cambiare solo nel secondo caso, che non consideriamo in questo capitolo, rendendo perciò il problema della membership risolubile[2]. Se necessario, la gestione dei cambiamenti volontari e l'identificazione dei processori falliti può essere eseguita ai livelli superiori dell'architettura, in accordo ai requisiti delle applicazioni.

Come conseguenza delle assunzioni ora discusse, fallimenti, mobilità e disconnessioni delle SM possono ritardare la terminazione del protocollo. Per garantire che la sua esecuzione non abbia una durata indefinita, il sistema deve soddisfare le seguenti *assunzioni sulla mobilità*:

Proprietà 4.2.1 [Liveness] *Ogni SM p è libera di muoversi arbitrariamente nel sistema e di essere temporaneamente disconnessa. Ma prima o poi p si connette ad un clusterhead ch e resta connessa ad esso per il tempo sufficiente a scambiare informazioni di stato con ch; inoltre, se esistono dei messaggi broadcast che uno tra p e ch ha ricevuto e l'altro no, p resta connesso per il tempo sufficiente perchè almeno uno di tali messaggi sia scambiato con successo[3] prima del successivo cambiamento di topologia.*

[2] In [36] si dimostra l'impossibilità di risolvere il problema della membership in sistemi asincroni in cui può verificarsi anche un solo fallimento crash. Non è chiaro se e come questo risultato sia applicabile ai sistemi wireless, in cui mobilità e disconnessioni introducono un fattore di asincronismo.

[3] Per "scambiato con successo" intendiamo che la SM che invia il messaggio (p o ch) riceve un ack per esso.

Proprietà 4.2.2 [Stabilità dei clusterhead] *Prima o poi esiste una SM che assume il ruolo di clusterhead per il tempo necessario a garantire che lo scambio di informazioni di stato con gli altri clusterhead sia completata con successo e, se ci sono messaggi broadcast diffusi parzialmente, che almeno uno di questi messaggi sia scambiato con successo tra i clusterhead.*

Le due proprietà precedenti non specificano nè vincoli sulla mobilità nè limiti superiori alla durata delle disconnessioni delle SM. Simili vincoli non sono necessari per garantire la terminazione eventuale del protocollo. Essi potrebbero essere imposti dalle applicazioni sovrastanti in accordo ai propri requisiti; in tal caso, la gestione delle eccezioni sarà effettuata al relativo livello dell'architettura.

Se la proprietà di *liveness* non fosse soddisfatta, una SM potrebbe non ricevere mai un messaggio m, oppure riuscire a ricevere m ma non riuscire mai a notificarne la ricezione. Se la proprietà di *stabilità dei clusterhead* non fosse soddisfatta, i messaggi potrebbero non essere mai diffusi con successo tra clusterhead differenti. In entrambi i casi il protocollo non potrebbe terminare.

4.3 Descrizione del protocollo

In questa sezione si descrive nel dettaglio il funzionamento del protocollo originale per il Reliable Broadcast in reti wireless multi-hop (protocollo WRBcast). Si iniziano con il definire le primitive di servizio e la struttura di instradamento adottata. Si illustra come il protocollo opera nel caso di topologia statica con canali affetti da fallimenti transienti. Quindi si discutono i meccanismi adoperati per affrontare i problemi dovuti alla mobilità delle SM e ai cambiamenti dell'insieme dei clusterhead. Infine, si mostra la struttura dei messaggi scambiati dal protocollo.

Nel seguito si assume che ogni messaggio broadcast sia univocamente identificato dalla coppia ⟨`ID sender, #sequenza`⟩, dove `ID sender` è l'identificatore della SM che genera il messaggio e `#sequenza` è un numero di sequenza progressivo da essa apposto. Nella descrizione del protocollo si assumerà che vi sia un solo messaggio da diffondere; nel caso della diffusione di più messaggi vengono eseguite parallelamente più istanze del protocollo, una per ogni messaggio.

Si dice che un messaggio è *stabile* quando è stato ricevuto da tutte le destinazioni. Quando un messaggio è stabile, l'istanza di protocollo per la sua diffusione affidabile può terminare e tutte le risorse di banda e memoria delle SM impiegate da essa possono essere rilasciate.

4.3.1 Primitive di servizio

L'interfaccia tra le applicazioni e il protocollo WRBCAST è costituita dalle primitive `WRBCAST.Rq`(msg, $dest\_ID$) e `DELIVER.Ind`(msg). Le primitive sono *non bloccanti*, perciò le entità di livello superiore possono procedere in parallelo con il protocollo WRBCAST. Quando viene invocata la primitiva `WRBCAST.Rq`, l'entità chiamante si blocca solo fino a quando non sa che il messaggio msg è stato correttamente ricevuto dal suo clusterhead[4]. Quando l'entità chiamante si sblocca, sa che msg sarà consegnato a tutte le SM indicate in $dest\_ID$. I valori che $dest\_ID$ può assumere sono quelli descritti in sezione 3.2; in questo capitolo considereremo però solo il caso di messaggi broadcast ($dest\_ID =$`any`). La primitiva `DELIVER.Ind` notifica all'entità di livello superiore la ricezione del messaggio msg; quando l'entità riceve la notifica sa che msg sarà prima o poi consegnato a tutte le altre destinazioni.

4.3.2 Schema del protocollo

Il protocollo WRBCAST è costituito da due fasi: la fase di *diffusione*, durante la quale il messaggio è distribuito dalla sorgente a tutte le destinazioni, e la fase di *raccolta*, durante la quale la sorgente colleziona gli acknowledgement (ack, in breve) inviati in risposta dalle destinazioni.

Gli ack rappresentano un alto costo da pagare, ma sono inevitabili. Infatti, si potrebbe in alternativa impiegare uno schema a *negative acknowledgement*, come in [6], ma solo sotto l'assunzione di una continua generazione di messaggi broadcast o dell'esistenza di qualche forma di sincronizzazione tra sorgenti e destinatari dei messaggi, in modo che questi ultimi conoscano quando un messaggio dovrebbe essere generato e possano accorgersi della sua mancata ricezione. Queste assunzioni non sono necessariamente soddisfatte da tutte le applicazioni, e sono di difficile applicazione in ambiente mobile. Al contrario, lo schema da noi scelto risulta di utilizzo generale. Inoltre l'uso di uno schema a negative ack rende difficile determinare quando un messaggio sia stabile, e perciò quando la relativa istanza di protocollo possa terminare. Uno schema alternativo è quello adottato dal protocollo *Scalable Reliable Multicast* (SRM) proposto in [37] per reti fisse, in cui le destinazioni periodicamente si scambiano informazioni riguardo ai messaggi consegnati: una destinazione che nota la mancata ricezione di un messaggio m ha la responsabilità di risolvere il fallimento inviando broadcast una richiesta per m; una stazione che ha m e riceve la richiesta invia m broadcast, così da risolvere contemporaneamente eventuali altri fallimenti che hanno interessato la diffusione di m. In [37] sono proposte ottimizzazioni, basate

[4] Se msg è generato da un utente che opera su una SM correntemente avente il ruolo di clusterhead, il controllo ritorna immediatamente all'entità chiamante.

sulla conoscenza della topologia della rete, per ridurre il numero di messaggi di richiesta e risposta ridondanti. In ambiente mobile questo approccio appare difficilmente adottabile; infatti:

1. il periodico scambio di informazioni di stato tra le destinazioni richiederebbe un grande consumo di banda;

2. il tasso di fallimento più alto che nelle reti fisse provocherebbe un più frequente ricorso al meccanismo di recovery descritto;

3. la topologia della rete varia dinamicamente ed è difficile poterla conoscere, perciò non sarebbe applicabile alcuna ottimizzazione.

Lo schema adottato da WRBCAST consente di rilevare rapidamente un fallimento, e risolverlo localmente alla zona in cui si è verificato evitando quanto più possibile di coinvolgere altre regioni della rete. Per raggiungere questo obiettivo, ogni clusterhead è reso responsabile della soluzione dei fallimenti che si verificano nel suo cluster. Inoltre, il numero di messaggi di ack generati è mantenuto limitato grazie allo sfruttamento dell'infrastruttura di instradamento individuata durante la fase di diffusione, che verrà descritta nella prossima sezione.

4.3.3 Il *Forwarding Tree*

Il *forwarding tree* (FT) è un albero di instradamento centrato sulla sorgente, che è costruito *on demand* quando la sorgente deve inviare un messaggio broadcast m, durante la fase di diffusione di m, e non richiede di essere mantenuto a fronte della mobilità delle SM. La sua durata è al massimo pari al tempo necessario per completare l'esecuzione dell'istanza del protocollo per m.

Il FT è costruito connettendo clusterhead adiacenti attraverso i gateway che li congiungono; la radice dell'albero è il clusterhead della SM che genera il messaggio broadcast. I cammini sull'albero sono i cammini più rapidi (non necessariamente i più brevi) che connettono la radice alle destinazioni al momento dell'esecuzione della fase di diffusione.

Il principale compito del FT durante la fase di diffusione è quello di fornire dei cammini di consegna efficienti e privi di cicli. Esso è però maggiormente utile nella fase di raccolta, per instradare gli ack dalle destinazioni verso la sorgente del messaggio e limitarne il numero. Il FT può essere distrutto prima del termine dell'esecuzione dell'istanza del protocollo a causa della mobilità, di disconnessioni o fallimenti. Se ciò avviene, il protocollo non intraprende alcuna azione per ricostruire l'albero: limitatamente alle zone in cui il FT è danneggiato, il protocollo ricorre al *flooding* per inviare gli ack. Non appena un ack che è stato così inviato è ricevuto da una stazione che è ancora sull'albero, essa interrompe il flood e riprende il modo normale di raccolta degli ack.

Ogni clusterhead fa parte del FT; per questa ragione è utilizzabile principalmente per la diffusione di messaggi broadcast o per il multicast a gruppi *densi*, ovvero aventi almeno un membro del gruppo in quasi tutti i cluster. Nel caso di gruppi *sparsi* è probabilmente più conveniente ricorrere a strutture di instradamento più specializzate, quali quelle proposte in [22, 23, 28] o quella proposta nel prossimo capitolo.

4.3.4 Le fasi di diffusione e di raccolta

In Figura 8(a) si mostra un esempio del funzionamento del protocollo, nel caso delle topologia di rete mostrata in Figura 2, e per un messaggio broadcast m generato dalla SM p_s che si trova nel cluster C_A. In questa sezione si ipotizza che la topologia sia *statica*, benchè possano verificarsi fallimenti transienti dei canali di comunicazione; i problemi derivanti dai cambiamenti di topologia sono analizzati in sezioni successive.

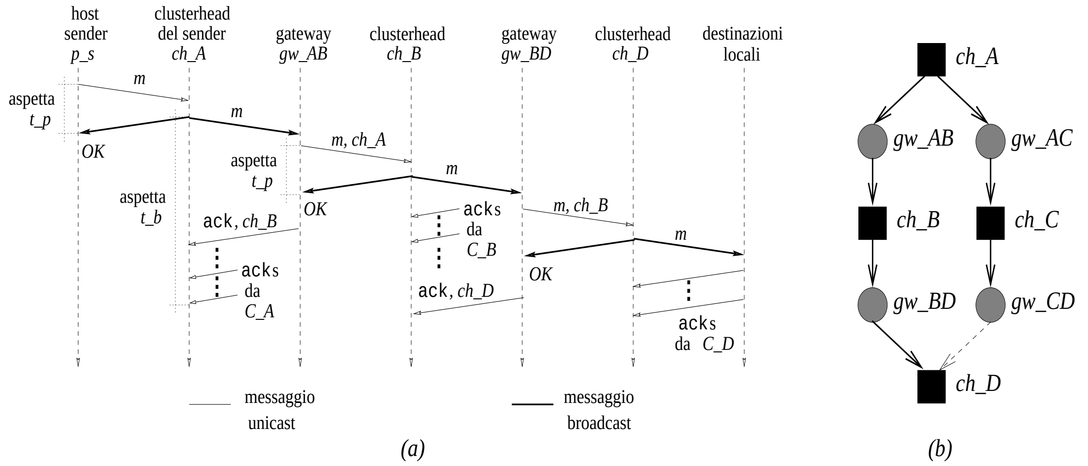


Figura 8: (a) Esempio di esecuzione del protocollo nel caso di topologia statica. (b) Il corrispondente FT costruito.

Esempio 4.3.1 La stazione p_s invia m al proprio clusterhead ch_A, che d'ora in avanti indicheremo come il clusterhead *originatore* di m, che sarà la radice del FT costruito per la diffusione di m. L'originatore aggiunge a m il proprio header, e ne fa il broadcast all'interno del proprio cluster, restando poi in attesa di ricevere un ack da ogni SM in C_A. Quando p_s riceve il broadcast di ch_A sa di avere raggiunto con successo il proprio clusterhead; a questo punto la primitiva `WRBCAST.Rq` può rendere il controllo all'entità chiamante. Ogni host regolare p_d in C_A alla ricezione di m invia il proprio ack a ch_A. Ogni gateway in C_A attende di inviare il proprio ack fino a quando non ha raggiunto l'obiettivo di propagare m nei cluster adiacenti. A questo scopo, gw_{AB} invia m a ch_B, insieme all'indicazione che esso ha ricevuto m da ch_A; analogamente opera gw_{AC} con ch_C. Quando ch_B riceve m, ricorda che esso proviene da ch_A e lo sceglie come proprio genitore nel FT per m; quindi fa il broadcast di m nel proprio cluster indicando tale scelta. Se ch_B riceve multiple copie di m attraverso gateway differenti, esso sceglie un solo clusterhead come proprio padre e non modifica la propria scelta successivamente. La pubblicazione di questa informazione è utile anche per evitare la formazione di loop dovuta alla diffusione del messaggio lungo cammini multipli tra una coppia di nodi, e per l'eliminazione dei duplicati. Infatti:

- ogni gateway in C_B che ha ricevuto m da un clusterhead differente da ch_B impara che ch_B è già in possesso di m. Perciò può evitare di propagare m a ch_B e, se non è adiacente ad altri clusterhead, può mandare il proprio ack al clusterhead da cui ha ricevuto m;

- gw_{AB}, per quanto detto al punto precedente, può mandare il proprio ack a ch_A. Inoltre, gw_{AB} segnala a ch_A che ha ch_B come discendente nel FT per m.

Quando ch_A ha ricevuto gli ack da tutte le stazioni in C_A (gateway compresi), sa che m è *localmente stabile*, e che i clusterhead adiacenti provvederanno alla diffusione di m nel resto del sistema. Il procedimento descritto è ripetuto negli altri cluster. Al termine, ogni clusterhead ha ricevuto il messaggio; ch_B sa di avere come discendente ch_D mentre ch_C sa di essere un clusterhead foglia nell'albero (Figura 8(b)).

Al termine della fase di diffusione per m ogni clusterhead conosce il proprio genitore ed i propri eventuali discendenti nel FT per m; queste sono le uniche informazioni mantenute riguardo al FT. Il fatto che i gateway non ricordino la struttura dell'albero contribuisce a renderlo più robusto rispetto alla mobilità: finchè esiste un gateway tra un clusterhead ed il proprio predecessore, il FT può essere usato. Ogni nodo ricorda inoltre quali sono i messaggi ricevuti, allo scopo di eliminare i duplicati.

La fase di raccolta è iniziata dai clusterhead foglia nel FT. Quando ch_D ha ricevuto gli ack per m di tutte le SM in C_D, li riunisce in un unico messaggio di `cluster-ack` che invia al proprio genitore ch_B. Analogamente opera ch_C. Al contrario, ch_B, che è interno all'albero, prima di inviare il proprio `cluster-ack` attende non solo di avere tutti gli ack locali, ma anche di ricevere i `cluster-ack` di tutti i propri discendenti. Quando ciò avviene, ch_B ingloba le informazioni locali e quelle ottenute dai discendenti in un unico messaggio di `cluster-ack`, che invia a propria volta al proprio genitore. Il procedimento è ripetuto fino al raggiungimento del clusterhead originatore di m. In tal modo, se il FT non è danneggiato ed i ritardi non sono elevati, per ogni ramo del FT viaggia un solo messaggio di `cluster-ack`, risparmiando in tal modo banda ed evitando la collisione di un gran numero di ack all'originatore.

I *fallimenti transienti* dei canali wireless possono provocare lo smarrimento sia di m, sia dei messaggi di ack e di `cluster-ack`; i primi due casi sono trattati nella prossima sezione. Per evitare che un clusterhead non foglia attenda indefinitamente il `cluster-ack` di uno o più dei suoi discendenti, ogni clusterhead interno a FT dopo aver raccolto tutti i propri ack locali attende per al più un tempo T_{ack}, la cui durata è pari approssimativamente a due volte il *round-trip-delay* su un link logico. Dopo questo tempo, il clusterhead procede autonomamente all'invio del proprio `cluster-ack`; questo può far aumentare il numero di messaggi di `cluster-ack` che attraversano un ramo dell'albero.

Se al contrario il `cluster-ack` di un discendente è ricevuto da un clusterhead interno prima che quest'ultimo abbia raccolto tutti gli ack locali, esso viene mantenuto temporaneamente in attesa in una coda di *hold-back* così da aumentare la probabilità di poter fondere in un solo messaggio di acknowledgement tutti gli ack. Questa coda viene svuotata periodicamente, con un periodo pari a T_{ack}, per evitare l'indefinito ritardo dei messaggi a causa della mobilità.

4.3.5 Stabilità dei messaggi

In questa sezione si descrive l'uso del FT per garantire la consegna affidabile dei messaggi, e si presentano le strutture dati principali utilizzate dall'algoritmo.

Seguendo i passi del protocollo descritti in precedenza e facendo riferimento all'albero mostrato in Figura 8(b), quando ch_A riceve gli ack da tutte le destinazioni di m, esso impara che m è *stabile*. La stabilità di ogni messaggio deve essere raggiunta nonostante i possibili fallimenti di link e la mobilità delle SM.

Il protocollo adotta una politica di ritrasmissione per risolvere i fallimenti transienti dei link, che possono provocare lo smarrimento dei messaggi broadcast e dei `cluster-ack`. Si ipotizza che sia sufficiente un numero finito di ritrasmissioni per superare un fallimento transiente. Le ritrasmissioni sono iniziate solo dai clusterhead, alla scadenza di un timeout T_b la cui durata è approssimativamente pari a due volte il *round-trip-delay* su un link logico. Questo valore è scelto in modo tale che le ritrasmissioni non siano troppo frequenti: esso corrisponde al tempo necessario perchè un messaggio sia propagato tra due cluster adiacenti (ovvero, tra i loro clusterhead), e il corrispondente ack torni indietro. Ogni clusterhead ha la responsabilità di provvedere alla soluzione dello smarrimento dei pacchetti nella porzione di rete che è sotto il suo controllo, ovvero il suo cluster e i rami verso il suo genitore e i suoi discendenti nel FT. Questo approccio permette di ottenere la distribuzione del carico di lavoro necessario per superare i fallimenti, e di evitare perciò di far partire le ritrasmissioni dei messaggi dal clusterhead originatore. I benefici così ottenuti in termini di banda e di latenza sono bilanciati da un costo di memorizzazione ad ogni SM, poichè esse devono mantenere una copia di ogni messaggio broadcast m fino a quando la stabilità di m non diventa nota. A questo scopo, ogni SM mantiene due strutture dati: *stable* e

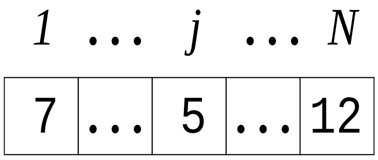


Figura 9: Esempio di array $stable_p$.

unstable. Nella prima (Figura 9) si memorizzano le identità dei messaggi globalmente stabili. Essa è costituita da un array di n elementi, tale che tutti i messaggi generati da p_j con numero di sequenza minore o uguale al valore dell'elemento j-esimo sono stabili. Ad esempio, in Figura 9, i messaggi generati da p_j con numero di sequenza compreso tra 0 e 5 sono stabili. Nella seconda (Figura 10) ogni messaggio broadcast ricevuto è mantenuto nella coda relativa alla SM che l'ha originato; le code sono mantenute ordinate in accordo ai numeri di sequenza dei messaggi. Ad esempio, una SM avente la struttura *unstable* mostrata in Figura 10 ha ricevuto da p_1 i messaggi m' e m'' di cui ancora non conosce la stabilità globale, e tali che m' ha numero di sequenza inferiore a quello di m''. Insieme ad ogni messaggio è memorizzato un *descrittore*, il significato dei cui campi è illustrato dal seguente esempio.

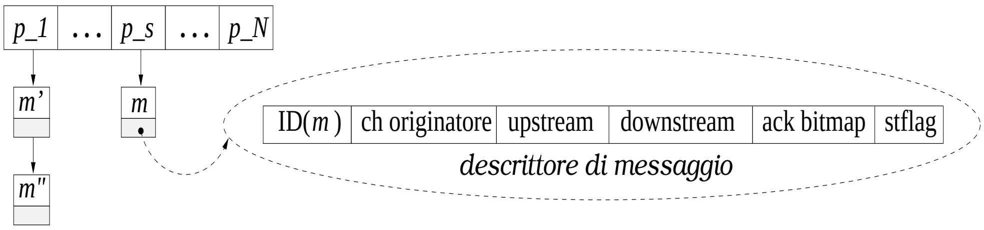


Figura 10: Esempio di struttura $unstable_{ch}$ e di descrittore di messaggio.

Esempio 4.3.2 Si consideri il sistema mostrato in Figura 8: il clusterhead ch_B durante la fase di diffusione del messaggio m aggiorna la propria *unstable* inserendovi m e crea per esso un descrittore. In esso ricorda l'identificatore del messaggio ($\mathtt{ID}(m)=\langle p_s,\ \#seq\rangle$), l'identificatore del clusterhead originatore ch_A, gli identificatori del proprio genitore (ch_A) e dei propri eventuali discendenti (ch_D) nel FT costruito per la diffusione di m, le identità delle SM nel proprio cluster da cui ha ricevuto un ack in risposta all'invio di m (`ack bitmap`), e lo stato di stabilità globale di m (`stflag`=FALSE inizialmente).

Se si verificano dei fallimenti nel proprio cluster, un clusterhead continua a ritrasmettere fino a quando non ha ricevuto un ack da ogni SM nel suo cluster (gateway compresi), ovvero ha completato la `ack bitmap` per il messaggio[5]. I duplicati che possono essere generati in conseguenza delle ritrasmissioni vengono individuati sfruttando le informazioni memorizzate nelle due strutture descritte; i duplicati sono eliminati ma un ack per essi è comunque generato per evitare ulteriori ritrasmissioni.

Le informazioni contenute nella `ack bitmap` sono usate per costruire il `cluster-ack` per il corrispondente messaggio. Per garantire la propagazione del `cluster-ack`, si adotta per il suo invio la medesima politica di trasmissione dei messaggi broadcast: un clusterhead che riceve il `cluster-ack` da un proprio discendente nel FT del messaggio a cui il `cluster-ack` si riferisce, invia al discendente un ack in risposta. Se l'ack non è ricevuto, è possibile che il FT non sia più connesso; nella prossima sezione si prende in esame questa eventualità.

Quando un clusterhead originatore di un messaggio m riceve un ack (direttamente o riportato da un `cluster-ack`) da ogni destinatario di m, sa che m è *globalmente stabile* nel sistema. Se tutti i messaggi generati dalla stessa SM che ha generato m, aventi numero di sequenza minore di quello di m, sono globalmente stabili, l'originatore può rimuovere le informazioni riguardanti m dalla propria *unstable* e aggiornare consistentemente la propria *stable*. Per permettere a tutte le altre SM di fare altrettanto, l'informazione riguardo alla stabilità di m deve essere diffusa in modo affidabile. Per far questo, quando il contenuto della struttura *stable* di un clusterhead cambia, essa è inviata insieme al primo messaggio broadcast per cui il clusterhead è originatore. Se non esistono messaggi da mandare per un certo periodo, la *stable* viene diffusa come un messaggio broadcast. In entrambi i casi, ogni SM prima o poi riceve il contenuto aggiornato della *stable* di ogni clusterhead, e aggiorna le proprie strutture dati in accordo: fonde la propria *stable* con quella ricevuta, prendendo per ogni elemento dell'array il massimo tra le due, e modifica consistentemente il contenuto della propria *unstable*.

Si osservi che non necessariamente un messaggio m che diventa globalmente stabile è immediatamente cancellato dalla *unstable*; può infatti esistere un messaggio m' generato dalla stessa SM con numero di sequenza inferiore a m e non ancora stabile. In tal caso, m viene ricordato come stabile nella *unstable* (`stflag=`TRUE in Figura 10) fino a quando non sono soddisfatte le condizioni per la sua rimozione. Ritardarne la cancellazione rappresenta un piccolo costo aggiuntivo in termini di occupazione di memoria, che è compensato dalla possibilità di mantenere una struttura molto semplice per la *stable*.

[5]La raccolta degli ack e la politica di ritrasmissione possono essere ottimizzate in funzione del particolare protocollo MAC adottato.

4.3.6 Mobilità delle SM

In questa sezione si descrive il comportamento del protocollo in presenza della mobilità delle SM che però non provoca cambiamenti nell'infrastruttura dei clusterhead. Quando una SM p si sposta dal cluster C_A al cluster C_B, si possono verificare i seguenti eventi:

1. p è sorgente di un messaggio m che è ancora instabile quando p si sposta;

2. p è destinatario di un messaggio m' e ch_A non ha ricevuto da p un ack per m'.

Nel caso (1) p non sa se ch_A ha ricevuto m e ne eseguirà pertanto la diffusione; nel caso (2) ch_A non sa se p ha ricevuto m' prima di uscire dal cluster. Entrambi i casi sono risolti attraverso lo scambio di informazioni di stato tra p e ch_B. Non appena p entra in un nuovo cluster, in questo esempio C_B, p e ch_B si scambiano i vettori *stable* e gli identificatori dei messaggi che ognuno di essi conserva nella propria *unstable*. Se esistono dei messaggi che uno solo tra p e ch_B ha, esso li invia all'altra SM. In questo modo, p può far ripartire la diffusione di m (caso (1)), mentre ch_B può aggiornare le proprie `ack-bitmap` ricordando quali messaggi p possiede (caso (2)). Se in conseguenza di ciò ch_B raccoglie nuove informazioni riguardo a un messaggio broadcast per cui ha già mandato il proprio `cluster-ack`, dovrà mandare un altro `cluster-ack` all'originatore di tale messaggio per favorire la sua stabilizzazione globale.

Riguardo al caso (1), in seguito allo scambio di informazioni di stato può accadere che sia ch_A sia ch_B inizino la diffusione di m in qualità di originatori. Questa situazione provoca la generazione di duplicati di m, che vengono rilevati ed eliminati come abbiamo visto in precedenza, e per i quali vengono inviati ack ai due clusterhead in conflitto. Per ridurre la quantità di messaggi in circolazione e identificare una sola SM responsabile della raccolta degli ack, si è scelto di identificare come unico clusterhead originatore il clusterhead che correntemente ha la SM sorgente del messaggio (p nell'esempio) nel proprio cluster. Tale clusterhead diviene radice del FT per il messaggio e raccoglie gli ack; altri eventuali clusterhead originatori che osservano una nuova diffusione del messaggio da parte di un clusterhead differente e non possiedono più la SM sorgente nel proprio cluster inviano al nuovo originatore gli ack fino ad allora raccolti.

4.3.7 Cambiamento dei clusterhead

I cambiamenti di topologia possono provocare l'elezione di un nuovo clusterhead. Quando una SM perde il ruolo di clusterhead, tutte le informazioni in suo possesso riguardo ai messaggi da diffondere e ai messaggi stabili sono perse e devono essere ricostruite dal nuovo clusterhead. Consideriamo la situazione seguente:

Esempio 4.3.3 Si supponga che nel cluster C_A il clusterhead old_ch_A è sostituito da new_ch_A.

Si possono presentare tre problemi:

1. old_ch_A ha un messaggio m la cui diffusione non è ancora iniziata;
2. old_ch_A ha un messaggio m la cui diffusione è stata solo parzialmente eseguita;
3. old_ch_A ha un messaggio m di cui è l'unico a conoscere la stabilità globale[6].

Il problema (1) è risolto grazie al fatto che il processo sorgente di m ne fa ripartire la diffusione, come descritto nella sezione precedente, poichè nessun clusterhead corrente ha m. Negli altri due casi si ricostruiscono le informazioni possedute da old_ch_A.

Quando new_ch_A viene eletto, tutte le SM nel suo cluster hanno un nuovo clusterhead, perciò inviano ad esso le informazioni riguardo al proprio stato, come descritto nella sezione precedente. In questo modo, new_ch_A conosce lo stato aggiornato di diffusione dei messaggi all'interno del proprio cluster ed acquisisce possibili messaggi posseduti da qualche SM nel suo cluster che non ha nella propria *unstable*. Analogamente, old_ch_A invia il proprio stato al proprio clusterhead corrente, che lo ingloba nel proprio. Se quest'ultimo è new_ch_A, i problemi (2) e (3) sono risolti perchè new_ch_A impara da old_ch_A sia lo stato di diffusione dei messaggi instabili che le identità dei messaggi stabilizzati. Altrimenti new_ch_A recupera queste informazioni dagli altri clusterhead: new_ch_A fa un broadcast affidabile del proprio stato ad ogni altro clusterhead[7], e ne attende lo stato. Lo stato è costituito, come nel caso considerato nella sezione precedente, dal vettore *stable* e dagli identificatori dei messaggi contenuti in *unstable*. Per quanto detto sopra, new_ch_A riceve perciò anche lo stato di old_ch_A.

La procedura descritta per ricostruire le informazioni di stato non è bloccante rispetto alla generazione di nuovi messaggi, che può riprendere non appena in C_A viene eletto il nuovo clusterhead. Riguardo ad un messaggio m la cui diffusione era già iniziata, si possono verificare due possibilità: (i) m è posseduto solo da new_ch_A; oppure (ii) m è posseduto da più di un clusterhead. Nel primo caso, new_ch_A fa ripartire la diffusione di m come nuovo clusterhead originatore. Nel secondo caso, la diffusione è fatta ripartire dal clusterhead che correntemente ha la SM sorgente di m, supponiamo p, nel proprio cluster. Tutti gli altri clusterhead marcano m come '`sospeso`', e non inviano più alcun messaggio (ack e `cluster-ack` compresi) riguardo a m fino a quando non conoscono l'identità del nuovo originatore. Nel caso in cui p sia disconnesso, la diffusione di m è sospesa da tutti i clusterhead. Essa riprende non

[6] Perchè ne è l'originatore e non ha ancora diffuso questa informazione nella propria *stable*.

[7] Si usa lo stesso meccanismo impiegato per la diffusione affidabile dei messaggi broadcast, ma, a differenza di questi ultimi, lo stato di new_ch_A è ricevuto da tutti i clusterhead ma non diffuso all'interno dei propri cluster se non per la propagazione nei cluster adiacenti.

appena p si riconnette; nel caso in cui il clusterhead a cui p si riconnette non avesse m, è p stesso a farne ripartire la diffusione. Grazie a questo meccanismo:

- evitiamo la comparsa di multipli clusterhead originatori per m, con conseguente moltiplicazione di messaggi e ack duplicati;
- evitiamo che i `cluster-ack` siano inoltrati alla SM che è stata l'originatore del messaggio ma ha perso il proprio ruolo di clusterhead, o, peggio, che continuino a circolare in rete perchè la SM originatore è correntemente disconnessa;
- otteniamo perciò un risparmio nel consumo delle risorse delle SM, al costo di un ritardo nel raggiungimento della stabilità del messaggio.

Se m è localmente stabile in ogni cluster quando new_ch_A viene eletto, allora new_ch_A ne osserva la stabilità globale, e diffonde questa informazione. Se m è noto ad ogni clusterhead ma non è globalmente stabile, è necessario attendere che la sua diffusione venga ripetuta prima di raggiungere la sua stabilizzazione.

4.3.8 Gestione degli acknowledgement

La mobilità delle SM può distruggere il FT per un messaggio m, e di conseguenza i cammini che i `cluster-ack` dovrebbero seguire verso l'originatore di m. Consideriamo la seguente situazione, con riferimento all'esempio mostrato in Figura 8:

Esempio 4.3.4 Il clusterhead ch_D perde il cammino al proprio genitore ch_B nel FT del messaggio m.

Quando ch_D manda il proprio `cluster-ack` a ch_B, osserva la scadenza del timer T_{ack} senza aver ricevuto un ack in risposta. ch_D non è in grado di distinguere tra il caso in cui il proprio `cluster-ack` sia andato perso a causa di fallimenti ed il caso in cui non esista più il ramo dell'albero[8]. Per garantire che il `cluster-ack` sia comunque ricevuto all'originatore, nel caso in cui non siano disponibili informazioni per l'instradamento unicast, il protocollo ricorre al *flooding*: un messaggio di `flood-ack`, contenente le stesse informazioni del `cluster-ack`, è generato da ch_D e inoltrato da tutti i gateway in C_D verso i cluster adiacenti. Se un clusterhead così raggiunto ha ancora il proprio genitore nel FT per m, fonde il `flood-ack` con il proprio `cluster-ack` e riprende la normale trasmissione degli ack verso l'originatore, interrompendo in questo modo il flooding. In caso contrario, i clusterhead adiacenti ripetono il flooding del `flood-ack`. L'invio di un `flood-ack` da parte di ch_D viene fatto periodicamente (ogni T_{ack}), fino a quando ch_D non impara che m è diventato globalmente stabile.

[8]Si ricorda che questo protocollo non fa alcuna assunzione sull'esistenza di un sottostante protocollo di instradamento; in caso contrario, si potrebbero ricavare informazioni da esso, ottimizzando l'algoritmo.

Grazie a questo meccanismo, il protocollo è in grado di adattare la modalità di trasmissione degli ack in accordo alla situazione di connettività esistente nelle diverse zone della rete. Ovviamente, l'uso del flooding genera informazioni duplicate, che vengono rilevate ed eliminate grazie all'uso delle strutture *unstable*.

4.3.9 Struttura dei messaggi

In Figura 11 è mostrata la struttura dei messaggi scambiati dal protocollo: il messaggio inter-cluster, i messaggi di ack e i messaggi per lo scambio di stato. Il primo è usato da clusterhead e gateway per propagare i messaggi broadcast attraverso i cluster. Può avere tipo (i) `bcast`[9] per i messaggi mandati da un clusterhead nel proprio

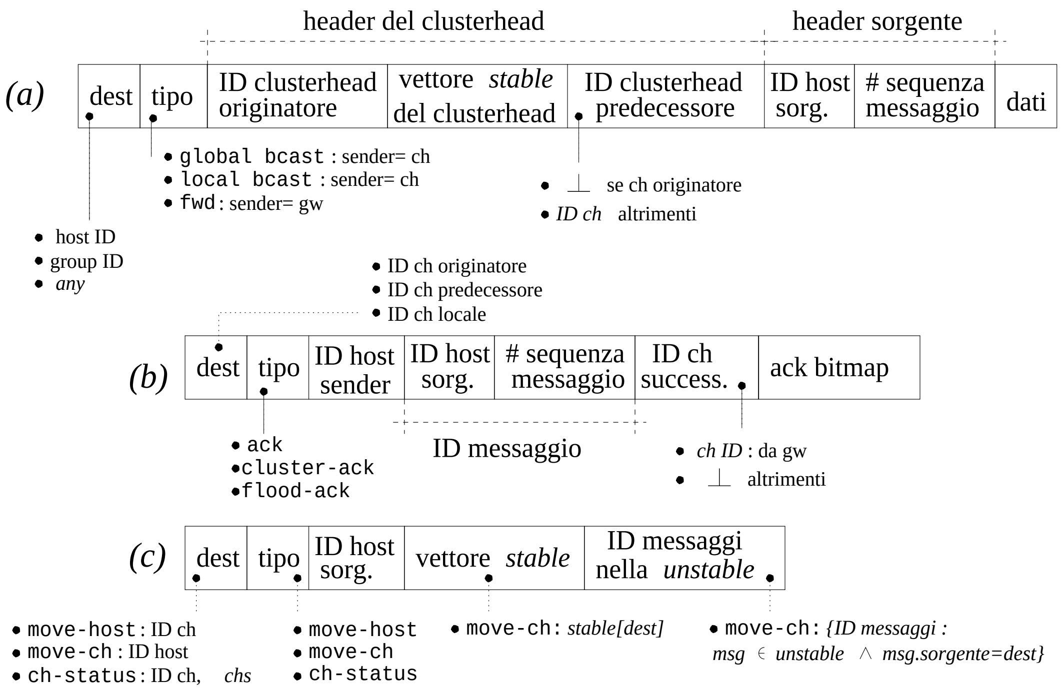


Figura 11: Schema della struttura dei messaggi: (a) messaggi inter-cluster; (b) messaggi `ack`, `cluster-ack` e `flood-ack`; (c) messaggi `move-host`, `move-ch` e `ch-status`.

cluster, oppure (ii) `fwd` nel caso di un messaggio propagato da un gateway a clusterhead ad esso adiacenti. Quando una SM manda un messaggio broadcast al proprio clusterhead, usa un formato di messaggio simile a quello mostrato in Figura 11(a), di tipo `ptp`. Il secondo tipo di messaggio è usato dalle SM che sono host regolari o

[9] `locale` o `globale` nel caso rispettivamente che tutte le destinazioni siano nel cluster dell'originatore o meno.

dai clusterhead per inviare i loro ack, in accordo al protocollo. Può perciò avere tipo (i) `ack` se è mandato da una SM al proprio clusterhead; (ii) `cluster-ack` se è mandato da un clusterhead verso il proprio genitore nel FT; oppure (iii) `flood-ack` se è mandato da un clusterhead all'originatore di un messaggio. Il terzo tipo di messaggio è usato nel caso di cambiamenti topologici. Può essere di tipo (i) `move-host` se è inviato da una SM che entra in un nuovo cluster al proprio nuovo clusterhead; (ii) `move-ch` se è inviato da un clusterhead ad una SM appena entrata nel suo cluster; oppure (iii) `ch-status` se è il broadcast del proprio stato da parte di un clusterhead appena eletto a tutti gli altri clusterhead o la risposta ad esso.

Si può dare una valutazione approssimativa della taglia dei messaggi mostrati. Si assuma che sia necessario 1 byte per rappresentare l'identificatore di una SM e 2 bit per rappresentare il campo `tipo`; sia n il numero di destinazioni del messaggio, *dati* la taglia della parte dati del messaggio e k il numero di messaggi nella coda *unstable*. Allora:

- i messaggi inter-cluster hanno dimensioni $(8n + 42) + dati$ bit;
- i messaggi di ack hanno dimensioni $n + 42$ bit;
- i messaggi di stato hanno dimensioni $8n + 16k + 18$ bit.

4.4 Correttezza del protocollo

In questa sezione si dimostra la correttezza del protocollo rispetto alla definizione del problema (definizione 2.2.1); le dimostrazioni fanno riferimento alla descrizione del protocollo con il formalismo delle macchine a stati riportata in Appendice A.

Si inizi con il notare che l'ipotesi che le SM non falliscono mai crash implica che tutte le loro strutture dati non vengono mai distrutte dai fallimenti; di conseguenza:

Osservazione 4.4.1 *Nè i messaggi ricordati nelle code di messaggi e nelle strutture unstable nè le informazioni mantenute nelle strutture stable sono mai persi.*

Questo risultato sta alla base di tutte le successive dimostrazioni.

Lemma 4.4.1 *Se una SM p invia un messaggio m ad un vicino one-hop q, allora prima o poi q riceve m.*

DIM. Se il movimento distrugge la vicinanza tra le due SM, allora l'ipotesi non è più vera. Altrimenti la tesi è conseguenza della proprietà di Liveness (proprietà 4.2.1). □

Questo Lemma è vero anche nel caso si verifichino fallimenti transienti, perchè si assume che il protocollo di livello MAC ritrasmetta ogni pacchetto fino alla ricezione con successo ai nodi vicini che ne sono destinatari.

Per provare che il protocollo WRBcast risolve il problema del Reliable Broadcast, iniziamo con il considerare il caso in cui la topologia sia statica.

Lemma 4.4.2 *Nel caso di topologia statica affetta da fallimenti transienti, un messaggio m diffuso da un clusterhead ch_A (i) è consegnato da ch_A; e (ii) prima o poi è ricevuto sia da tutte le SM nel cluster di ch_A sia da tutti i clusterhead adiacenti a ch_A.*

Dim. La tesi è conseguenza della descrizione del protocollo, che prevede la ripetuta trasmissione di un messaggio fino alla ricezione degli ack (stato CH-SEND, evento di scadenza di T_b nella macchina a stati del clusterhead) e della proprietà di Liveness, applicando ripetutamente il Lemma 4.4.1 alla trasmissione di m (i) da ch_A ad ognuna delle SM in C_A (e perciò anche ai gateway) e (ii) da ogni gateway in C_A ad ogni clusterhead adiacente a tale gateway. □

Lemma 4.4.3 *Sia p una SM che riceve un messaggio m; allora p consegna m esattamente una volta.*

Dim. La tesi segue dal protocollo e dall'ipotesi che le SM non falliscono mai crash. Quando p riceve m per la prima volta, consegna m e lo memorizza nella propria *unstable* (stato IDLE, eventi RBCASTRq e MSGrecvInd nella macchina a stati del clusterhead; stato IDLE, eventi RBCASTRq, MSGrecvInd e MSGbcastInd nella macchina a stati degli host regolari). Dalla descrizione del protocollo e dall'Osservazione 4.4.1, da quel momento in poi m è ricordato o nella *unstable* o nel vettore *stable*[10](evento STABLEbcastInd nelle macchine a stati). Perciò p è in grado di rilevare ed eliminare possibili duplicati di m e non consegnerà più una copia di m. □

Teorema 4.4.1 *Il protocollo* WRBcast *risolve il problema del Reliable Broadcast nel caso di topologia statica senza partizioni, affetta da fallimenti transienti.*

Dim. Sia p una SM che genera un messaggio m e sia ch_A il clusterhead originatore di m; proviamo la tesi per m. La proprietà di Integrità è conseguenza del Lemma 4.4.3. Grazie alla proprietà di Liveness e all'assenza di partizioni, applicando ripetutamente il Lemma 4.4.2 a tutti i clusterhead si dimostra che tutte le SM ricevono m, e consegnano m in accordo al Lemma 4.4.3. Da questo seguono le proprietà di Validità e Accordo.

Dalla proprietà di Liveness, ogni trasmissione di m su un link logico è completata con successo entro un tempo finito; perciò la diffusione di m a tutte le SM del sistema è completata entro un tempo finito, da cui segue la proprietà di Terminazione. □

[10]Nel senso che il numero di sequenza di m è minore o uguale all'elemento della *stable* corrispondente alla SM che lo ha generato.

Dimostriamo ora che il problema è correttamente risolto nel caso di spostamenti delle SM che non provocano cambiamenti nell'infrastruttura dei clusterhead.

Lemma 4.4.4 *Sia p una SM che genera un messaggio m. Nel caso di movimento delle SM che non provoca cambiamenti nell'infrastruttura dei clusterhead, prima o poi un clusterhead inizia la diffusione di m.*

DIM. Grazie alla proprietà di Liveness, p prima o poi entra in un cluster, supponiamo C_A, e vi resta per un tempo sufficientemente lungo da permettergli di completare lo scambio di informazioni di stato con ch_A e lo scambio di un messaggio. Dallo scambio di stato, p impara che il proprio clusterhead corrente non possiede m. Prima o poi m sarà il messaggio inviato con successo da p al suo clusterhead corrente dopo lo scambio di stato. In accordo alla descrizione del protocollo (stato IDLE, evento MSGrecvInd nella macchina a stati del clusterhead) e all'ipotesi che l'infrastruttura dei clusterhead non cambia, prima o poi tale clusterhead inizierà la diffusione di m. □

Lemma 4.4.5 *Sia p una SM destinataria di un messaggio m. Nel caso di movimento delle SM che non provoca cambiamenti nell'infrastruttura dei clusterhead e in assenza di partizioni, prima o poi p consegna m.*

DIM. L'assenza di partizioni e la proprietà di Liveness garantiscono, applicando ripetutamente il Lemma 4.4.1, che ogni clusterhead prima o poi riceve m. Applicando le stesse considerazioni fatte per la dimostrazione del Lemma 4.4.4, si dimostra che prima o poi p entra in un cluster, supponiamo C_A, e resta connesso per il tempo sufficiente a completare lo scambio di informazioni di stato, in conseguenza del quale ch_A impara che p non ha ancora ricevuto m, e a ricevere m da ch_A. Dal Lemma 4.4.3, p consegna m. □

Teorema 4.4.2 *Il protocollo* WRBCAST *risolve il problema del Reliable Broadcast nel caso di mobilità delle SM che non provoca cambiamenti nell'infrastruttura dei clusterhead e in assenza di partizioni.*

DIM. Sia p una SM che genera un messaggio m. Dal Lemma 4.4.4 p prima o poi invia m al proprio clusterhead corrente, supponiamo sia ch_A, che inizia la diffusione di m. Applicando le medesime considerazioni fatte per la dimostrazione del Teorema 4.4.1, si dimostra che tutte le SM e i clusterhead che non si muovono prima o poi ricevono m e lo consegnano esattamente una volta. Sia q una SM che si sposta mentre è in corso la diffusione di m. Dal Lemma 4.4.5 q prima o poi consegna m, e per il Lemma 4.4.3 lo consegna esattamente una volta. Di conseguenza, le proprietà di Integrità, Validità e Accordo sono soddisfatte.

La proprietà di Liveness è verificata entro un tempo finito; di conseguenza:

1. entro un tempo finito p invia m a ch_A, che inizia la diffusione di m (per il Lemma 4.4.4);

2. entro un tempo finito m è diffuso affidabilmente a tutte le SM che non si spostano (per il Teorema 4.4.1);

3. entro un tempo finito m è ricevuto e consegnato esattamente una volta da tutte le SM che si spostano (per il Lemma 4.4.5).

Perciò entro un tempo finito m è consegnato da tutte le destinazioni, soddisfacendo la proprietà di Terminazione. □

Per dimostrare il risultato principale, consideriamo ora gli effetti dei cambiamenti nell'infrastruttura dei clusterhead. Si ricorda che per le assunzioni fatte sul sistema (sezione 4.2) le partizioni della rete sono prima o poi riparate; di conseguenza, prima o poi esiste un cammino per inviare messaggi ad una SM.

Lemma 4.4.6 *Sia p una SM che genera un messaggio m. Nel caso in cui il clusterhead di p, old_ch_A, sia sostituito da un nuovo clusterhead new_ch_A prima che m diventi globalmente stabile, m è prima o poi diffuso a tutte le sue destinazioni.*

DIM. In seguito al cambiamento di topologia si possono verificare i seguenti casi:

1. nessun clusterhead ha m. Allora, dalla proprietà di Liveness, p prima o poi fa ripartire la diffusione di m dal proprio clusterhead corrente. A questo punto si applica il Teorema 4.4.2;

2. solo new_ch_A possiede m. Dalla descrizione del protocollo (stato I-AM-NEWCH, evento NEWCHbcastInd nella macchina a stati del clusterhead), new_ch_A prima o poi fa ripartire la diffusione di m. A questo punto si applica il Teorema 4.4.2;

3. solo un vecchio clusterhead o più di un clusterhead conosce m. Dalla proprietà di Liveness prima o poi p si riconnette ad un clusterhead, supponiamo sia ch_B, e resta connesso per un tempo sufficientemente lungo da scambiare con successo con ch_B le informazioni di stato ed uno dei messaggi parzialmente diffusi. Se ch_B non ha m, si applicano le medesime considerazioni fatte per il Lemma 4.4.4; altrimenti, dalla descrizione del protocollo (stato WAIT, evento NEWCHbcastInd nella macchina a stati del clusterhead), ch_B fa ripartire la diffusione di m. In entrambi i casi, una volta che la diffusione di m è ripresa, si applicano le medesime considerazioni fatte per il Teorema 4.4.2.

□

Lemma 4.4.7 *Sia p una SM che genera un messaggio m, con clusterhead originatore ch_A. Se un un clusterhead destinatario old_ch_B è sostituito da un nuovo clusterhead new_ch_B, m prima o poi è diffuso a tutte le sue destinazioni.*

DIM. Si possono verificare i seguenti casi:

1. p non si sposta e ch_A rimane il clusterhead originatore. In seguito allo scambio di informazioni di stato tra clusterhead, ch_A viene a conoscenza del fatto che new_ch_B non possiede m. Dalla proprietà di Liveness, applicando ripetutamente il Lemma 4.4.1 alla trasmissione di m da ch_A a new_ch_B, entro un tempo finito m è ricevuto con successo da new_ch_B. Dalla descrizione del protocollo (stato IDLE, evento MSGrecvInd nella macchina a stati del clusterhead), prima o poi new_ch_B diffonde m nel proprio cluster; a questo punto si applica il Teorema 4.4.2;

2. p si disconnette e ch_A rimane il clusterhead originatore: si applicano le medesime considerazioni del punto precedente;

3. p si sposta in C_B e lo scambio di informazioni con new_ch_B termina con successo. Allora p nota che new_ch_B non possiede m e invia ad esso il messaggio; new_ch_B diventa il nuovo clusterhead originatore per m e ad esso si applica il Lemma 4.4.6;

4. p si sposta in C_C e lo scambio di informazioni con ch_C termina con successo. Se ch_C possiede già m, allora ch_A rimane il clusterhead originatore e si applicano le medesime considerazioni del punto (1). Altrimenti ch_C diventa il nuovo originatore e ad esso si applicano le medesime considerazioni del punto (1).

□

Teorema 4.4.3 *Il protocollo* WRBCAST *risolve il problema del Reliable Broadcast nell'ambiente considerato.*

DIM. La tesi è già stata provata nel caso di topologia statica affetta da fallimenti transienti e nel caso di mobilità delle SM che non modifica l'infrastruttura dei clusterhead. Nel caso di cambiamenti all'insieme dei clusterhead, le proprietà di Liveness e di Stabilità dei clusterhead (proprietà 4.2.2) assicurano che l'esecuzione del protocollo prima o poi avanza anche nel caso in cui i clusterhead cambino dinamicamente. Nel caso peggiore, tutte le SM diventano ciclicamente clusterhead. Applicando ripetutamente i Lemmi 4.4.5, 4.4.6 e 4.4.7, si dimostra che ogni SM riceve m o quando è clusterhead o quando si trova nel cluster di un clusterhead che possiede m. Perciò, prima o poi tutte le SM ricevono m, e lo consegnano esattamente una volta per il Lemma 4.4.3, soddisfacendo quindi le proprietà di Integrità, Validità e Accordo. La proprietà di Terminazione deriva dal fatto che le proprietà di Liveness e di Stabilità dei clusterhead sono verificate entro un tempo finito. □

Teorema 4.4.4 *Il protocollo soddisfa la proprietà di Quiescenza Eventuale nell'ambiente considerato.*

DIM. Sia m un messaggio diffuso da una SM p, il cui clusterhead originatore è ch_A. Dal Teorema 4.4.3, prima o poi tutte le SM ricevono e consegnano m. Grazie alla proprietà di Liveness, prima o poi ogni SM invia con successo il proprio ack in risposta alla ricezione di m; perciò prima o poi ogni clusterhead impara la stabilità locale di m nel proprio cluster. Se l'infrastruttura dei clusterhead non cambia, nel caso peggiore ogni clusterhead genera messaggi di `flood-ack` indirizzati a ch_A. Dalla proprietà di Liveness, applicando ripetutamente il Lemma 4.4.1 alla trasmissione di questi ack, prima o poi ch_A scopre la stabilità globale di m. Se l'infrastruttura dei clusterhead cambia, grazie alla proprietà di Stabilità dei clusterhead prima o poi lo scambio di informazioni di stato tra clusterhead è completato con successo e i nuovi clusterhead scoprono la stabilità di m. Le stesse considerazioni si applicano a tutti i messaggi generati da p prima di m; perciò, prima o poi il numero di sequenza di m viene ricordato nel vettore *stable* di qualche clusterhead. Applicando il Teorema 4.4.3 alla diffusione o del primo messaggio insieme al quale questa *stable* è inviata, o alla diffusione della *stable* stessa, si dimostra che prima o poi tutte le SM conoscono la stabilità di tutti i messaggi che precedono m e di m stesso. Dalla descrizione del protocollo (evento STABLEbcastInd nella macchina a stati degli host regolari), prima o poi tutte le SM cancellano m dalle loro *unstable* e non mandano ulteriori messaggi riguardanti m. □

4.5 Analisi del protocollo

In questa sezione si presenta un'analisi di complessità del protocollo WRBCAST, in termini di complessità di comunicazione, latenza e occupazione di memoria. Si discutono i risultati ottenuti dalla misurazione delle prestazioni mediante tecniche di simulazione. Si analizzano le prestazioni del protocollo adattato al caso di sistemi single-hop, in confronto a quelle offerte dal protocollo per il Reliable Broadcast descritto in [1, 2].

4.5.1 Valutazione della complessità

Definiamo la complessità del protocollo WRBCAST in termini dei tre seguenti parametri:

complessità di comunicazione MSG: quantità di messaggi scambiati dal protocollo per la diffusione affidabile di un messaggio broadcast;

complessità temporale TL: tempo di latenza, ovvero tempo trascorso dal momento della generazione di un messaggio broadcast al momento in cui il suo clusterhead originatore ne apprende la stabilità globale. Con TL_S si indica il tempo necessario perchè la stabilità globale del messaggio sia nota in tutto il sistema;

complessità di memoria UNS: numero di messaggi contenuti nelle strutture *unstable*.

Ai fini dell'analisi, si assume che un messaggio broadcast occupi un singolo pacchetto di rete, e che i tempi di computazione siano trascurabili. Si indica con t_r il tempo richiesto per trasmettere un messaggio all'interno di un cluster, ovvero il tempo di trasmissione su un link logico. Entro un tempo $2t_r$ un clusterhead può ricevere gli ack per quel messaggio dalle SM nel proprio cluster, ed entro un tempo $4t_r$ può ricevere gli ack dai propri gateway, che lo informano che il messaggio è stato ricevuto dai clusterhead adiacenti; perciò $T_b \simeq 4t_r$. Nel caso di una rete wireless WaveLAN avente una larghezza di banda di 2 Mbps T_b vale approssimativamente 20 msec.

Sia D la distanza in hop tra il clusterhead originatore e il clusterhead foglia più lontano da esso. In assenza di fallimenti transienti, quando la topologia è statica si ha $TL = O((4 + 2D)t_r)$; infatti, il messaggio e i `cluster-ack` attraversano globalmente due volte il diametro della rete, e 4 hop ulteriori sono richiesti per la comunicazione tra la SM sorgente del messaggio ed il clusterhead originatore, e tra il clusterhead foglia e le SM nel suo cluster. Ulteriori $D + 1$ hop, ognuno richiedente un tempo $O(t_r)$, sono necessari perchè la stabilità del messaggio sia nota alle SM differenti dall'originatore. Inoltre, ogni messaggio che riporta l'informazione di stabilità deve aspettare mediamente un tempo $1/2f_m$ prima di essere trasmesso dall'originatore, dove f_m è la frequenza di generazione dei messaggi broadcast; perciò, il tempo richiesto per stabilizzare un messaggio in tutto il sistema ammonta a $TL_S = O((3D + 5)t_r + 1/2f_m)$. Se n è il numero di SM nel sistema, allora $MSG = O(n)$ e $UNS = O(f_m \cdot TL)$. Per esempio, si supponga che la rete wireless sia una WaveLAN con larghezza di banda di 2 Mbps; se $n = 20$, $D = 4$, $f_m = 50$ messaggi/sec e la massima taglia del campo *dati* del messaggio è 1024 byte, allora TL vale approssimativamente 50 msec e MSG è pari a circa 40 messaggi. Se si verificano fallimenti transienti, si indichi con ρ il numero atteso di ritrasmissioni per superare un fallimento; allora TL e MSG sono pari ai valori precedentemente stimati moltiplicati per ρ.

Per quanto riguarda la taglia dei descrittori di messaggi, essa dipende principalmente dal numero dei discendenti di una SM nel FT, ovvero dal numero dei clusterhead adiacenti. Dato che, per le proprietà di una struttura a cluster (proprietà 1.2.1), due clusterhead non possono mai essere connessi, si ritiene che i cluster non possano essere troppo densi nel sistema e il numero di discendenti nel FT sia molto inferiore a n. La taglia del campo `ack-bitmap` è pari a n, e 4 byte sono sufficienti per ricordare l'identificatore del messaggio e gli identificatori del clusterhead originatore e del nodo genitore nel FT.

Queste valutazioni preliminari indicano che la complessità dell'algoritmo è $O(n)$, almeno finchè non si verificano cambiamenti di topologia; in questo caso, in assenza di un servizio di instradamento unicast, la complessità può degenerare a $O(n^2)$ per effetto dei messaggi di `flood-ack`. In particolare, il movimento di una SM p tra due cluster, che non provochi l'elezione di un nuovo clusterhead, impone un ritardo addizionale pari al più a $O((D+4)t_r) + t_{move}$ prima che un messaggio diventi stabile, dove t_{move} indica il tempo trascorso tra il momento in cui p lascia il vecchio cluster ed il momento della riconnessione di p al nuovo clusterhead. La complessità di comunicazione del protocollo aumenta al più di $O(D+4)$ messaggi unicast.

L'elezione di un nuovo clusterhead impone un ritardo addizionale pari a $O((5D+7)t_r + 1/f_m) + t_{elect}$ sul tempo necessario per stabilizzare un messaggio, dove t_{elect} è il tempo speso per l'esecuzione del protocollo di clustering. Questo limite superiore include lo scambio di informazioni di stato tra clusterhead ed il costo per far ripartire la diffusione di un messaggio instabile. La complessità di comunicazione aumenta di $O(n+2D+2)$.

4.5.2 Valutazione delle prestazioni

Ambiente di simulazione

Il protocollo WRBcast è stato implementato sopra il simulatore Netsimpro (sezione 3.4) allo scopo di valutarne le prestazioni. La rete riprodotta è costituita da un numero variabile di SM che si muovono in un'area quadrata di lato circa 150 metri; il raggio dei cluster (pari alla portata del segnale) è di circa 37 metri. Le SM possono muoversi sia in modo casuale che lungo una traiettoria; nel primo caso, ad ogni spostamento viene scelta casualmente la direzione lungo la quale la SM si muoverà in quel frame alla velocità fissata. Nel secondo caso la direzione del moto è scelta inizialmente e mantenuta finchè la SM non raggiunge i limiti dell'area del sistema. La velocità di movimento è variabile tra 0 e 45 Km/h; il carico di messaggi broadcast nel sistema varia tra 0.83 e 5.83 pck/sec. Il frame rappresenta l'unità di tempo di simulazione; negli esperimenti si è considerato un sistema di 20 nodi con 70 data slot per frame, la cui durata è perciò pari a 0.24 secondi. Il tempo totale di simulazione è di 1000 frame, che corrispondono a 240 secondi di tempo simulato.

Nella realizzazione del protocollo in Netsimpro non si è fatto uso del servizio di instradamento unicast; di conseguenza, le misure di prestazione presentate nella prossima sezione si riferiscono alle condizioni peggiori di funzionamento, in cui il protocollo opera direttamente sopra il livello MAC.

Risultati di simulazione

Gli indici considerati per la valutazione delle prestazioni di WRBcast sono:

tempo di diffusione: tempo trascorso tra il momento in cui il messaggio è generato e il momento in cui esso è stato ricevuto da tutte le SM nel sistema;

tempo di stabilizzazione: tempo trascorso dalla generazione del messaggio al momento in cui il clusterhead originatore del messaggio viene a conoscerne la stabilità globale;

tempo di latenza: tempo trascorso dalla generazione del messaggio al momento in cui la sua stabilità globale è nota a tutte le SM del sistema ed esse lo rimuovono dalle proprie strutture *unstable*;

throughput: numero di messaggi broadcast stabilizzati nel sistema per secondo.

Poichè non esistono in letteratura altri protocolli per il broadcast affidabile in reti mobili ad hoc, l'unico confronto possibile è stato quello con l'algoritmo di *flooding* reso affidabile: una SM p che genera un messaggio broadcast m ne inizia il flooding. Ogni destinazione che riceve m invia unicast a p un ack per m. Se p entro un tempo Δt dalla generazione di m non ha ricevuto tutti gli ack, fa ripartire il flooding di m; negli esperimenti eseguiti si è posto $\Delta t = 200$ frame = 48 secondi.

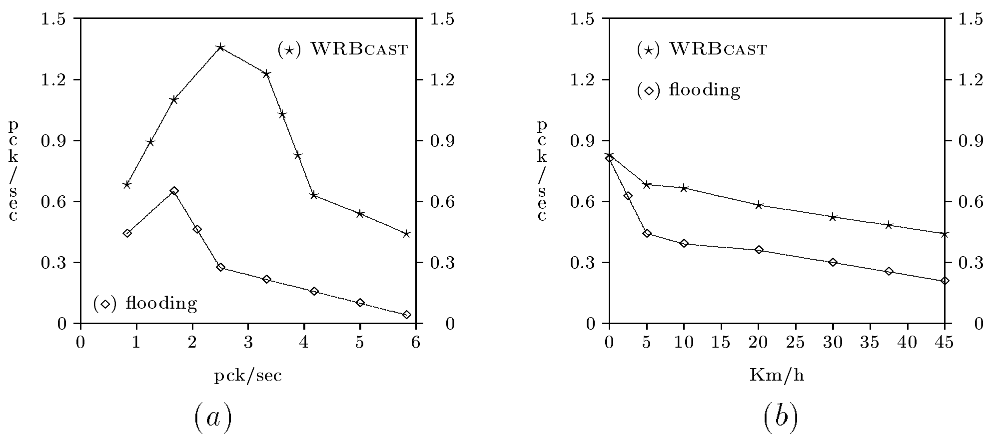


Figura 12: Confronto tra WRBcast e flooding: throughput ottenuto con (a) carico variabile e mobilità random alla velocità di 5 Km/h; (b) carico di 0.83 pck/sec. e mobilità random a velocità variabile.

In Figura 12(a) si confronta il throughput ottenuto da WRBcast e flooding per carico variabile e mobilità casuale alla velocità di 5 Km/h. WRBcast riesce a controllare la diffusione del messaggio broadcast sfruttando l'infrastruttura del FT, il che gli consente di eliminare i loop e i cammini multipli tra coppie di SM; il FT permette anche di ridurre il numero di pacchetti ack generati. Un ulteriore limitazione al traffico deriva dalla capacità di rilevare e risolvere localmente i fallimenti dei canali e i cambiamenti di topologia. Conseguenza di ciò è che WRBcast provoca una minore congestione della rete, riuscendo perciò ad ottenere una maggiore efficienza e a gestire carichi di rete più elevati.

Questo comportamento si nota anche in Figura 12(*b*), dove i due protocolli sono confrontati a fronte di un carico offerto di 0.83 pck/sec. e velocità di movimento casuale variabile. La minore sensibilità di WRBcast alla mobilità è una conseguenza della soluzione locale degli smarrimenti di messaggi. Se una SM p non riceve un messaggio m perchè si sposta in una regione della rete in cui la diffusione di m è già stata effettuata, con WRBcast p riceve subito m dal proprio clusterhead corrente. Al contrario, nell'altro caso deve essere rieseguito il flooding; nel frattempo p può cambiare nuovamente locazione e la precedente situazione può ripresentarsi.

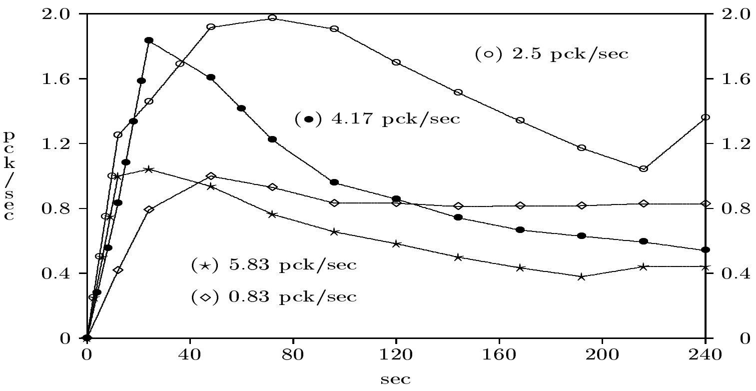
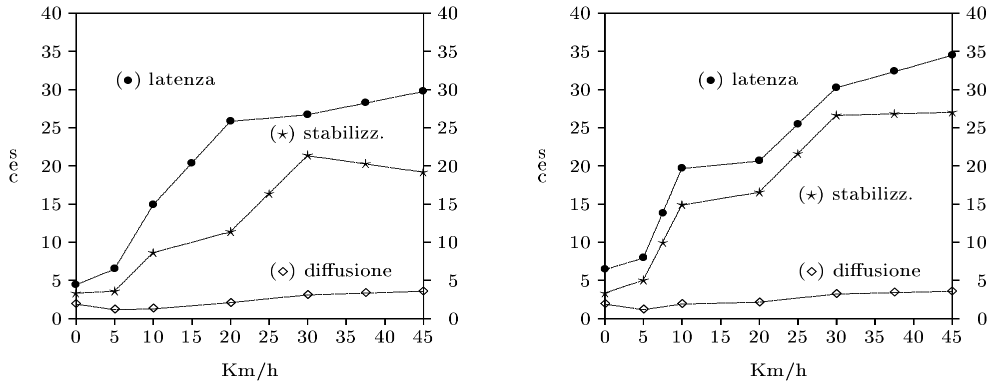


Figura 13: Variazione del throughput nel tempo con carico variabile e topologia statica.

In Figura 13 è mostrata la variazione del throughput nel tempo per differenti carichi e in assenza di mobilità. Al crescere del carico la rete si congestiona sempre più rapidamente. Inizialmente viene raggiunto in tempi brevi un throughput elevato. Quando, con il passare del tempo, aumenta il numero di nodi che hanno generato messaggi e di conseguenza il traffico in rete, il sistema si satura e il throughput decresce. Con il carico offerto più alto tra quelli sperimentati il sistema si congestiona immediatamente. I risultati ottenuti dipendono anche dalle caratteristiche del protocollo MAC impiegato, come si osserverà più in dettaglio nel prossimo capitolo (sezione 5.5); un differente protocollo potrebbe consentire un uso più efficiente della banda e pertanto migliori prestazioni.

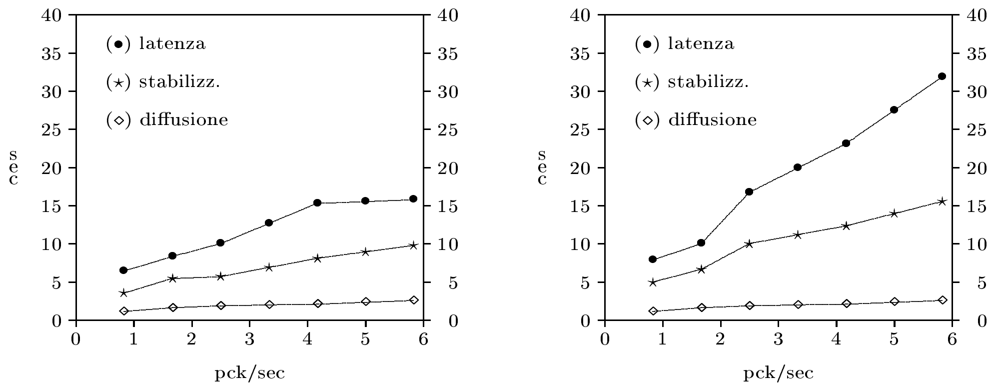
In Figura 14 si confrontano i tempi impiegati per effettuare il broadcast di un messaggio sotto i due modelli di mobilità: casuale e in traiettoria. Gli eperimenti sono stati condotti con un carico di 0.83 pck/sec e velocità variabile. Come osservazione preliminare, si nota che il tempo di latenza di un messaggio m nelle strutture *unstable* è almeno pari al tempo di stabilizzazione del messaggio m più il tempo di diffusione di un nuovo messaggio insieme al quale viene distribuita la struttura *stable* del clusterhead originatore di m che notifica la stabilità globale di m.



(a) (b)

Figura 14: Tempi di diffusione, stabilizzazione e latenza per un messaggio broadcast, con carico offerto di 0.83 pck/sec e velocità variabile con movimento (a) casuale oppure (b) in traiettoria.

Come è ovvio, al crescere della velocità di spostamento diventano più frequenti i cambiamenti di topologia che, come si è osservato in sezione 4.5.1, aumentano i ritardi nel completamento del protocollo. Questo fenomeno è più evidente nel caso di movimento in traiettoria (Figura 14(b)), particolarmente a basse velocità. Infatti



(a) (b)

Figura 15: Tempi di diffusione, stabilizzazione e latenza per un messaggio broadcast, con carico offerto variabile e mobilità alla velocità di 5 Km/h (a) causale o (b) in traiettoria.

a bassa velocità il movimento casuale tende a mantenere i nodi sempre nelle medesime posizioni, per cui la topologia del sistema cambia raramente. Con movimenti in traiettoria la topologia si modifica invece più frequentemente; nel corso delle simulazioni, nel secondo caso si è osservato un maggior numero di partizioni della rete di maggiore durata. Alle alte velocità, il comportamento osservato con mobilità casuale si avvicina a quello osservato con mobilità in traiettoria, perchè l'alta velocità rende comunque più probabili i cambiamenti di topologia. I due modelli di mobilità sono confrontati per bassa velocità (5 Km/h) e carico offerto variabile in Figura 15.

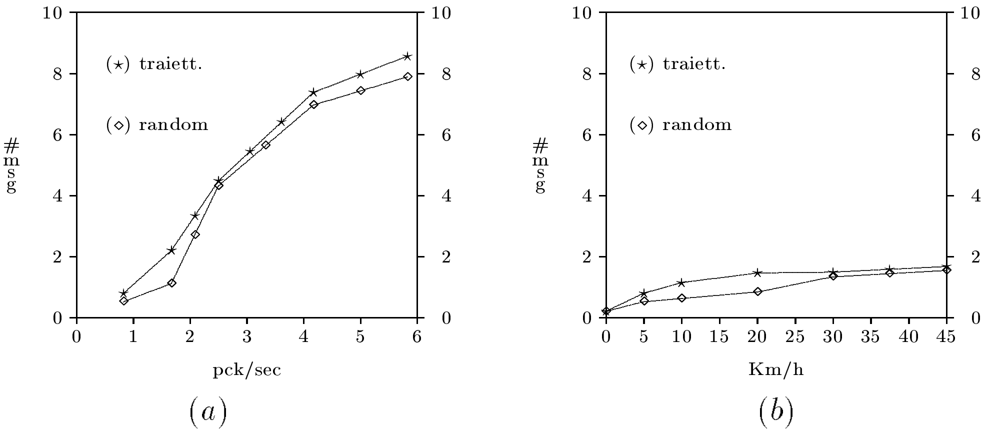


Figura 16: Lunghezza media della struttura *unstable* (a) per velocità di spostamento 5 Km/h e carico offerto variabile; (b) per carico offerto di 0.83 pck/sec e velocità di spostamento variabile.

In Figura 16 è riportata la lunghezza media di una struttura *unstable* in numero di messaggi che vi sono memorizzati. In questo caso i valori ottenuti sono pressochè indipendenti dal modello della mobilità e dalla velocità di spostamento (Figura 16(b)). Questo è dovuto al fatto che i movimenti fanno in effetti rallentare la stabilizzazione dei messaggi (Figura 14), ma il carico offerto è basso; pertanto il numero di messaggi in circolazione, che le SM devono memorizzare, è piccolo. Mentre ovviamente all'aumentare del carico (Figura 16(a)) cresce il numero di messaggi diffusi contemporaneamente e con esso la lunghezza media delle *unstable*.

4.5.3 Adattamento a reti single-hop

Il protocollo WRBCAST è stato progettato con l'obiettivo di ridurre il più possibile la quantità di informazioni mantenute ad ogni SM per descrivere lo stato globale del sistema; tutte le SM, eccetto i clusterhead, mantengono solo informazioni locali. Questo approccio rende semplice l'adattamento del protocollo per il funzionamento in una rete single-hop. Le funzioni dei clusterhead possono essere assegnate alle SSM ed ogni cluster viene perciò a coincidere con una cella; i gateway non sono necessari, poichè l'interconnessione tra le celle è fornita dalla rete fissa. Il FT diventa una struttura statica che realizza l'instradamento tra le SSM attraverso i canali wired.

Se assumiamo, analogamente a [1, 2], che le SSM non falliscano mai e che il tempo di computazione sia trascurabile rispetto ai ritardi di comunicazione, è possibile calcolare il valore degli indici di complessità considerati in sezione 4.5.1. In Tabella 1 sono confrontati il protocollo WRBcast ed il protocollo descritto in [1]. Indichiamo

		MSG	TL
topologia statica	WRBcast	$O(b + mB + (n+1)U + mu)$	$O(3H + 2h)$
	[1]	$O(b + mB + (n+1)U + mu + b)$	$O(3H + 3h)$
fallimenti dei link	WRBcast	$O(b + m\rho B + (n+1)\rho U + mu)$	$O(3\rho H + 2h)$
	[1]	$O(b + m\rho B + (n+1)\rho U + mu + b)$	$O(3\rho H + 3h)$
mobilità delle SM	WRBcast	$< \infty$ /* $O(2U)$ $\forall$ move */	$< \infty$ /* $O(2H)$ */
	[1]	$< \infty$ /* $O(2U + 2u)$ $\forall$ move */	$< \infty$ /* $O(2H + 2h)$ */

Tabella 1: Confronto della complessità.

con B e b i messaggi broadcast che attraversano rispettivamente la rete wireless e la rete wired, con U (u) i messaggi unicast; m rappresenta il numero di SSM nel sistema. TL è calcolato come la somma del numero di hop nella rete wired (h) e nella rete wireless (H). Nel caso di topologia statica, WRBcast richiede:

- 1 messaggio broadcast dalla SSM originatrice alle altre SSM;
- 1 messaggio broadcast in ogni cella alle SM locali;
- n messaggi unicast per gli ack dalle SM alle rispettive SSM;
- 1 messaggio unicast da ogni SSM all'originatrice, contenente gli ack raccolti localmente.

Il protocollo descritto in [1] richiede un ulteriore messaggio broadcast sulle rete fissa, generato dalla SSM originatrice alle altre SSM per informarle che il messaggio è stabile e consentire loro di cancellarlo dai buffer. Da MSG si può ricavare facilmente TL.

Nel caso di canali wireless affetti da fallimenti transienti, non si può determinare alcun limite superiore ai valori di MSG e TL; i risultati in Tabella 1 sono ricavati ipotizzando che un fallimento sia superabile con un numero atteso ρ di ritrasmissioni. Nel caso di mobilità delle SM entrambi i protocolli possono garantire solo una terminazione eventuale, sfruttando la proprietà di Liveness (indicata come *assunzione sulla mobilità* in [1]). Ogni volta che una SM si sposta in una cella differente, per WRBcast scambia due messaggi di stato con la nuova SSM; al contrario, il protocollo in [1] non prevede alcuno scambio di messaggi a fronte di questo evento, ma richiede l'esecuzione della procedura di *handoff* tra la vecchia e la nuova SSM.

Sia f_m la frequenza di generazione dei messaggi broadcast; come precedentemente stimato, il messaggio che informa sulla raggiunta stabilità di un messaggio broadcast deve aspettare in media $1/2f_m$ prima di essere inviato. Di conseguenza, il tempo che trascorre tra la generazione di un messaggio broadcast e la sua stabilizzazione a tutte le SM è $TL + 1/2f_m + H + h$. Come in sezione 4.5.1, si può valutare anche la complessità di memoria UNS, che risulta essere $UNS = O(f_m \cdot TL)$.

4.6 Conclusioni

In questo capitolo si è descritto un protocollo originale per la soluzione del problema del Reliable Broadcast in reti mobili ad-hoc. La soluzione del problema considerato è fondamentale per la realizzazione di servizi di comunicazione di gruppo a valore aggiunto.

Il protocollo è in grado di operare anche in assenza di servizi di instradamento, benchè la disponibilità di un servizio di instradamento unicast possa essere sfruttata al fine di migliorarne le prestazioni. Il protocollo è stato dimostrato corretto: esso garantisce la consegna affidabile di un messaggio a tutte le SM nel sistema entro un tempo finito, e garantisce il rilascio eventuale di tutte le risorse della rete impiegate per il raggiungimento di questo obiettivo.

Si è valutata analiticamente la complessità del protocollo e se ne sono misurate le prestazioni mediante tecniche di simulazione. Poichè non esistono in letteratura protocolli di Reliable Broadcast progettati appositamente per reti multi-hop, il comportamento del protocollo è stato confrontato con quello dell'algoritmo di flooding reso affidabile. La capacità del protocollo di:

1. risolvere localmente i problemi dovuti a fallimenti, mobilità e disconnessioni;
2. adattare la modalità di recupero in una zona della rete in funzione delle condizioni locali di connettività;
3. sfruttare l'infrastruttura di instradamento individuata durante la fase di diffusione del messaggio per rendere più efficiente la procedura di raccolta degli ack;

lo avvantaggiano rispetto al protocollo di confronto, in termini sia di tempi di latenza sia di numero di messaggi generati.

Dato che il protocollo è facilmente adattabile a reti single-hop, si è effettuato analiticamente il paragone tra di esso e il protocollo descritto in [1] in questo ambiente; dal confronto risulta che i due protocolli hanno prestazioni comparabili.

In futuro si potranno studiare adattamenti del protocollo alle caratteristiche di particolari architetture, così da sfruttare i servizi dei livelli inferiori che esse offrono per ottenere migliori prestazioni.

Capitolo 5

Mobile Shared Tree

5.1 Introduzione

In questo capitolo si descrive un protocollo originale per l'instradamento multicast ideato per reti wireless multi-hop (protocollo *Mobile Shared Tree*, o MSTP in breve) [57]. Il protocollo realizza una struttura ad *albero condiviso*, ovvero, una struttura comprendente tutti i membri del gruppo multicast, ma che *non* comprende gli host sorgenti di messaggi destinati al gruppo. Un messaggio generato da un processore non appartenente all'albero delle destinazioni, viene inviato punto-punto al nodo radice dell'albero, che provvederà poi ad instradarlo lungo l'albero. Questo approccio all'instradamento multicast è ispirato al *Core-Based-Tree* originariamente proposto per Internet [12].

In reti mobili multi-hop i protocolli di instradamento progettati per reti fisse non possono essere adottati senza modifiche. Nelle reti fisse, a differenza delle reti mobili, i cambiamenti di topologia sono poco frequenti mentre esiste abbondanza di banda e di memoria delle stazioni. Nelle reti mobili, può essere più importante la capacità di adattarsi rapidamente ai cambiamenti topologici piuttosto che l'abilità di individuare i cammini ottimali.

La principale caratteristica che differenzia il protocollo realizzato da altri precedentemente proposti in letteratura è il fatto che esso adotta un approccio *ibrido*, nel senso che la costruzione e la manutenzione dell'infrastruttura di instradamento sono fatte *on-demand*, ovvero solo al momento in cui la struttura deve essere usata per instradare un messaggio multicast, mentre per la cancellazione dei rami obsoleti viene adottato un approccio *soft-state* (si rimanda alla sezione 2.5.2 per una discussione degli approcci hard-state, soft-state e on-demand). Questo meccanismo assicura flessibilità e adattabilità ai cambiamenti topologici, con alta efficienza multicast nonostante la mobilità, e una procedura di mantenimento dell'albero di basso costo, che interviene selettivamente solo dove la struttura è stata danneggiata. Il protocollo ha altre importanti caratteristiche: (i) è indipendente dal sottostante servizio di instradamento unicast; (ii) evita l'uso di acknowledgement, migliorando così l'efficienza, grazie all'adozione di un meccanismo di recupero dai fallimenti iniziato dai riceventi; (iii) in presenza di sorgenti con un alto tasso di generazione di messaggi, commuta autonomamente e temporaneamente ad un'architettura ad albero orientato alla sorgente per prevenire la congestione al nodo radice. Infine, il protocollo supporta l'*indirizzamento di gruppo*: l'identificatore $\mathcal{G}$ del gruppo è un alias dell'identificatore del nodo radice dell'albero condiviso per il gruppo $\mathcal{G}$.

Il capitolo è strutturato come segue: si inizia con lo specificare le assunzioni fatte sul sistema, rispetto al modello generale descritto nel capitolo 1. In sezione 5.3 si descrive il protocollo (di cui si può trovare una descrizione in pseudo-codice nell'Appendice B) e se ne dà la dimostrazione di correttezza rispetto al problema di instradamento multicast definito nella sezione 2.3. Quindi si discutono i risultati ottenuti valutando il comportamento dell'algoritmo mediante tecniche di simulazione. La descrizione del protocollo fa riferimento al problema definito in sezione 2.3 (Definizione 2.3.1) e al modello di sistema definito in sezione 2.4 (Definizione 2.4.1).

5.2 Modello multicast

MSTP sfrutta i servizi di un sottostante protocollo unicast; ogni nodo deve conoscere quale dei propri vicini è il proprio successore (*next hop*) sul cammino verso una determinata destinazione. Questo è l'unico requisito imposto al servizio sottostante; non ha importanza quale sia il protocollo impiegato per realizzare il servizio soddisfacendo tale requisito.

Nel seguito indichiamo con $\mathcal{R}$ il nodo radice dell'albero per il gruppo $\mathcal{G}$. Se esistono più gruppi nel sistema, viene eseguita un'istanza del protocollo per ognuno di essi; le istanze del protocollo sono indipendenti. Indichiamo con $\mathcal{T}$ l'albero per $\mathcal{G}$; $\mathcal{T}$ è costituito da SM *riceventi*, ovvero i nodi membri di $\mathcal{G}$, e da *forwarder*, ovvero nodi che fanno parte di $\mathcal{T}$ ma non appartengono a $\mathcal{G}$ (Figura 17). Questi ultimi possono essere solo nodi interni di $\mathcal{T}$. Tutte le SM possono entrare prima o poi a far parte di $\mathcal{T}$ ed eseguire funzioni di instradamento.

La radice $\mathcal{R}$ di $\mathcal{T}$ dovrebbe essere scelta in modo da minimizzare i cammini a tutti i riceventi. Questo è un obiettivo facile da raggiungere quando i membri del gruppo si muovono in modo solidale (ad esempio, nel caso di operazioni su un campo di battaglia mobile); in questo caso $\mathcal{R}$ può essere fissato manualmente al momento della configurazione del sistema. Quando le SM si muovono in modo casuale, ad esempio nel caso di applicazioni per la gestione del movimento di veicoli, o in operazioni di soccorso in situazioni di emergenza, è improbabile che $\mathcal{R}$ mantenga a lungo una posizione ottimale per l'albero. Questo è spesso accettabile nel caso di sistemi piccoli; altrimenti, possono essere utilizzati algoritmi come quelli proposti in [18, 34]

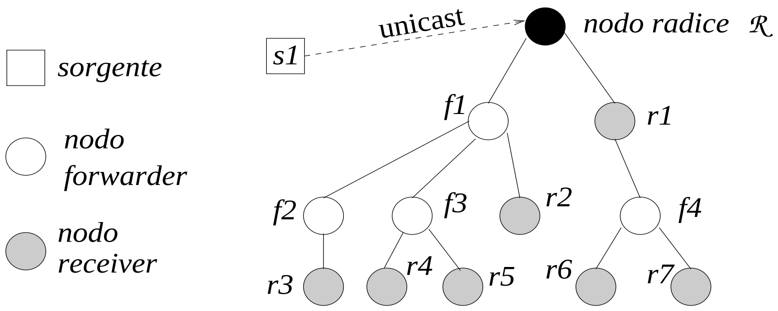


Figura 17: Esempio di albero condiviso costruito da MSTP.

per rilocare dinamicamente $\mathcal{R}$. Qualunque sia la tecnica scelta per determinare $\mathcal{R}$, staticamente o dinamicamente, la sua identità deve essere nota nel sistema. Questo requisito può essere soddisfatto al momento della configurazione del sistema, o contattando a run time un server ben noto che mantiene l'indirizzo di $\mathcal{R}$. Il cammino da $\mathcal{R}$ ad ogni membro di $\mathcal{G}$ è il cammino più breve fornito dal sottostante servizio di instradamento unicast, dove per 'più breve' si intende il cammino di costo minore in accordo alla metrica di costo usata dal servizio di instradamento unicast.

Ogni volta che una SM è sorgente di un messaggio multicast m, essa lo invia unicast a $\mathcal{R}$; una volta che m è sull'albero, viene diffuso lungo i rami dell'albero. I nodi membri del gruppo possono a propria volta generare messaggi; essi si comportano come sorgenti esterne all'albero inviando unicast il messaggio a $\mathcal{R}$, che è l'unica SM responsabile della diffusione dei messaggi multicast. Questo approccio è motivato dalla scelta di sfruttare l'infrastruttura di instradamento multicast nel modo più conveniente in accordo alla natura unidirezionale di molti canali wireless.

In accordo al modello di fallimento assunto (Definizione 2.4.1), i canali wireless possono perdere i messaggi; per garantire la terminazione eventuale del protocollo è necessario fare le seguenti assunzioni:

Proprietà 5.2.1 **[Liveness dei canali wireless]** *Se due SM p e q sono nella medesima partizione e non si verificano cambiamenti di topologia, allora se p invia un pacchetto a q, q riceve il pacchetto entro un numero finito ρ di ritrasmissioni.*

Proprietà 5.2.2 **[Convergenza unicast]** *Il protocollo di instradamento unicast garantisce che, se la topologia si stabilizza, entro un tempo finito le tabelle di instradamento sono consistenti.*

5.3 Descrizione del protocollo

Il protocollo MSTP adotta per la costruzione della struttura di instradamento multicast un approccio *ibrido* che combina i meccanismi hard-state e soft-state e li applica in uno schema on-demand, eliminandone allo stesso tempo alcuni dei problemi. L'approccio ibrido è fondamentalmente un meccanismo soft-state in cui la ricostruzione dell'albero è eseguita *on-demand* quando la radice dell'albero ha un messaggio multicast da diffondere. Ciò significa che la frequenza di ricostruzione è determinata dinamicamente in accordo al tasso di arrivo dei messaggi multicast alla radice. Inoltre, se il tasso di mobilità non è uniforme nella rete, il meccanismo di ricostruzione è attivato solo per i rami danneggiati, mentre non si incorre in alcun costo per il mantenimento delle porzioni connesse dell'albero. Per raggiungere questi risultati, i nodi dell'albero devono mantenere alcune informazioni relative alla struttura dell'albero (analogamente all'approccio hard-state), ovvero gli identificatori del nodo genitore e dei nodi figli. Queste informazioni sono necessarie al nodo radice per eseguire la manutenzione della struttura e per instradare facilmente i messaggi multicast su una rete in cui le SM potrebbero appartenere contemporaneamente a più alberi.

In questa sezione si descrive in dettaglio il funzionamento del protocollo. Inizialmente si specificano le primitive per accedere al servizio. Quindi si illustra la procedura di costruzione dell'albero e si analizza il metodo usato dal protocollo per gestire mobilità e disconnessioni. Si descrive il comportamento del protocollo a fronte della formazione di loop nell'albero o di smarrimenti di pacchetti a causa di fallimenti dei canali. Infine si presenta un possibile meccanismo per il controllo della congestione nel nodo radice dell'albero.

5.3.1 Primitive di servizio

Il servizio di instradamento multicast è accessibile attraverso due tipi di primitive:

- le primitive di `Join.Rq` e `Leave.Rq` per richiedere rispettivamente l'ingresso o l'uscita da $\mathcal{T}$;
- la primitiva `mcast.Rq` per richiedere la diffusione di un messaggio multicast ai membri di $\mathcal{G}$.

La `Join.Rq`($\mathcal{G}$) provoca l'invio di un messaggio unicast di `Join` a $\mathcal{R}$, che ricorda la composizione del gruppo e assume che tutte le SM nel sistema possano generare messaggi multicast per $\mathcal{G}$. La procedura per fare `Join` a $\mathcal{G}$ è rappresentata nell'automa per i riceventi mostrato in Figura 19; il processo richiedente esegue ritrasmissioni finchè non entra a far parte di $\mathcal{T}$. La `Leave.Rq`($\mathcal{G}$) provoca l'invio di un messaggio di `Leave` da un membro del gruppo che vuole lasciare $\mathcal{G}$ a $\mathcal{R}$, che viene ritrasmesso fino alla ricezione di un ack. A questo punto il processo richiedente rilascia tutte le proprie strutture dati riguardo al gruppo. Solo un ricevente che non ha discendenti in $\mathcal{T}$ può generare una `Leave.Rq`. La `mcast.Rq`(m, $\mathcal{G}$) provoca l'invio di un messaggio unicast `multicast` a $\mathcal{R}$ perchè provveda alla diffusione del messaggio multicast m lungo $\mathcal{T}$.

5.3.2 Schema del protocollo

In MSTP la costruzione e il mantenimento dei rami dell'albero sono iniziati *on demand*, vale a dire, quando la radice $\mathcal{R}$ ha un messaggio multicast da diffondere, e sono eseguiti da $\mathcal{R}$. Altri protocolli per la costruzione di alberi di instradamento condivisi (per esempio, quello descritto in [62]) forniscono meccanismi distribuiti per ricostruire i rami danneggiati, in accordo ai quali nodi interni all'albero possono eseguire le operazioni di recupero. Nella versione corrente di MSTP si è adottato un controllo centralizzato per due ragioni:

1. per semplificare il meccanismo di controllo dell'assenza di cicli nella struttura;

2. per evitare che più nodi su un ramo danneggiato in più punti eseguano contemporaneamente la procedura di ricostruzione.

Non appena $\mathcal{R}$ ha un messaggio da diffondere, inizia la costruzione di $\mathcal{T}$ sulla base delle tabelle di instradamento unicast (TU) fornite dal livello sottostante. L'albero è mantenuto con un meccanismo *ibrido*: se la mobilità delle SM non provoca cambiamenti nella connettività di $\mathcal{T}$, allora $\mathcal{T}$ può essere riutilizzato per messaggi successivi. Altrimenti, $\mathcal{R}$ viene informata riguardo alle identità dei riceventi che si sono disconnessi, ma ritarda la loro riconnessione all'albero fino all'arrivo di un nuovo messaggio multicast. Dato che probabilmente il tasso di mobilità non è uniforme nella rete, la ricostruzione dell'albero riguarda solo i rami che sono stati danneggiati, mentre le porzioni dell'albero che rimangono integre sono riutilizzate. Per la costruzione di $\mathcal{T}$ sono usati due tipi di pacchetti di controllo (Figura 18): i pacchetti di `Build` per costruire l'albero o un suo ramo, che vengono spediti appena prima della diffusione di un messaggio multicast, e i pacchetti di `Lost` che sono usati per riportare a $\mathcal{R}$ le identità dei riceventi irraggiungibili[1]. Tutte le altre azioni intraprese dal protocollo per mantenere $\mathcal{T}$ sono basate su un approccio soft-state e non richiedono l'uso di ulteriori pacchetti di controllo.

tipo `Build`	*sorgente*	$\mathcal{G}$	*TD_sorgente.discendenti*	*T_refresh*	*num_hop*

tipo `Lost`	*sorgente*	$\mathcal{G}$	*genitore*	*{destinazioni perse}*

Figura 18: Struttura dei pacchetti di controllo.

La ricezione di una `Build` attiva le funzioni di instradamento di una SM, che da quel momento in poi agisce come un forwarder per un tempo $T_{refresh}$. Allo scadere di questo intervallo, la SM rinuncia al proprio ruolo di forwarder mentre un ricevente

[1] Il significato dei campi mostrati in figura verrà reso chiaro nel seguito.

entra in uno stato temporaneo (COMPLAIN) in cui attende la ricezione di un nuovo pacchetto `Build`. In Figura 19 si mostrano gli schemi di transizione di stato sia per le SM forwarder che per i riceventi. In MSTP il meccanismo che previene lo spegnimento

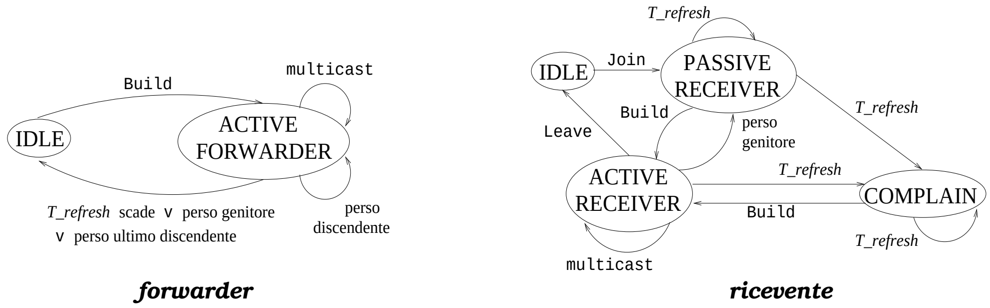


Figura 19: Schemi di transizione di stato.

anticipato dei forwarder è fornito dal flusso di messaggi multicast: $\mathcal{R}$ mantiene una stima del tasso di arrivo dei messaggi, in base al quale computa $T_{refresh}$. Questo valore è poi propagato insieme alle `Build` e ai messaggi multicast a tutti i nodi dell'albero. Nel caso la stima di $\mathcal{R}$ fosse sbagliata, o si verificassero degli smarrimenti di pacchetti a causa di fallimenti sui canali, viene attivata dai riceventi una procedura di recupero.

Nel seguito si descrivono in dettaglio le operazioni eseguite dal protocollo, assumendo inizialmente che non si verifichino fallimenti di canali o smarrimenti di pacchetti. Si descrive la procedura di costruzione dell'albero, e quindi le procedure per il suo mantenimento in caso di mobilità e per la prevenzione dei cicli. Quindi si descrivono i meccanismi usati per superare i fallimenti di link, ed infine la tecnica usata per effettuare il controllo di congestione a $\mathcal{R}$.

5.3.3 Costruzione dell'albero

In Figura 20 si mostrano le strutture dati mantenute da ogni SM nell'albero $\mathcal{T}$ del gruppo $\mathcal{G}$. $IaminG$ permette alla SM di distinguere se è un forwarder o un ricevente. Il vettore S serve a $\mathcal{R}$ per ricordare l'insieme di riceventi disconnessi che devono essere riconnessi a $\mathcal{T}$ non appena arriva un messaggio multicast; inizialmente contiene tutti i membri di $\mathcal{G}$. Ad ogni SM nell'albero S viene utilizzata durante la procedura di costruzione *Build-Tree* (Appendice B). Il descrittore dell'albero TD (*tree descriptor*) descrive per ogni SM nell'albero la porzione di $\mathcal{T}$ per essa rilevante: le identità della SM genitore nell'albero e delle SM discendenti, l'identità di $\mathcal{R}$ e la lista di riceventi che possono essere raggiunti attraverso ogni discendente; la funzione del campo *num_hop* sarà descritta successivamente.

La costruzione dell'albero è iniziata da $\mathcal{R}$ e continuata dalle SM che sono sui cammini più brevi tra $\mathcal{R}$ e i riceventi. Quando riceve un messaggio multicast, $\mathcal{R}$

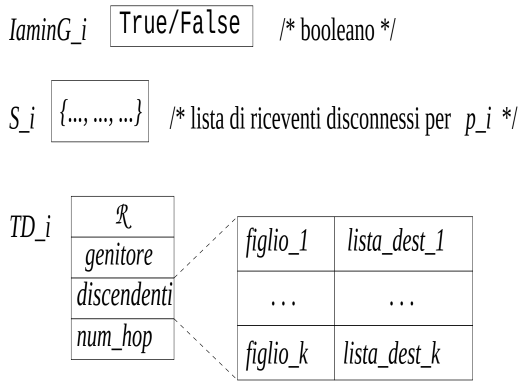


Figura 20: Strutture dati per la SM p_i.

costruisce il campo *discendenti* in $TD_{\mathcal{R}}$ consultando le TU per ogni ricevente elencato in S; quindi fa un broadcast di $TD_{\mathcal{R}}.discendenti$ con un pacchetto di `Build`. I vicini di $\mathcal{R}$ che ne ricevono il broadcast e sono indicati come discendenti nell'albero per un sottoinsieme dei riceventi diventano forwarder per $\mathcal{T}$, mentre le altre SM scartano il pacchetto. La medesima procedura è ripetuta da ogni discendente per le destinazioni ad esso associate, fino ad arrivare ai nodi *foglia*, che sono quei discendenti indicati in un pacchetto `Build` con una lista vuota di destinazioni. In seguito all'invio della

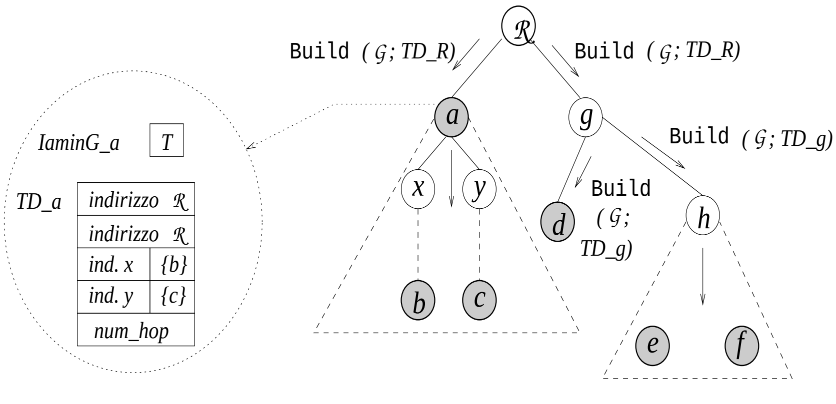


Figura 21: $\mathcal{T}$ per le SM a, b, c, d, e e f, strutture dati di a.

`Build` $\mathcal{R}$ può iniziare la diffusione del messaggio tramite la primitiva `multicast`: il messaggio viene inviato broadcast, e inoltrato lungo l'albero solo dai nodi che sono forwarder mentre le altre SM lo scartano. In Figura 21 si mostra un esempio di albero condiviso che può essere costruito applicando il procedimento ora descritto ad un gruppo $\mathcal{G}$ a cui appartengono le SM a, b, c, d, e, f.

5.3.4 Gestione della mobilità

Quando una SM si sposta, la topologia della rete può cambiare e l'albero può dover essere modificato di conseguenza. MSTP opera come segue: ogni SM nell'albero conosce le identità dei propri vicini one-hop, mantenute nel proprio TD. Finchè le SM riportate in TD vengono indicate a distanza di un hop dal sottostante instradamento unicast, la connettività di $\mathcal{T}$ è assicurata. Consideriamo l'esempio in Figura 22.

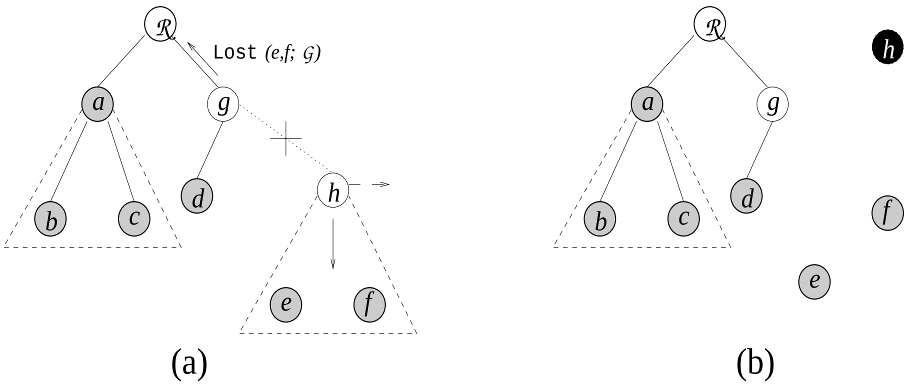


Figura 22: Esempio di mobilità delle SM.

Esempio 5.3.1 Il forwarder h si sposta, perciò provocando la scomparsa del canale tra h ed il suo genitore g nell'albero. Quando g nota che h non è più un vicino one-hop, g rimuove l'elemento corrispondente a h nel proprio TD e manda al proprio genitore ($\mathcal{R}$) un pacchetto di `Lost` per informarlo della disconnessione delle destinazioni e e f, associate a h, da $\mathcal{T}$. Se il canale tra g ed il suo genitore dovesse essere interrotto, g inizializza il proprio TD ed esce silenziosamente dall'albero.

In Appendice B queste operazioni sono descritte dalla procedura *Check-Tree*. In *Check-Tree* si assume che una SM genitore prima o poi rilevi la scomparsa di un proprio discendente; questo verrà formalmente provato in sezione 5.4.

Può accadere che l'informazione riguardo alla disconnessione di un ricevente si propaghi verso $\mathcal{R}$ contemporaneamente alla diffusione da parte di $\mathcal{R}$ di un messaggio; in tal caso, il ricevente disconnesso non riceverà tale messaggio. Il ritardo di propagazione dei pacchetti di `Lost` rappresenta una 'zona d'ombra' per il protocollo, enfatizzata dal fatto che la ricostruzione dei rami danneggiati è sempre iniziata da $\mathcal{R}$; si vedrà in sezione 5.5 come si può compensare questa sorgente di smarrimento di pacchetti con un semplice meccanismo di ridondanza.

L'uso di $T_{refresh}$ elimina la necessità di un meccanismo per la potatura dei rami obsoleti dell'albero, ovvero dei tronconi di rami rotti. Questo meccanismo può però portare alla formazione di sottoalberi disconnessi, quando i nodi appartenenti ad un sottoalbero sono temporaneamente all'oscuro del fatto che si è verificata la rottura di un ramo a livelli superiori di $\mathcal{T}$. Questa inconsistenza scompare alla scadenza di $T_{refresh}$; nel frattempo può però essere rilevata la presenza di cicli apparenti, come si vedrà nella prossima sezione.

5.3.5 Soluzione dei cicli

Si possono formare cicli nell'albero in conseguenza della mobilità delle SM o della lenta convergenza delle TU, che possono essere temporaneamente inconsistenti. La procedura di costruzione dell'albero presenta tre possibili casi di formazione di cicli, rappresentati in Figura 23.

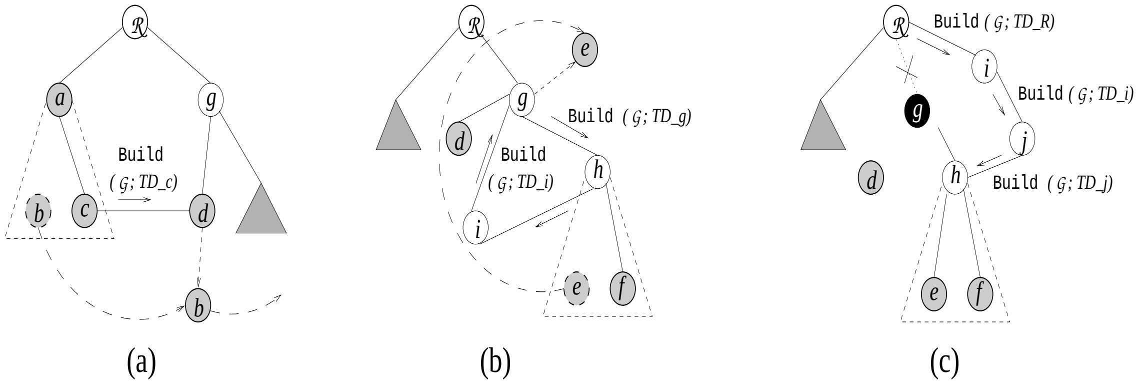


Figura 23: Condizioni di formazione di cicli.

Esempio 5.3.2 In Figura 23(a) la SM b si sposta dal sottoalbero con radice in a al sottoalbero con radice in d; di conseguenza, viene riportata la sua disconnessione a $\mathcal{R}$, che invia una `Build` per riconnettere b non appena riceve un messaggio multicast. La `Build` può essere ancora instradata attraverso il nodo a come risultato del lento aggiornamento della TU di $\mathcal{R}$, e creare poi un cammino sub-ottimo "inseguendo" b nei suoi spostamenti. L'aspetto rilevante non è tanto la creazione di un cammino non ottimo, ma il fatto che il nodo c indica come proprio discendente per la destinazione b la SM d, che si ritrova pertanto ad avere due genitori e ad appartenere a due differenti sottoalberi. Per la conoscenza che d ha della struttura dell'albero, questa situazione potrebbe essere dovuta alla formazione di un ciclo, nel caso c appartenesse al sottoalbero con radice in d.

Esempio 5.3.3 In Figura 23(b) è mostrato un esempio simile al precedente: $\mathcal{R}$ invia una `Build` per connettere la destinazione e mentre questa si sta muovendo lungo la traiettoria tratteggiata mostrata in figura. Il pacchetto di `Build` è instradato attraverso i nodi g e h; h cerca di "catturare" e inoltrando la `Build` a i, che a propria volta la manda a g formando perciò un ciclo.

In entrambi gli esempi precedenti una SM che rileva la possibile formazione di un ciclo (d in Figura 23(a) e g in Figura 23(b)) deve scegliere uno dei due possibili genitori e distruggere il ramo discendente dall'altro. In MSTP la scelta tra i genitori in competizione è fatta usando il seguente meccanismo: la `Build` contiene il campo *num_hop*, che è inizializzato a 0 da $\mathcal{R}$; ogni nodo che riceve la `Build` ricorda il valore di *num_hop* che associa al proprio genitore e lo incrementa prima di inviare la `Build` ai propri discendenti. Quando un nodo già in $\mathcal{T}$ riceve una `Build` da un possibile nuovo genitore, confronta il valore di *num_hop* contenuto in essa con quello associato al proprio genitore corrente; mantiene il genitore a cui corrisponde il valore minore e distrugge il ramo proveniente dall'altro inviando ad esso una `Lost`. In questo modo si privilegia il cammino più corto che unisce una SM a $\mathcal{R}$. Questo meccanismo funziona correttamente in entrambi gli esempi precedentemente mostrati, ma non nel seguente caso.

Esempio 5.3.4 In Figura 23(c), g perde il proprio genitore e di conseguenza lascia $\mathcal{T}$ silenziosamente; $\mathcal{R}$ nota la disconnessione di g e considera persi i riceventi dipendenti da esso, vale a dire e e f. La `Build` mandata da $\mathcal{R}$ può cercare di riconnettere le due destinazioni attraverso un cammino alternativo verso h, passante per i nodi i e j, prima che scada $T_{refresh}$ per h; di conseguenza h ritiene di avere due genitori e distrugge il nuovo ramo perchè sospetta la formazione di un ciclo.

In questo caso l'uso di *num_hop* porta alla scelta sbagliata. L'uso di un contatore di versione dell'albero funzionerebbe correttamente in tutti i casi ma introdurrebbe il problema del riciclo dei numeri di versione; in questa prima versione di MSTP si è perciò preferito usare *num_hop*. La situazione mostrata nell'esempio 5.3.4 viene comunque risolta allo scadere di $T_{refresh}$ a h, al costo di una riduzione del throughput; si stima che comunque tale situazione abbia una probabilità molto bassa di verificarsi.

Le operazioni per la gestione dei sospetti loop sono descritte nella procedura *Loop-Solve* in Appendice B. Per superare possibili fallimenti che riguardino la `Lost` mandata al genitore rifiutato, una SM p nell'albero invia una `Lost` ad un nodo q che non considera come proprio padre se osserva la diffusione da parte di q di un messaggio multicast indirizzato anche a p.

5.3.6 Recupero dai fallimenti dei canali

Nelle sezioni precedenti si è assunto che i canali wireless fossero esenti da fallimenti e che le SM avessero una disponibilità infinita di memoria; sotto queste ipotesi i messaggi multicast non sono mai persi, perciò anche se un ricevente p_r in $\mathcal{T}$ osserva lo scadere del proprio $T_{refresh}$ sa che:

- la stima di $\mathcal{R}$ era sbagliata e prima o poi riceverà un `multicast`; oppure
- un messaggio di `Lost` relativo a p_r sarà ricevuto da $\mathcal{R}$ e prima o poi p_r riceverà una `Build` che lo riconnette a $\mathcal{R}$.

Se i pacchetti di `Lost` e `Build` possono essere persi, $\mathcal{R}$ potrebbe non riuscire ad ottenere una visione consistente dello stato dell'albero e i riceventi potrebbero aspettare indefinitamente di essere riconnessi a $\mathcal{T}$. Una possibile soluzione a questi problemi potrebbe essere quella di raccogliere ad ogni hop un ack in risposta alla ricezione di ogni pacchetto di controllo. Un'alternativa meno costosa consiste nell'adottare un approccio soft-state e periodicamente ricostruire l'intero albero, sistemando in tal modo le inconsistenze e riconnettendo i riceventi persi.

Il meccanismo adottato da MSTP è differente dai precedenti: la procedura è iniziata dai riceventi, e non richiede nè la definizione di un periodo di riaggiornamento dell'albero nè l'occasionale flooding di messaggi di membership come ad esempio in [24]. In caso di fallimenti, se un ricevente p_r osserva lo scadere del proprio $T_{refresh}$ e non riceve alcuna `Build`, dopo un certo tempo avvisa $\mathcal{R}$ di essere disconnesso, con un messaggio unicast, ed attende di essere riconnesso. Il messaggio unicast (*bubble*) contiene unicamente il valore di $T_{refresh}$ noto a p_r. Il procedimento è periodicamente ripetuto finchè p_r non riceve una `Build`; la frequenza con cui p_r manda il proprio avviso alla radice è progressivamente ridotto in accordo alla seguente funzione decrescente: $T_{refresh} \leftarrow T_{refresh} - \frac{T_{refresh} - T_{min}}{2}$, dove T_{min} indica il round-trip-delay tra $\mathcal{R}$ e p_r. Se il valore di $T_{refresh}$ stimato da $\mathcal{R}$ è già in partenza inferiore a T_{min}, allora

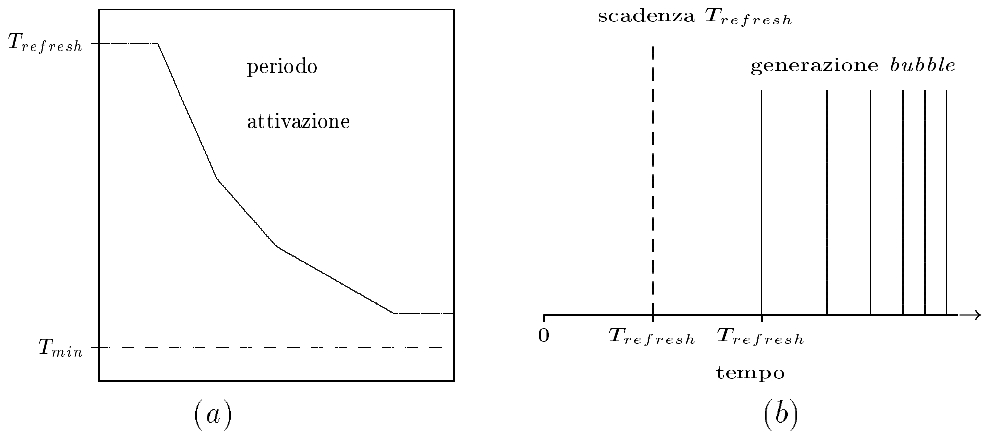


Figura 24: (a) Andamento della funzione per il calcolo del periodo di riattivazione. (b) Frequenza di riattivazione del ricevente p_r.

il periodo con cui p_r riattiva questa procedura è T_{min}. Il valore di T_{min} può essere ottenuto dal sottostante servizio di instradamento unicast. In Figura 24(a) si mostra l'andamento della funzione per l'aggiornamento del periodo di riattivazione, mentre in Figura 24(b) è rappresentata la frequenza di riattivazione della procedura nel tempo.

Alla ricezione del messaggio *bubble*, $\mathcal{R}$ autonomamente decide se rispondere: (i) con un messaggio `wait` in cui fornisce a p_r la nuova stima di $T_{refresh}$; oppure (ii) con un messaggio di `Build`. La scelta è fatta confrontando il valore di $T_{refresh}$ di $\mathcal{R}$ con quello contenuto nel messaggio *bubble*: se $\mathcal{R}$ prevede di ricevere un multicast prima che p_r generi un nuovo messaggio *bubble*, allora opterà per la scelta (ii); per la scelta (i) altrimenti.

La procedura *Complain* in Appendice B include le operazioni ora descritte. In sezione 5.4 si dimostrerà che anche in presenza di smarrimento di messaggi $\mathcal{R}$ conosce entro un tempo finito le identità dei riceventi disconnessi.

5.3.7 Controllo della congestione

Nel caso si utilizzino per l'instradamento multicast degli alberi orientati alle sorgenti, può accadere che un nodo appartenga contemporaneamente a più alberi e abbia difficoltà a smaltire il traffico generato dalle relative sorgenti. In questo caso, i punti di congestione sono distribuiti nella rete ed è difficile realizzare una politica di controllo di congestione che coordini più sorgenti di traffico.

Con l'approccio ad albero condiviso l'unico punto di congestione è la radice $\mathcal{R}$; MSTP realizza in $\mathcal{R}$ il proprio meccanismo per il controllo della congestione. Il meccanismo è autonomamente attivato da $\mathcal{R}$ quando essa osserva l'esaurimento dello spazio disponibile per la memorizzazione dei messaggi multicast inviatigli dalle sorgenti; a questo punto $\mathcal{R}$ inizia a rifiutare ulteriori richieste di servizio dalle sorgenti, propagando in tal modo ad esse lo stato di congestione. In alcuni casi questo temporaneo riempimento dei buffer delle sorgenti può essere sufficiente per risolvere la congestione a $\mathcal{R}$ dovuta ad un burst di traffico; altre volte questo può provocare la congestione e l'overflow dei buffer delle sorgenti, con conseguente perdita di messaggi e riduzione del throughput. Per prevenire questa situazione, in MSTP le sorgenti possono autonomamente decidere come operare per controllare la propria congestione. Quando $\mathcal{R}$ rifiuta la richiesta di servizio di una sorgente, invia ad essa un messaggio `Nack` contenente la composizione del gruppo e l'abilitazione per un tempo τ della sorgente alla costruzione di un proprio albero di instradamento. Alla ricezione del messaggio `Nack` la sorgente può:

- adottare una politica non aggressiva e ripresentare più tardi la propria richiesta, se non si trova in una situazione di congestione e la sovrastante applicazione ha un basso tasso di generazione di messaggi multicast;

- costruire un proprio albero di instradamento $\mathcal{S}$ di cui è la radice, in caso contrario.

L'albero $\mathcal{S}$ viene costruito usando la procedura *Build-Tree*; la sorgente può instradare sopra di esso i propri messaggi multicast finchè non scade il tempo τ indicato da $\mathcal{R}$. Tale tempo corrisponde all'intervallo stimato da $\mathcal{R}$ tra due successive riconfigurazioni dell'albero $\mathcal{T}$; quando esso scade, tutti i nodi appartenenti all'albero $\mathcal{S}$ reinizializzano il proprio stato. Se si verificano dei cambiamenti di topologia durante il tempo di vita di $\mathcal{S}$, essi non vengono riportati alla sorgente, che non intraprende perciò alcuna operazione di manutenzione di $\mathcal{S}$. Il valore di τ è una stima del tasso di mobilità delle SM; usandolo come durata di vita di $\mathcal{S}$ si spera di avere una buona probabilità che $\mathcal{S}$ non si partizioni nel frattempo e i messaggi multicast raggiungano le proprie destinazioni.

5.4 Correttezza del protocollo

In questa sezione si dimostra la correttezza di MSTP rispetto alla definizione del problema data in sezione 2.3 (Definizione 2.3.1). Le dimostrazioni fanno riferimento allo pseudo-codice del protocollo riportato in Appendice B.

Si inizia con il dimostrare che la radice $\mathcal{R}$ dell'albero $\mathcal{T}$ conosce in tempo finito la disconnessione di un ricevente.

Lemma 5.4.1 *In assenza di fallimenti, $\mathcal{R}$ sa la lista dei riceventi disconnessi entro al massimo un tempo t_L dalla loro disconnessione, dove t_L è il tempo impiegato da un pacchetto di* `Lost` *per attraversare il diametro dell'albero.*

DIM. $\mathcal{R}$ impara la disconnessione di un ricevente o attraverso la ricezione di una `Lost` o controllando i propri diretti discendenti in $\mathcal{T}$. Se si sposta un figlio di $\mathcal{R}$, $\mathcal{R}$ immediatamente nota la disconnessione di tutti i riceventi nel sottoalbero con radice in quel figlio. L'upper bound t_L è raggiunto quando una SM foglia p_r di disconnette dal proprio padre; sia T_D l'istante in cui il padre di p_r nota la disconnessione. Sia k la lunghezza del ramo a cui p_r appartiene, e siano $a_1, \ldots, a_{k-1}$ i nodi intermedi tra $\mathcal{R}$ e p_r (Figura 25). Dopo lo spostamento di p_r, il nodo padre a_{k-1} manda una `Lost` a a_{k-2}; o a_{k-2} riceve la `Lost`, oppure i due nodi si spostano e entro un tempo finito a_{k-2} nota la disconnessione del proprio figlio. In entrambi i casi, entro un tempo finito a_{k-2} scopre che una disconnessione si è verificata nel proprio sottoalbero e che una `Lost` deve essere inviata al proprio padre. Ripetendo la dimostrazione per a_{k-2} e a_{k-3} e così via si dimostra la tesi. □

Lemma 5.4.2 *In presenza di fallimenti transienti dei canali wireless, $\mathcal{R}$ sa la lista dei riceventi disconnessi dall'albero ma raggiungibili entro al massimo un tempo t_{Lf} dalla loro disconnessione, dove t_{Lf} è inferiore a $(\rho + 2)T_{refresh} + t_L$.*

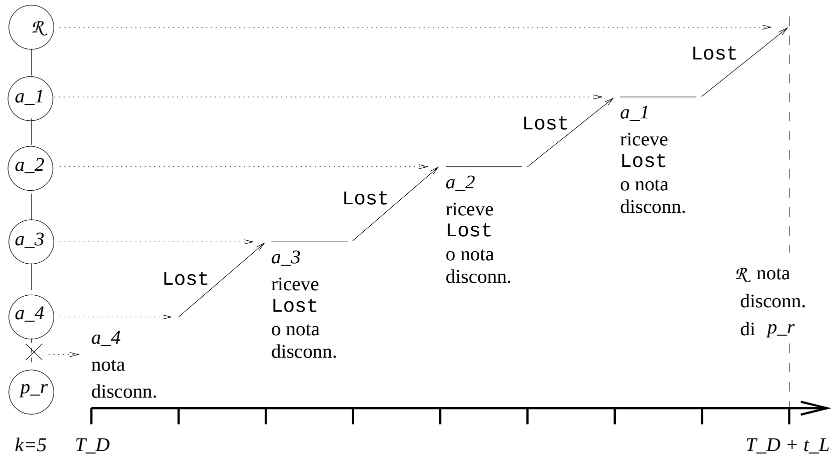


Figura 25: Propagazione dell'informazione riguardo alla disconnessione di p_r.

Dim. La tesi è conseguenza della proprietà di liveness dei canali wireless (proprietà 5.2.1). Sia p_r un ricevente foglia dell'albero, attaccato ad un ramo di lunghezza k, che perde il proprio genitore ma che è raggiungibile da $\mathcal{R}$. Se nessun pacchetto di `Lost` è perso, allora vale il Lemma precedente. Altrimenti entro un tempo finito p_r vede scadere il proprio $T_{refresh}$; nel caso peggiore, p_r riceve un messaggio appena prima di T_D, perciò dopo $T_D + 2T_{refresh}$ attiva il meccanismo di avviso della disconnessione a $\mathcal{R}$ (procedura *Complain*). Applicando la proprietà 5.2.1 al cammino tra p_r e $\mathcal{R}$, entro un numero finito di ritrasmissioni $\mathcal{R}$ riceve il messaggio *bubble* di p_r e ne impara la disconnessione. Poichè p_r genera un pacchetto *bubble* al più ogni $T_{refresh}$, dopo un tempo $\rho T_{refresh} + t_L$ si verifica la ricezione di *bubble*. □

Dimostriamo che le proprietà di safety di cui alla Definizione 2.3.1 sono soddisfatte dal protocollo MSTP.

Lemma 5.4.3 [Safety(1)] *Durante periodi di cambiamenti arbitrari della topologia la struttura mantenuta dal protocollo è priva di cicli.*

Dim. Come osservato durante la descrizione del protocollo, la mobilità delle SM e le inconsistenze temporanee delle TU possono portare alla formazione di due tipi di "cicli": (i) l'intreccio di due sottoalberi (Figura 23(a)); oppure (ii) un ciclo vero e proprio (Figura 23(b)). In entrambi i casi esiste una SM nell'albero che ha due padri.

Dalla descrizione del protocollo, i rami di $\mathcal{T}$ vengono costruiti solo dai pacchetti di `Build` (linee 10-14 Appendice B); una `Build` viene accettata solo se:

1. il nodo non ha alcun padre. Ma in tal caso, per quanto notato, nessuna delle due strutture cicliche può formarsi;

2. il nodo ha già un padre, e la `Build` ricevuta ha percorso un cammino costituito da un minor numero di hop (procedura *Loop-Solve*).

Nel caso (2), dalla descrizione del protocollo il nodo comunque mantiene un solo genitore e manda all'altro un messaggio di `Lost` (linee 11.3-11.4 Appendice B), che viene ricevuto dal genitore rifiutato entro un tempo finito per la proprietà 5.2.1. Di conseguenza, i cicli di cui al punto (i) entro un tempo finito vengono distrutti. Dimostriamo che il meccanismo di scelta di un solo genitore non può portare alla situazione mostrata in Figura 26(a), in cui il nodo a rifiuta il genitore che lo connette alla radice e mantiene l'altro, provocando così la formazione di un ciclo disconnesso dall'albero.

Dalla descrizione del protocollo, nella procedura di costruzione dei rami di $\mathcal{T}$ ogni nodo prima acquisisce un padre, quindi (eventualmente) dei figli (linee 13-14 Appendice B). Consideriamo il ciclo formato dai nodi a, b e c mostrato in Figura 26(a), e supponiamo senza perdita di generalità che il nodo a sia entrato a far parte dell'albero prima degli altri due nodi. Per definizione, $TD_a.num\_hop$ è il valore del campo num_hop nel messaggio di `Build` la cui ricezione ha provocato la connessione di a all'albero. In accordo al protocollo (linea 14 Appendice B), quando a riceve la `Build`, incrementa il valore contenuto nel campo num_hop prima di propagarla a quei vicini che individua come propri figli in $\mathcal{T}$ in accordo alle proprie TU. Perciò, ogni figlio di a entra nell'albero per la ricezione di una `Build` con $num\_hop = TD_a.num\_hop + 1$. La procedura è ripetuta dai figli di a. Poichè, per l'assunzione fatta, c è discendente di a, in quanto entra nell'albero ad un tempo successivo all'ingresso di a, c ha $TD_c.num\_hop > TD_a.num\_hop$ e propaga la `Build` con $num\_hop = TD_c.num\_hop{+}1$. Quando a riceve la `Build` che forma il ramo $c \rightarrow a$, tale `Build` ha quindi $num\_hop > TD_a.num\_hop$ e a possiede già un padre; di conseguenza in accordo al protocollo (procedura *Loop-Solve*) la `Build` viene rifiutata. □

Dal lemma precedente e dal Lemma 5.4.2, i cicli vengono distrutti entro un tempo finito. In realtà, dal punto di vista della diffusione dei messaggi non esistono mai cicli; infatti, se un nodo rifiuta un padre, scarta i messaggi che esso diffonde anche prima che esso sappia di essere stato rifiutato.

Lemma 5.4.4 [Safety(2)] *In caso di cambiamenti arbitrari di topologia, il protocollo MSTP garantisce che entro un tempo finito i cammini non più validi vengono eliminati.*

DIM. I cambiamenti di topologia possono causare due eventi: (i) la scomparsa di cammini esistenti; o (ii) la formazione di cammini alternativi migliori di quelli esistenti[2]. Esaminiamo separatamente i due casi.

[2]Se esistono cammini peggiori, questi non saranno indicati dal servizio di instradamento unicast e quindi non potranno essere impiegati come rami di $\mathcal{T}$ sostituendo i rami attuali.

Caso (*i*): supponiamo che scompaia il canale tra i nodi a e c in Figura 26(b); allora a cancella c dal proprio TD mentre il sottoalbero con radice in c non riceve più messaggi da a. Entro tempo finito c rileva la disconnessione dal padre e reinizializza il proprio stato (procedura *Check-Tree*); di conseguenza il ramo $a \rightarrow c$ viene cancellato. Inoltre, essendo uscito da $\mathcal{T}$, il nodo c non esegue più alcuna diffusione di messaggi multicast. I nodi figli di c, non ricevendo più alcun messaggio multicast, entro un

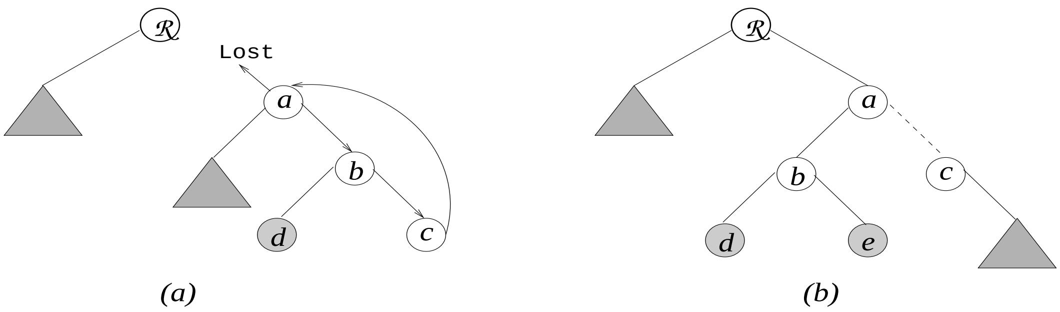


Figura 26: Riconnessione a $\mathcal{T}$ di destinazioni disconnesse.

tempo finito osserveranno scadere il proprio $T_{refresh}$ e a propria volta reinizializzeranno il proprio TD e usciranno dall'albero (linee 21-23 Appendice B). Applicando le medesime considerazioni ai discendenti dei figli di c si dimostra che entro un tempo finito tutti i nodi nel sottoalbero con radice in c escono da $\mathcal{T}$, e perciò i relativi rami scompaiono da $\mathcal{T}$.

Caso (*ii*): cammini che costituiscono rami migliori di quelli esistenti possono essere usati se si devono appendere nuovi riceventi a sottoalberi già esistenti; in tal caso, un nodo riceve una `Build` da un nuovo padre che lo connette a $\mathcal{R}$ con un cammino migliore, e perciò rinuncia al vecchio padre e ai vecchi riceventi (procedura *Loop-Solve*, linea 11.4). I successivi messaggi multicast diffusi non saranno perciò indirizzati ai vecchi riceventi, che entro tempo finito osserveranno la scadenza del proprio $T_{refresh}$ ed usciranno dall'albero, per le medesime considerazioni di cui al caso precedente. Per argomenti analoghi a quelli usati nella dimostrazione del Lemma 5.4.3, inoltre, il vecchio nodo padre sa di aver perso tale ruolo entro un tempo finito. □

Si dimostra ora che le destinazioni disconnesse dall'albero che si trovano nella medesima partizione di $\mathcal{R}$ vengono riconnesse a $\mathcal{T}$ entro un tempo finito.

Lemma 5.4.5 *Sia p_r un ricevente disconnesso da $\mathcal{T}$ nella stessa partizione di $\mathcal{R}$, e supponiamo che non si verifichino fallimenti transienti o cambi di topologia. Allora entro un tempo finito p_r riceve un pacchetto di* `Build` *che lo riconnette all'albero.*

DIM. Si consideri l'esempio mostrato in Figura 26(b), in cui il sottoalbero con radice in c si disconnette dalla radice, e dimostriamo che tutte le destinazioni nel sottoalbero prima o poi vengono riconnesse ad esso; sia p_r una di tali destinazioni. Dai Lemmi 5.4.1 e 5.4.2 segue che entro tempo finito $\mathcal{R}$ viene a conoscenza della disconnessione di questi riceventi; di conseguenza manda dei pacchetti di `Build` per riconnetterli. Si possono verificare tre casi: **(1)** la `Build` è ricevuta a p_r; **(2)** si forma un ciclo; **(3)** si crea un nodo dell'albero con due padri. I casi rilevanti sono (2) e (3); esaminiamoli separatamente.

Caso 2: dal Lemma 5.4.3, entro un tempo finito il ciclo è distrutto; poichè p_r di conseguenza non ha un genitore nell'albero, dal protocollo continua l'invio di messaggi *bubble* (linee 38-44 Appendice B) e $\mathcal{R}$ entro tempo finito noterà che p_r è ancora disconnesso per il Lemma 5.4.2. Perciò verrà generata una nuova `Build` per p_r e ad essa si applica la medesima dimostrazione. La formazione di un ciclo è conseguenza dell'inconsistenza nelle TU di SM diverse oppure della mobilità di una SM (Figura 23(b)). La prima condizione prima o poi non è più vera per l'assunzione fatta sulla convergenza dell'algoritmo di instradamento unicast (proprietà 5.2.2); la seconda non può verificarsi per l'ipotesi che la topologia non cambi. Poichè si è assunto che non si verifichino fallimenti transienti, la nuova `Build` raggiungerà p_r, che verrà appeso a $\mathcal{T}$.

Caso 3: dal Lemma 5.4.3 entro un tempo finito il nodo sceglie uno dei due padri. Se sceglie il nuovo padre, p_r entra nell'albero. Altrimenti, significa che la `Build` ha individuato un cammino peggiore (in numero di hop) del precedente; questo può essere causato da un'inconsistenza nelle TU, ed in tal caso si applicano le medesime considerazioni del caso precedente. Oppure, si è verificata la situazione mostrata in Figura 23(c); dal Lemma 5.4.4 i cammini obsoleti sono rimossi entro tempo finito. Nel frattempo, p_r non ha ancora un genitore nell'albero e ad esso si applicano le medesime considerazioni fatte nel caso precedente. □

Corollario 5.4.1 [Liveness(1)] *Se prima o poi la topologia della rete si stabilizza, allora il protocollo MSTP garantisce che entro un tempo finito tutte le destinazioni in $\mathcal{G}$ nella stessa partizione di $\mathcal{R}$ sono connesse all'albero.*

DIM. La dimostrazione segue applicando il risultato del Lemma 5.4.5 a tutti i riceventi in $\mathcal{G}$, con le seguenti osservazioni per quanto riguarda la soluzione dei fallimenti transienti:

1. se il fallimento riguarda un pacchetto di `Lost`, nel Lemma 5.4.2 si dimostra che entro un tempo finito $\mathcal{R}$ conosce le identità dei riceventi disconnessi;

2. se il fallimento riguarda un pacchetto di `Build`, allora il ricevente a cui la `Build` era destinata resta senza padre nell'albero ed entro tempo finito riprende a mandare pacchetti *bubble* (procedura *Complain*). Per il Lemma 5.4.2 entro un tempo finito $\mathcal{R}$ si accorge che il ricevente è ancora disconnesso e manda una nuova `Build`. Per la proprietà 5.2.1, entro un numero finito ρ di tentativi la `Build` è ricevuta dal ricevente, se questo rimane nella medesima partizione di $\mathcal{R}$.

□

Lemma 5.4.6 [Liveness(2)] *Il protocollo MSTP è esente da deadlock.*

DIM. La condizione di deadlock è verificata nel caso esista un insieme di processi in attesa circolare. L'enunciato è verificato grazie alle seguenti considerazioni, che derivano dalla descrizione del protocollo (Appendice B):

1. le SM forwarder non restano mai bloccate in attesa di un evento esterno, ma solo eventualmente in attesa della scadenza del locale $T_{refresh}$;

2. analoga considerazione vale per le SM riceventi;

3. $\mathcal{R}$ resta in attesa di messaggi multicast da diffondere per eseguire le operazioni di manutenzione di $\mathcal{T}$. Se nessun messaggio è generato il sistema non è in deadlock perchè in effetti il protocollo non è attivo[3].

□

Teorema 5.4.1 *Il protocollo MSTP risolve correttamente il problema dell'instradamento multicast enunciato nella Definizione 2.3.1.*

DIM. I Lemmi 5.4.3, 5.4.4, 5.4.6 e il Corollario 5.4.1 dimostrano che MSTP soddisfa le proprietà enunciate nella definizione. □

5.5 Valutazione delle prestazioni

5.5.1 Ambiente di simulazione

Il protocollo MSTP è stato implementato sopra il simulatore NETSIMPRO (sezione 3.4) allo scopo di valutarne le prestazioni. La rete considerata è costituita da $n = 100$ SM che si muovono lungo traiettorie casuali su un'area di $500 \times 500\ mt^2$, ad una velocità predefinita compresa tra 0 e 80 Km/h.

[3]Ed in questo caso non è rilevante che $\mathcal{T}$ sia disconnesso, perchè non deve essere usato per la diffusione di alcun messaggio multicast.

Il raggio di un cluster è di 120 metri e la larghezza di banda è di 2Mbps; questi parametri caratterizzano una rete wireless LAN. I pacchetti di controllo di MSTP sono scambiati nei control slot. Ipotizzando che ogni frame includa 20 data slot, la durata del frame è pari a 0.21 secondi.

Sotto queste ipotesi, il 70% della banda è riservata per gli slot di controllo. Si noti che in ogni cluster sono riservati n slot di controllo; poichè nel nostro sistema ogni cluster in media contiene 10 SM, circa il 90% della banda di controllo (il 63% dell'intera banda) è sprecato, limitando perciò la crescita del throughput. In futuro si progetta di adottare un differente protocollo di livello MAC, come quello proposto nello standard IEEE 802.11 [35], per poter sfruttare la banda di controllo inutilizzata per l'invio di traffico multicast.

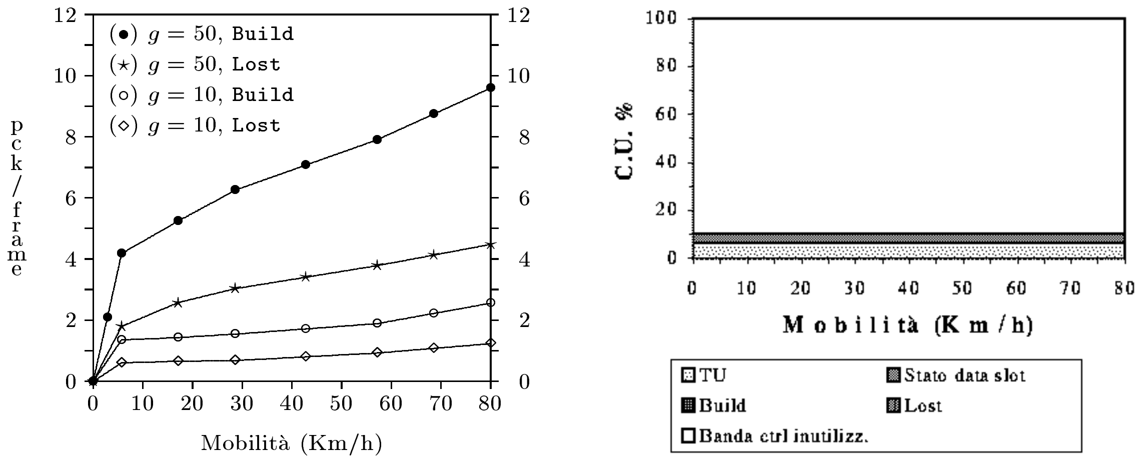
Negli esperimenti si è considerato un pattern di traffico "*uno-a-molti*", costituito da una sorgente di messaggi multicast situata alla SM $\mathcal{R}$ e da un numero g di destinazioni variabile tra 10 e 50. I membri del gruppo sono scelti a caso tra tutte le SM del sistema. La massima frequenza di generazione di messaggi multicast è 1 pck/frame, che corrisponde circa a 5 pck/sec. Il tempo totale di simulazione è di 5000 frame, che corrispondono a circa 1000 secondi di tempo simulato.

5.5.2 Risultati di simulazione

Messaggi di controllo

In Figura 27(a) si mostra il numero medio di pacchetti di controllo, `Build` e `Lost`, generati in ogni frame per la manutenzione dell'albero condiviso. Si sono considerati gruppi con 10 e con 50 riceventi. I gruppi più numerosi sono maggiormente influenzati dalla mobilità, perchè è più probabile che una SM che si sposta appartenga all'albero e che di conseguenza il suo movimento rompa un ramo. Il numero dei pacchetti di `Build` supera quello dei pacchetti di `Lost`: una `Lost` può informare $\mathcal{R}$ riguardo alla disconnessione di più destinazioni appartenenti allo stesso sottoalbero; perciò, come conseguenza di un singolo pacchetto di `Lost` possono essere generati più pacchetti di `Build`. In tutte le condizioni, il numero di pacchetti di controllo cresce linearmente sia con la velocità che con g.

Con $g = 50$ vengono generati al massimo 14 pacchetti di controllo per frame; perciò almeno lo 86% degli slot di controllo disponibili non contiene informazioni relative al mantenimento dell'albero, e la banda riservata per il traffico di controllo resta inutilizzata la maggior parte delle volte. Questo è confermato dalle misure relative all'utilizzazione del canale per informazioni di controllo (Figura 27(b)). Si definisce la capacità della rete come $\mathcal{C} = C \times B \times T$, dove C è il numero medio di cluster nel sistema, B è la banda attraverso un canale wireless e T è il tempo di simulazione. Il carico del canale dovuto ai pacchetti di tipo P è calcolato come $CU_P = num_P \times size_P/\mathcal{C}$, dove num_P indica il numero di pacchetti di tipo P generati



(a) (b)

Figura 27: (a) Numero di messaggi di controllo generati in funzione della velocità. (b) Utilizzazione del canale per i pacchetti di controllo.

durante il tempo T e $size_P$ è la taglia di un pacchetto di tipo P. Il 6.35% della banda di controllo è usata per scambiare le tabelle di instradamento unicast; il 3.89% è usato per le informazioni riguardanti lo stato degli slot dati. La banda usata per pacchetti di `Build` e `Lost` è trascurabile, meno dello 0.025%, mentre il 90% è inutilizzato.

Efficienza multicast

Come in [24], si definisce l'*efficienza multicast* ME come $\frac{H}{MTX}$, dove H è il numero totale di hop compiuti dai messaggi multicast e MTX è il numero totale di trasmissioni multicast; H è calcolato ai riceventi accumulando il conto del numero di hop compiuti dai messaggi multicast. L'efficienza multicast è una buona misura dell'effienza nell'uso delle risorse di rete da parte del protocollo. Se $ME > 1$ allora sono sufficienti poche trasmissioni multicast (cioè poche risorse di rete) per raggiungere molti riceventi, mentre $ME = 1$ rappresenta il caso di trasmissioni unicast. Infine, $ME < 1$ indica che vengono effettuate trasmissioni ridondanti, ovvero trasmissioni che non raggiungono alcun ricevente, come può accadere se esistono rami rotti nell'albero. Come si nota in Figura 28(a), l'efficienza aumenta con la cardinalità del gruppo perchè il gruppo è più *denso* nel sistema e quindi una sola trasmissione broadcast può raggiungere più destinazioni. L'efficienza multicast è influenzata negativamente dalla velocità delle SM, come mostrato nella sezione successiva, ma la perdita di efficienza è indipendente dalla cardinalità del gruppo. Ponendo a 100 l'efficienza osservata quando le SM non si spostano, in Figura 28(b) si mostra come varia l'efficienza in funzione della velocità per tre differenti cardinalità del gruppo.

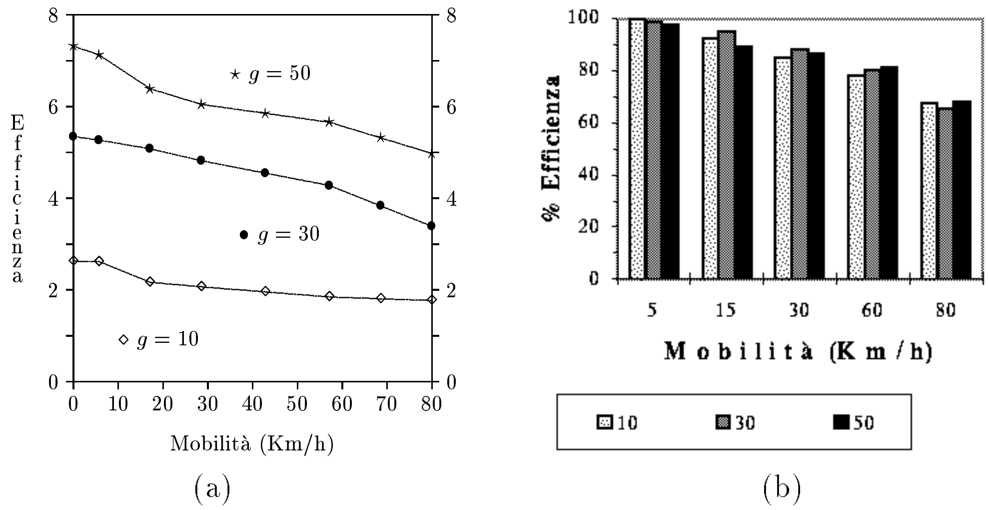


Figura 28: (a) Efficienza multicast rispetto alla mobilità. (b) Percentuale di efficienza multicast in funzione della mobilità e della cardinalità del gruppo.

Throughput

In Figura 29(a) si mostra il throughput ottenuto per diverse cardinalità del gruppo e diverse velocità. La mobilità ha sul throughput gli stessi effetti che ha sull'efficienza multicast; la perdita di throughput è ancora indipendente dalla cardinalità del gruppo, come mostrato in Figura 29(b). Il throughput è la quantità totale di messaggi multicast ricevuti dai membri del gruppo; è calcolato sommando i messaggi ricevuti ai vari riceventi. Il throughput può essere influenzato da: (i) l'eliminazione di pacchetti alle SM per overflow dei buffer; (ii) lo smarrimento di pacchetti a causa di collisioni; oppure (iii) la bassa connettività della struttura ad albero. Il primo problema ha basso impatto sulle prestazioni se si adotta un meccanismo di controllo di congestione alla radice dell'albero. L'impatto del secondo problema dipende dalle caratteristiche del protocollo MAC. Il terzo è dovuto al fatto che, come conseguenza della mobilità delle SM, un sottoinsieme dei riceventi può essere temporaneamente irraggiungibile attraverso l'albero. In un approccio soft-state, questo evento si verifica perchè il periodo ottimale di ricostruzione dell'albero è difficile da determinare. Nell'approccio ibrido adottato da MSTP, la ridotta connettività è dovuta alla possibile diffusione di un messaggio multicast durante la propagazione di una `Lost` verso $\mathcal{R}$; l'impatto di questo evento è tanto maggiore quanto maggiore è la profondità dell'albero, mentre diminuisce al decrescere del carico (Figura 30(a)).

La dipendenza del throughput dalla profondità dell'albero può essere limitata introducendo delle trasmissioni *ridondanti* dei messaggi multicast quando l'albero viene ricostruito, solo alle destinazioni che si erano disconnesse e nei punti in cui è eseguita

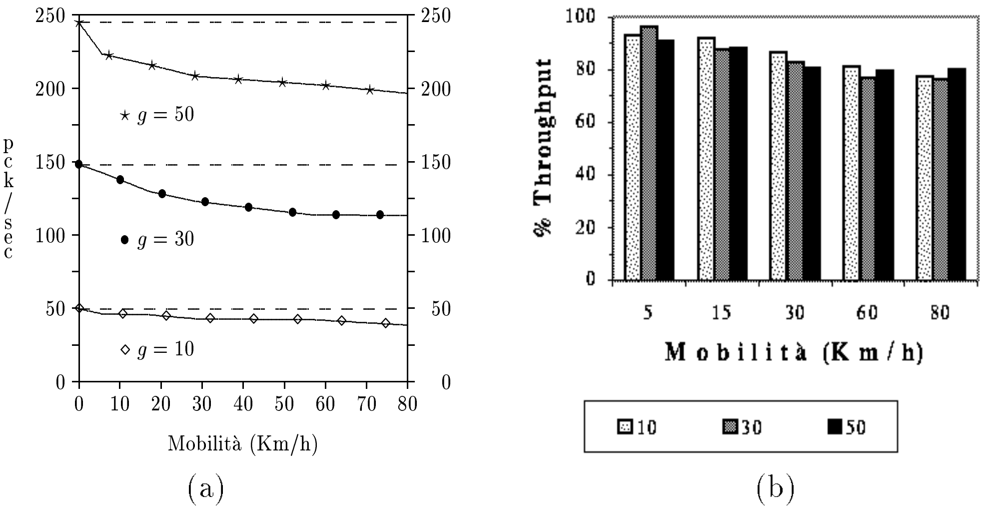
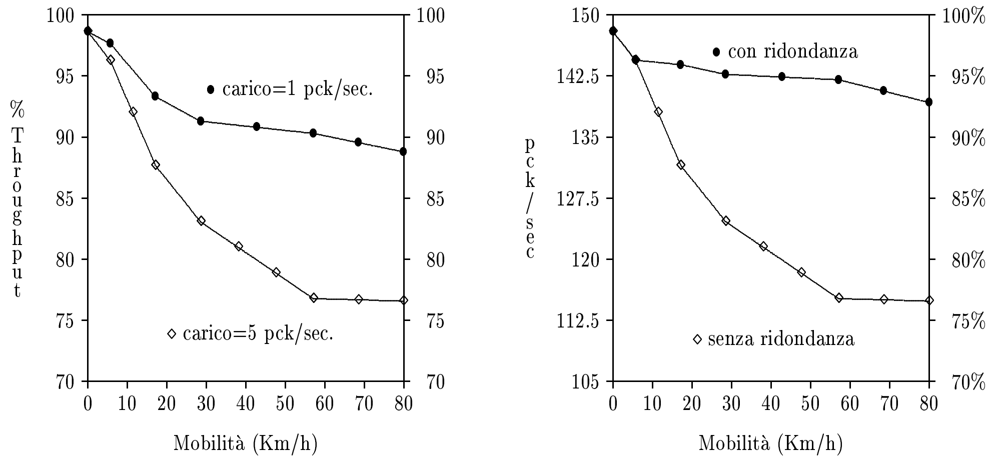


Figura 29: (a) Throughput in funzione della mobilità. (b) Percentuale di throughput in funzione della mobilità e della cardinalità del gruppo.

la riconnessione. Per verificare l'efficacia di questo approccio, si è leggermente modificata l'implementazione di MSTP per mantenere in ogni nodo di $\mathcal{T}$ una copia degli ultimi h messaggi multicast osservati dal nodo, in un buffer circolare. Assumendo un tasso di generazione di un messaggio multicast per frame, h può essere fissato approssimativamente pari a D a $\mathcal{R}$ e a $D - k$ ad un nodo interno p_f, dove D è il diametro dell'albero e k è la distanza tra $\mathcal{R}$ e p_f [57]; negli esperimenti h è stato posto ovunque uguale a 3. Ogni volta che una SM p_f in $\mathcal{T}$ riceve una `Build` che provoca la creazione di un nuovo ramo discendente da p_f per connettere riceventi persi, p_f invia i propri h messaggi su quel ramo. I riceventi scarteranno i pacchetti ridondanti se sono duplicati e li consegneranno all'entità di livello superiore altrimenti. Come mostrato in Figura 30(b), per un gruppo di 30 riceventi e differenti velocità di movimento il ricorso a trasmissioni ridondanti compensa efficacemente la perdita di messaggi multicast mentre i riceventi sono disconnessi. Questo risultato è ottenuto a prezzo di una maggiore occupazione di memoria, benchè il costo diminuisca al crescere della distanza da $\mathcal{R}$. L'efficienza multicast non è influenzata dal meccanismo di ridondanza; infatti, i messaggi ridondanti sono trasmessi solo lungo i rami appena costruiti, perchè gli altri discendenti a un hop dalla sorgente p_f della trasmissione ridondante individuano il messaggio duplicato e lo scartano senza propagarlo ulteriormente.

5.6 Conclusioni

In questo capitolo si è descritto il protocollo originale MSTP per la costruzione di un'infrastruttura ad albero condiviso per l'instradamento di messaggi multicast. Si



(a) (b)

Figura 30: (a) Throughput in funzione della velocità in un gruppo di 30 riceventi e sotto differenti condizioni di carico. (b) Throughput con e senza ridondanza (30 riceventi in $\mathcal{G}$).

è dimostrato che il protocollo risolve correttamente il problema dell'instradamento multicast definito in sezione 2.3 e se ne è valutato il comportamento mediante tecniche di simulazione.

L'approccio ibrido utilizzato dal protocollo per la manutenzione della struttura permette di eseguire la ricostruzione dei soli rami danneggiati da cambiamenti topologici, solo nel momento in cui la struttura deve essere utilizzata per la diffusione di un messaggio multicast. Il protocollo ingloba un meccanismo per il controllo della congestione al nodo radice. Il carico di rete del protocollo è trascurabile, ed è richiesta solo una limitata quantità di memoria per conservare le informazioni necessarie.

In futuro si progetta di effettuare ulteriori valutazioni delle prestazioni di MSTP utilizzando un protocollo di livello MAC che consenta un più efficiente uso della banda. Inoltre si modificherà il protocollo così da permettere l'esecuzione delle procedure di mantenimento dell'albero anche ai nodi interni, e si confronteranno queste due versioni di MSTP con protocolli proposti in letteratura che adottano gli approcci hard-state e soft-state.

Capitolo 6

Conclusioni e sviluppi futuri

6.1 Considerazioni finali

In questa tesi si sono proposti due protocolli originali per la soluzione in reti wireless ad-hoc dei fondamentali problemi di comunicazione di gruppo: il problema del *Reliable Broadcast* e il problema dell'*instradamento multicast*. Si è dimostrato che i protocolli proposti risolvono correttamente i due problemi nei sistemi considerati. Le prestazioni dei due protocolli sono state valutate mediante tecniche di simulazione; nel caso del protocollo WRBcast si è effettuato il confronto con altri protocolli proposti in letteratura che possono essere impiegati per la soluzione dei medesimi problemi.

Il protocollo WRBcast realizza un propria struttura di instradamento e garantisce la consegna affidabile di messaggi broadcast a tutte le SM del sistema. Tale protocollo può essere utilizzato anche quando i cambiamenti topologici siano tanto frequenti da rendere inefficiente la costruzione di una struttura di instradamento multicast. Inoltre, poichè esso può essere collocato ad un basso livello nell'architettura, il servizio di Reliable Broadcast da esso fornito può essere sfruttato per semplificare la costruzione di altri protocolli di basso livello. I risultati di simulazione sono incoraggianti in confronto alle prestazioni ottenute ricorrendo al flooding o all'invio punto-punto di copie dei messaggi. Gli esperimenti mostrano che il punto debole del protocollo è costituito dalla congestione dei clusterhead a causa dell'implosione di messaggi ack; questo problema potrebbe essere risolto adottando una differente strategia per il protocollo di livello MAC o progettando un appropriato protocollo per il controllo della congestione.

Il protocollo MSTP costruisce un albero di instradamento condiviso, adottando un approccio ibrido on-demand per il suo mantenimento. Le caratteristiche di MSTP consentono di ottenere una buona scalabilità a basso costo, come mostrato dai risultati di simulazione. Gli esperimenti condotti evidenziano come le prestazioni dipendano in qualche misura dalla profondità dell'albero; per ovviare a questo inconveniente è possibile ricorrere ad algoritmi per la periodica rilocazione del nodo radice, quali quelli proposti in [18, 34], che possono anche consentire di ottenere la tolleranza ai fallimenti della radice. Un'altra possibilità consiste nel permettere ai nodi interni dell'albero di eseguire le procedure di manutenzione. Ulteriori misure sono necessarie per valutare l'utilità di questo meccanismi, come pure del meccanismo per il controllo di congestione.

6.2 Sviluppi futuri

I protocolli proposti possono essere modificati, così da migliorarne le prestazioni, sfruttando la eventuale presenza nel sistema di stazioni più affidabili delle altre o con maggiori risorse, o aventi minore probabilità di movimento o disconnessione, quali ad esempio le stazioni fisse. Ad esempio, la scelta di una stazione fissa come nodo radice dell'albero di instradamento multicast costruito dal protocollo MSTP ridurrebbe la probabilità di congestione della stessa o di fallimento con conseguente disconnessione dell'albero. Queste modifiche potrebbero essere successivamente estese in modo da fornire l'interoperabilità tra sottoreti wired e wireless, e consentire l'accesso a Internet da parte di SM.

Ulteriori miglioramenti potrebbero essere ottenuti adattando i due protocolli alle particolari caratteristiche dei servizi di più basso livello: si è notato dalle simulazioni l'influenza delle caratteristiche del protocollo di livello MAC sui risultati degli esperimenti effettuati. La conoscenza del pattern di traffico generato dalle applicazioni può anch'essa essere sfruttata per ottenere migliori prestazioni; ad esempio, nel caso di un pattern uno-a-molti potrebbe essere vantaggioso far coincidere la radice dell'albero costruito da MSTP con la sorgente.

L'aspetto più interessante riguarda il possibile uso dei protocolli proposti per la realizzazione di servizi che risolvano problemi più complessi. Sembra essere abbastanza semplice combinare i due protocolli per realizzare un servizio di *Reliable Multicast*; si sono già iniziati a condurre esperimenti in tal senso. Il servizio di Reliable Multicast è interessante perchè, dato l'alto numero di stazioni prevedibilmente connesse in sistemi wireless, è probabile che i gruppi di utenti cooperanti siano sottoinsiemi di tutti gli utenti connessi. Come si è osservato nel capitolo 4, il protocollo WRBCAST non è conveniente per la consegna affidabile di messaggi a gruppi sparsi, dal punto di vista del consumo sia di banda sia di energia per l'invio di messaggi a SM che non ne sono destinatarie e che quindi li scartano. Sostituendo il forwarding tree costruito da WRBCAST con l'albero di instradamento costruito da MSTP è possibile dirigere i messaggi solo alle destinazioni interessate e al contempo garantire l'affidabilità. Ovviamente, il protocollo WRBCAST deve essere modificato in modo tale che le informazioni di stato rilevanti e le procedure per il recupero dai malfunzionamenti risiedano solo nei nodi appartenenti all'albero condiviso.

Il protocollo WRBcast può essere adattato per fornire servizi a valore aggiunto di consegna *ordinata* di messaggi, come richiesto dai problemi di FIFO, Causal e Atomic Broadcast [44]. Nei primi due casi è forse possibile ottenere algoritmi corretti applicando direttamente a WRBcast le trasformazioni descritte in [43] per effettuare riduzioni tra problemi. Nel terzo caso il discorso è più complesso: il problema di Atomic Broadcast è stato formalmente dimostrato equivalente al problema del Consenso [44], ed entrambi sono impossibili da risolvere in sistemi asincroni in presenza di fallimenti crash [36]. Ulteriori ostacoli potrebbero poi derivare dalla possibilità di disconnessione delle SM: una disconnessione può essere interpretata come un fallimento transiente; in [39, 63, 65] sono riportati risultati che evidenziano come la possibilità di questo genere di fallimenti renda più complessa la soluzione di problemi di comunicazione affidabile di gruppo o restringa gli ambiti in cui essa è possibile. È quindi necessario procedere prima alla formalizzazione del modello di sistema per l'ambiente mobile ed indagare sulla risolubilità del problema di Atomic Broadcast in questo modello, sulla traccia di quanto fatto nel caso dei sistemi fissi.

Appendice A

Specifica del protocollo WRBcast

Nelle Tabelle alle pagine seguenti si descrive il protocollo WRBcast per mezzo di macchine a stati finiti, per unn clusterhead, un gateway ed un host regolare. Nelle Figure 31, 32 e 33 le tre tabelle sono introdotte per mezzo di automi semplificati. Le macchine a stati descrivono le azioni eseguite dalle SM per il broadcast dii un singolo messaggio m; la gestione concorrente di altri messaggi non è rappresentata per ragioni di semplicità, benchè sia supportata dal protocollo. Questo ha due conseguenze:

1. la procedura per la gestione dei `cluster-ack` è attivata non appena viene ricevuto un messaggio. In genere, essa è eseguita autonomamente, indipendentemente dall'arrivo di messaggi;

2. l'informazione riguardo alla stabilità globale di m è sempre distribuita per mezzo di un broadcast separato del vettore *stable*, mentre in generale il vettore è diffuso insieme ad un successivo messaggio broadcast.

Analogamente, la macchina a stati dei gateway è stata semplificata ipotizzando che il gateway connetta due soli clusterhead, ch e ch'. Nella macchina a stati dei gateway si descrive solo la diffusione dei messaggi broadcast e dei `cluster-ack` tra cluster adiacenti. Parallelamente un gateway esegue la macchina a stati degli host regolari.

In Figura 31 un clusterhead passa dallo stato IDLE allo stato CH-SEND o quando riceve un messaggio broadcast da diffondere generato da una SM nel suo cluster, oppure quando genera esso stesso un messaggio. Nello stato CH-SEND un clusterhead ritrasmette m finchè non ha raccolto tutti gli ack locali per m. Quando il clusterhead conosce la stabilità locale di m, passa allo stato STABILIZE, in cui attende di conoscere la stabilità globale di m; in questo stato il clusterhead invia i propri messaggi di `cluster-ack` o `flood-ack` per m. Quando il clusterhead riceve un vettore *stable* che indica la stabilità globale di m, ritorna allo stato IDLE. Lo stato HANDLE-MOB descrive le azioni intraprese per gestire l'ingresso di una nuova SM nel cluster, ovvero lo scambio di stato tra il clusterhead e la SM. Un host regolare che viene eletto

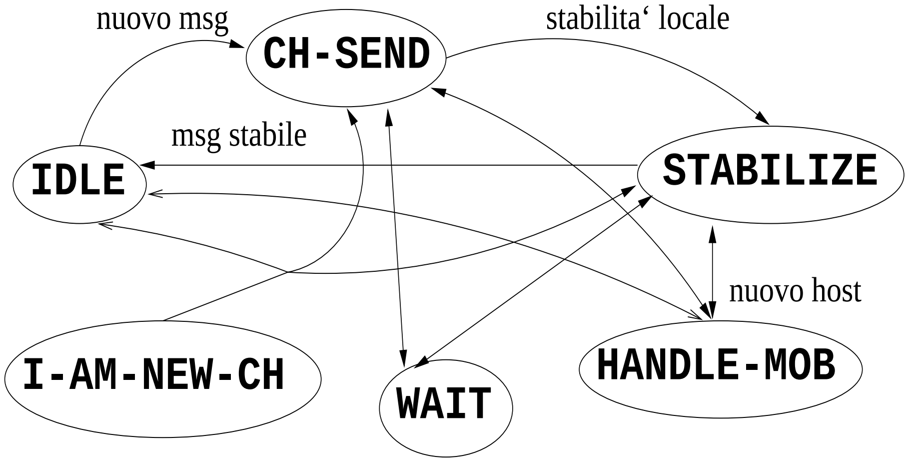


Figura 31: Automa dei clusterhead.

clusterhead passa dall'automa per gli host regolari (Figura 33) allo stato I-AM-NEW-CH dell'automa del clusterhead. Questo stato descrive lo scambio di informazioni di stato del nuovo clusterhead con le SM locali e con gli altri clusterhead. Un clusterhead *ch* passa allo stato di WAIT quando il clkusterhead originatore di m scompare dal sistema e *ch* non sa a chi indirizzare i propri messaggi di `cluster-ack` o `flood-ack` per m; in questo stato *ch* marca m come 'sospeso' e attende di conoscere quale clusterhead assumerà il ruolo di originatore per m.

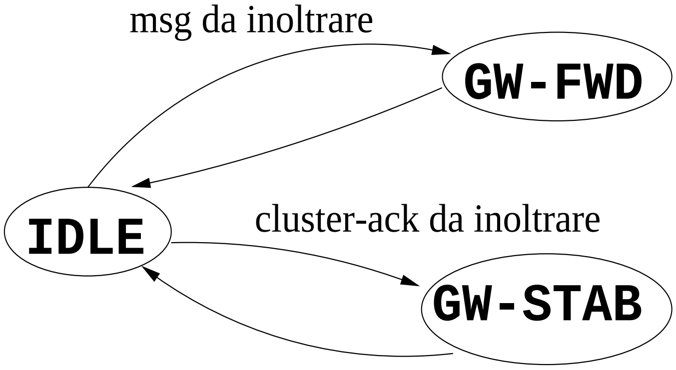


Figura 32: Automa dei gateway.

In Figura 32, un gateway *gw* passa dallo stato IDLE allo stato GW-FWD quando riceve un messaggio broadcast m da inviare agli altri clusterhead adiacenti; in questo stato *gw* aspetta di ricevere un ack per m da ogni clusterhead a cui ha inviato m, per poi inviare il proprio ack al clusterhead predecessore e tornare allo stato IDLE. Analogamente, la transizione dallo stato IDLE allo stato GW-STAB avviene quando *gw* deve inoltrare un `cluster-ack`; *gw* ritorna allo stato IDLE dopo aver inviato l'ack relativo al messaggio di `cluster-ack`.

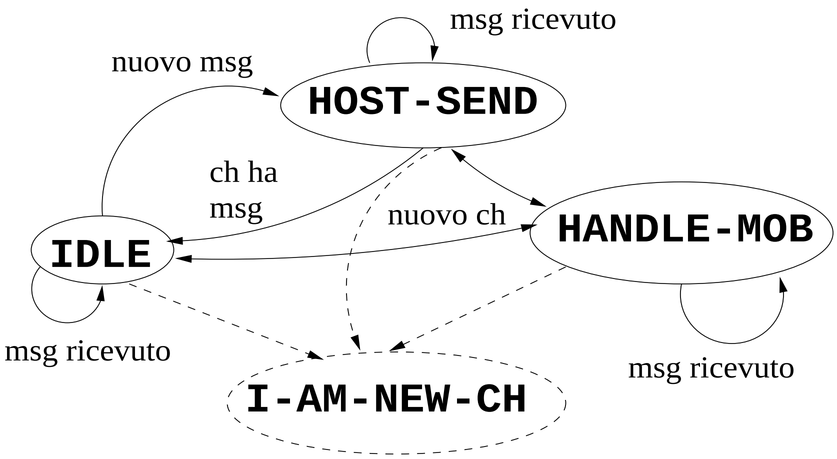


Figura 33: Automa degli host regolari.

In Figura 33 un host regolare p passa dallo stato IDLE allo stato HOST-SEND quando genera un messaggio m e lo manda al proprio clusterhead per la diffusione; in questo stato p ritrasmette il messaggio finchè riceve il broadcast di m dal clusterhead. A questo punto, p sa che il clusterhead diffonderà m, e torna allo stato IDLE. Un host regolare passa allo stato HANDLE-MOB quando ha un nuovo clusterhead; questo stato descrive le azioni per lo scambio di informazioni di stato con il clusterhead.

Macchina a stati Reliable Broadcast Clusterhead

STATI	
IDLE	il clusterhead non sta diffondendo messaggi e tutti i precedenti messaggi sono "stabili"
CH-SEND	il clusterhead sta diffondendo un messaggio *m* e attende gli ack locali
STABILIZE	il clusterhead ha ricevuto tutti gli ack locali e sta inviando cluster-ack al clusterhead originatore di *m*
HANDLE-MOB	il clusterhead sta gestendo un nuovo host entrato nel cluster locale
I-AM-NEWCH	l'host e` stato eletto clusterhead; sta collezionando informazioni di stato prima di stabilizzare messaggi
WAIT	il clusterhead sta aspettando che venga ripresa la diffusione di un messaggio parzialmente diffuso

EVENTI IN INPUT	
RBCASTRq	richiesta del servizio di Reliable Broadcast dall'entita` di livello superiore
MSGrecvInd	ricezione di un messaggio unicast
ACKrecvInd	ricezione di un ack locale per un messaggio o un cluster-ack
CLACKrecvInd	ricezione di un messaggio di cluster-ack inoltrato da un gateway
FLOODbcastInd	ricezione di un messaggio di flood-ack
STABLEbcastInd	ricezione di un messaggio di stabilita`
MOVErecvInd	ricezione di un messaggio di stato da un nuovo host entrato nel cluster locale
NEWCHbcastInd	ricezione di un messaggio di stato da un clusterhead, quando un nuovo clusterhead e` stato eletto
L_NEWHOSTInd	il livello link segnala l'ingresso di un nuovo host nel cluster locale

EVENTI IN OUTPUT	
DELIVERInd	notifica della ricezione di un messaggio all'entita` di livello superiore
MSGsendRq	trasmissione unicast di un messaggio
MSGbcastRq	trasmissione broadcast di un messaggio
ACKsendRq	trasmissione di un ack locale al gateway che ha inoltrato un cluster-ack
CLACKbcastRq	trasmissione broadcast di un cluster-ack
FLOODbcastRq	trasmissione broadcast di un flood-ack
STABLEbcastRq	trasmissione broadcast di un messaggio di stabilita`
MOVEsendRq	trasmissione unicast dello stato del clusterhead a un nuovo host entrato nel cluster locale
NEWCHbcastRq	trasmissione dello stato del clusterhead, quando un nuovo clusterhead e` stato eletto

TIMER	
T_p	tempo di attesa di un ack per messaggi unicast
T_b	tempo di attesa di un ack per messaggi broadcast
T_ack	periodo di attivazione della procedura che elabora i messaggi di ack in uscita

PREDICATI	
P1	il messaggio *m* e` nella coda *unstable* di *ch*
P2	il sender di *m* non e` nel cluster di *ch*
P3	l'identificatore di *m* e` incluso nella *stable*
P4	il messaggio *m* e` stabile per *ch*
P5	*ch* ha ricevuto tutti gli ack locali per *m*
P6	o la *hold_back* o la coda di uscita di *ch* contengono un cluster-ack o un flood-ack per *m*
P7	*ch* ha un successore nel forwarding tree di *m*
P8	il sender di *m* e` nel cluster di *ch* (*ch* e` il clusterhead originatore di *m*)
P9	per il cluster-ack di *ch* per *m* e` stato ricevuto un ack dal predecessore di *ch*
P10	*ch* sa che gli altri clusterhead non conoscono *m*

P11	*ch* ha ricevuto lo stato dagli altri clusterhead
P12	il messaggio *m* e` marcato come “sospeso”
P13	un sottoinsieme dei clusterhead conosce il messaggio *m*
P14	il nuovo clusterhead non conosce il messaggio *m*
P15	il nuovo host nel cluster di *ch* conosce il messaggio *m*
P16	ack per *m*
P17	ack al cluster-ack per *m* mandato da *ch*
P18	il cluster-ack per *m* non e` stato convertito in un flood-ack
P19	il nuovo host entrato nel cluster di *ch* era un clusterhead
P20	il messaggio *m* e` ricordato nella *stable* del nuovo host

AZIONI	
[1]	aggiungi messaggio *m* alla *unstable* di *ch*
[2]	*m.sorgente* $\leftarrow$ *ID ch; m.tipo* $\leftarrow$*bcast; m.dest* $\leftarrow$*all; m.ID-originatore* $\leftarrow$ *ID ch; m.ID-predecessore* $\leftarrow \perp$
[3]	scarta duplicato del messaggio *m*
[4]	marca il messaggio *m* come stabile
[5]	togli al messaggio *m* il marchio “sospeso”
[6]	aggiungi ack alla bitmap per *m* nella *unstable* di *ch*
[7]	aggiungi un cluster-ack per *m* nella *hold_back* di *ch*
[8]	cancella il cluster-ack per il messaggio *m*
[9]	fondi gli ack per *m*
[10]	poni il messaggio di ack ricevuto nella *hold_back* di *ch*
[11]	*m.ID-sender* $\leftarrow$ *ID ch; m.seq#* $\leftarrow$*numero sequenza ch*
[12]	ricorda *m.ID-predecessore* nella *unstable* di *ch* ; *m.ID-sorgente* $\leftarrow$ *ID ch; m.tipo* $\leftarrow$*bcast ; m.dest* $\leftarrow$*any*
[13]	set T_ack
[14]	sposta i messaggi nella *hold_back* alla coda di uscita di *ch*
[15]	marca il cluster-ack come spedito
[16]	converti il cluster-ack spedito in un flood-ack e mettilo nella coda di uscita di *ch*
[17]	aggiorna lo stato di *ch* in accordo allo stato degli altri clusterhead e aggiorna la *unstable* di *ch*
[18]	marca il messaggio *m* come “sospeso”
[19]	set T_p
[20]	set T_b
[21]	fondi lo stato del nuovo host entrato nel cluster di *ch* con lo stato di *ch*
[22]	reset T_ack
[23]	reset T_p
[24]	reset T_b

	IDLE	CH-SEND	STABILIZE	HANDLE-MOB	I-AM-NEWCH	WAIT
RBCASTRq	[1],[2],[11],[13] DELIVERInd MSGbcastRq [20],CH-SEND					
MSGrecvInd	[1], [13] P8: [2] $\neg$P8: [12] DELIVERInd MSGbcastRq [20], CH-SEND	[3], ACKsendRq	[3],ACKsendRq	[3], ACKsendRq	[3], ACKsendRq	P12: [20], [5] MSGbcastRq CH-SEND $\neg$P12:[3], ACKsendRq
ACKrecvInd		[6] P5$\wedge$(P6$\vee\neg$P7): [9], [20] CLACKbcastR	P17$\wedge$P18: [8]	P17$\wedge$P18: [8] P16: [6]	P17$\wedge$P18: [8] P16: [6]	

		P5∧¬P6∧P7:[7] STABILIZE				
CLACK recvInd		[10], ACKsendRq	ACKsendRq ¬P9: [9] P9: CLACKbcastRq	ACKsendRq ¬P5: [10] P5: [9]	ACKsendRq ¬P5: [10] P5: [9],	[10], ACKsendRq
FLOOD bcastInd		[10]	¬P9: [9] P9: FLOODbcastRq	¬P5: [10] P5: [9]	¬P5: [10] P5: [9]	[10]
STABLE bcastInd			P3: [4], [24],[22] STABLEbcastRq IDLE	P3: [4], [22]	P3: [4]	
MOVErecvInd				P19: [21], [23] P20: [4], [22], [23],STABLEbcastRq, IDLE ¬P15∨¬P5: MSGsendRq,[23] [20], CH-SEND P15∧P5∧(P6∨¬P7)∧P9: [20],[23] CLACKbcastRq, STABILIZE P15∧P5∧(P6∨¬P7)∧¬P9: [9],[23] STABILIZE P15∧P5∧¬P6∧P7: [9],[7],[23] STABILIZE	P15: [9] P19: [21]	
NEWCHbcastInd	NEWCHbcastRq	NEWCHbcastRq P14∧¬P2: [20] MSGbcastRq, CH-SEND ¬ P14∨P2: [18], [24], WAIT	NEWCHbcastRq P4: [4], [24],[22] STABLEbcastRq IDLE ¬P4∧P14∧¬P2: MSGbcastRq, [20], CH-SEND ¬P4∧(¬P14∨P2): [18], [24], WAIT	NEWCHbcastRq P4: [4],[22], [23], [24],STABLEbcastRq, IDLE ¬P4∧P14∧¬P2: MSGbcastRq,[23] [20], CH-SEND ¬P4∧(¬P14∨P2): [18], [23], [24], WAIT	P11: [17] P4: [4],[22], [24],STABLEbcastRq, IDLE ¬P4∧P10∧P13∧P1:MSGbcastRq,[20],CH-SEND ¬P4∧P10∧P13∧¬P1:[18],[24] WAIT	NEWCHbcastRq P4: [4],[22], [24], STABLEbcastRq, IDLE ¬P4∧P14∧¬P2: MSGbcastRq, [20], CH-SEND ¬P4∧(¬P14∨P2): [18], [24], WAIT
L_NEWHOSTInd	MOVEsendRq, [19], HANDLE-MOB	MOVEsendRq [19], HANDLE-MOB	MOVEsendRq [19], HANDLE-MOB	MOVEsendRq [19]	MOVEsendRq, [19]	MOVEsendRq [19], HANDLE-MOB
T_ack		[14], [16] CLACKbcastRq FLOODbcastRq [15], [13]	[14], [16] CLACKbcastRq FLOODbcastRq [15], [13]	[14], [16], CLACKbcastRq FLOODbcastRq [15], [13]	[14], [16], CLACKbcastRq FLOODbcastRq [15], [13]	
T_p				MOVEsendRq, [19]	MOVEsendRq, [19]	
T_b		MSGbcastRq, [20]	[16],[24]		NEWCHbcastRq, [20]	

Macchina a stati Reliable Broadcast
Host regolare

STATI	
IDLE	lo host *p* non sta diffondendo un messaggio e tutti i suoi messaggi precedenti sono stati consegnati
HOST-SEND	lo host *p* sta mandando un nuovo messaggio *m* al proprio clusterhead
HANDLE-MOB	lo host *p* si e\` spostato in un cluster differente

EVENTI IN INPUT	
RBCASTRq	richiesta del servizio di Reliable Broadcast dall'entita\` di livello superiore
MSGrecvInd	ricezione di un messaggio unicast
MSGbcastInd	ricezione di un messaggio broadcast
ACKrecvInd	ricezione di un ack per un messaggio
STABLEbcastInd	ricezione di un messaggio di stabilita\`
MOVErecvInd	ricezione di un messaggio di stato inviato da un nuovo clusterhead
C_NEWCHInd	il livello di clustering segnala che lo host ha un nuovo clusterhead
C_YOUARECHInd	il livello di clustering segnala che lo host e\` stato eletto clusterhead

EVENTI IN OUTPUT	
DELIVERInd	notifica della ricezione di un messaggio all'entita\` di livello superiore
MSGsendRq	trasmissione unicast di un messaggio
ACKsendRq	trasmissione di un ack locale al clusterhead
MOVEsendRq	trasmissione unicast dello stato dello host al nuovo clusterhead

TIMER	
T_p	tempo di attesa dello ack per messaggi unicast

PREDICATI	
P1	il clusterhead non conosce il messaggio *m* generato da *p*
P2	ricezione messaggio non generato da *p*

AZIONI	
[1]	*m.ID-sender ← ID p; m.seq# ← numero sequenza p; m.dest ← ID clusterhead di p*
[2]	aggiungi messaggio *m* nella *unstable* di *p*
[3]	set T_p
[4]	*p* marca come “stabile” tutti i messaggi il cui identificatore e\` in*stable* e aggiorna la propria *unstable*
[5]	reset T_p

	IDLE	HOST-SEND	HANDLE-MOB
RBCASTRq	[1], [2], [3], MSGsendRq, HOST-SEND		
MSGrecvInd	[2], DELIVERInd, ACKsendRq	[5], DELIVERInd, IDLE	DELIVERInd P2: ACKsendRq
MSGbcastInd	[2], DELIVERInd, ACKsendRq	[5], DELIVERInd, IDLE	DELIVERInd P2: ACKsendRq
ACKrecvInd		[5], DELIVERInd, IDLE	DELIVERInd

STABLEbcastInd	[4]	[4]	[4]
MOVErecvInd			[5] ¬P1: IDLE P1: [3], MSGsendRq, HOST-SEND
C_NEWCHInd	MOVEsendRq, [3] HANDLE-MOB	MOVEsendRq, [3], HANDLE-MOB	MOVEsendRq, [3]
C_YOUARECHInd	NEWCHbcastRq, I_AM_NEWCH	NEWCHbcastRq, I_AM_NEWCH	NEWCHbcastRq, I_AM_NEWCH
T_p		MSGsendRq, [3]	MOVEsendRq, [3]

Macchina a stati Reliable Broadcast Gateway

STATI	
IDLE	il gateway *gw* non sta diffondendo un messaggio e tutti i messaggi precedenti sono stati consegnati
GW-FWD	*gw* sta inoltrando a tutti i propri clusterhead vicini un messaggio ricevuto da un clusterhead adiacente
GW-STAB	il gateway *gw* sta inoltrando un cluster-ack da uno dei suoi clusterhead vicini ad un altro

EVENTI IN INPUT	
MSGrecvInd	ricezione di un messaggio unicast da un clusterhead vicino
MSGbcastInd	ricezione di un messaggio broadcast da un clusterhead vicino
ACKrecvInd	ricezione di un ack locale per un messaggio inoltrato o per un cluster-ack inoltrato
CLACKbcastInd	ricezione di un messaggio di cluster-ack da inoltrare
FLOODbcastInd	ricezione di un messaggio di flood-ack

EVENTI IN OUTPUT	
MSGsendRq	trasmissione unicast di un messaggio ai clusterhead vicini
ACKsendRq	trasmissione di un ack locale al clusterhead per la ricezione di un messaggio o inoltro di un ack al clusterhead che ha mandato un cluster-ack
CLACKsendRq	trasmissione unicast di un cluster-ack al clusterhead destinatario
FLOODbcastRq	trasmissione broadcast di un flood-ack a tutti i clusterhead vicini

PREDICATI	
P 1	*gw* ha ricevuto il messaggio *m* dal clusterhead predecessore *ch*
P 2	*ch'* ha scelto *ch* come proprio predecessore
P 3	*gw* ha ricevuto o una copia di *m* o un ack per *m* da ogni clusterhead vicino ≠ *ch*
P 4	*gw* conosce qualche clusterhead *ch'* che ha scelto *ch* come proprio predecessore
P 5	*gw* e` un vicino del clusterhead destinatario
P 6	ack al cluster-ack che *gw* ha precedentemente inoltrato

AZIONI	
[1]	*gw* aggiunge il messaggio *m* nella propria struttura unstable e ricorda il proprio clusterhead predecessore *ch*
[2]	*m.ID-predecessore ← ID ch; m.dest ← ID ch'; m.ID-sorgente ← ID gw; m.tipo ← fwd*
[3]	*ack.sorgente ← ID gw; ack.dest ← ID ch'; ack.ID-predecessore ← ⊥*
[4]	*gw* ricorda *ch'* come successore del proprio predecessore *ch*
[5]	*ack.sorgente ← ID gw; ack.dest ← ID ch; ack.ID-predecessore ← ch'*
[6]	*ack.sorgente ← ID gw; ack.dest ← ID ch; ack.ID-predecessore ← ⊥*

[7]	scarta il messaggio
[8]	*gw* ricorda il clusterhead sorgente del cluster-ack ricevuto
[9]	*ack.dest* ← *ID* del clusterhead che ha mandato il cluster-ack inoltrato da *gw*

	IDLE	GW-FWD	GW-STAB
MSGrecvInd	[1], [2], MSGsendRq, GS-FWD	P1: [2], MSGsendRq ¬P1: [3], ACKsendRq, P2: [4] P3∧P4: [5] P3∧¬P4: [6] ACKsendRq, IDLE	¬P1 : [3], ACKsendRq P1∧¬P4: [6], ACKsendRq P1∧P4: [5], ACKsendRq
MSGbcastInd	[1], [2], MSGsendRq, GS-FWD	P1: [2], MSGsendRq ¬P1: [3], ACKsendRq, P2: [4] P3∧P4: [5] P3∧¬P4: [6] ACKsendRq, IDLE	¬P1 : [3], ACKsendRq P1∧¬P4: [6], ACKsendRq P1∧P4: [5], ACKsendRq
ACKrecvInd		P3∧P4: [5] P3∧¬P4: [6] ACKsendRq, IDLE	¬P6: [7] P6: [9], ACKsendRq, IDLE
CLACKrecvInd	P5: [8], CLACKsendRq, GW_STAB ¬P5: [7]		P5: [8], CLACKsendRq ¬P5: [7]
FLOODbcastInd	FLOODbcastRq	FLOODbcastRq	FLOODbcastRq

Appendice B

Specifica del protocollo MSTP

Nel seguito si riporta lo pseudo-codice dell'algoritmo, a cui fa riferimento la dimostrazione di correttezza. Le strutture dati principali usate sono quelle descritte nel Capitolo 5. Per semplicità, si considera nel seguito il caso di un solo gruppo, e si omette pertanto l'indicazione del suo identificatore $\mathcal{G}$.

Costruzione dell'albero

Di seguito è riportata la procedura di costruzione dell'albero per il nodo radice $\mathcal{R}$ e per una SM p_i, eseguita quando $\mathcal{R}$ riceve un messaggio multicast da diffondere e conosca l'esistenza di riceventi disconnessi. La procedura *Build-Tree* descrive le operazioni eseguite per la costruzione della struttura TD.

<u>$\mathcal{R}$:</u>

when ricevuto "`send` ($\mathcal{G}$, msg)" da un host sorgente **do**
  `send` pacchetto acknowledgement alla sorgente di msg;
  stima $T_{refresh}$;
  if ($IaminG$) **then** `Deliver` (msg);
  if ($S_{\mathcal{R}} \neq \emptyset$) **then**
    Build-Tree {
      $\forall$ elemento in $S_{\mathcal{R}}$ cerca nella corrente TU$_{\mathcal{R}}$ il migliore next-hop per quella destinazione e mettilo in TD$_{\mathcal{R}}$.*discendenti*; rimuovi gli elementi in $S_{\mathcal{R}}$ e fai broadcast della `Build`;
    }
  `multicast` ($\mathcal{G}$, msg, $T_{refresh}$);

<u>host p_i:</u>

when ricevuto `Build` **and** $p_i \in$ `Build`.$TD\_sorgente.discendenti$ **do**
  /* sono padre per un sottoinsieme di dest.; sia k t.c. $p_i =$ `Build`.$TD\_sorgente.discendenti_k$ */
  poni `Build`.$TD\_sorgenti.discendenti_k.lista\_dest_k$ in S_i tranne il mio identificatore;
  $TD_i.genitore \leftarrow$ `Build`.$sorgente$; $TD_i.num\_hop \leftarrow$ `Build`.num_hop;
  `Build`.$num\_hop \leftarrow$ `Build`.$num\_hop + 1$; *Build-Tree*; set $T_{refresh}$;
when ricevuto `multicast` ($\mathcal{G}$, msg, $T_{refresh}$) **do**
  if (io **non** sono su $\mathcal{T}$ per $\mathcal{G}$) **then** scarta msg;

17. **else**
18. **if** ($IaminG$) **then** `Deliver` (msg);
19. set $T_{refresh}$;
20. **if** (io **non** sono un nodo foglia) **then** **`multicast`** ($\mathcal{G}$, msg, $T_{refresh}$);
21. **when** $T_{refresh}$ scade **do**
22. azzera valori in TD_i;
23. **if** (**not** $IaminG$) **then** reset $T_{refresh}$;

Rilevamento delle disconnessioni

Di seguito è riportata la procedura *Check-Tree* per il controllo della connettività dell'albero ed il rilevamento della rottura di rami. Essa comprende anche il trattamento di messaggi di `Lost` che una SM può ricevere dai discendenti, che segnalano l'avvenuta di una disconnessione nel sottoalbero della SM.

host p_i:
24. *Check-Tree* {
25. controlla se $TD_g.genitore$ e $TD_g.discendenti$ sono ancora nodi adiacenti;
26. **if** (padre disconnesso) **then** azzera valori in TD_i; **if** ($\neg$ $IaminG$) **then** reset $T_{refresh}$;
27. **else if** (qualche figlio $TD_g.discendenti_k$ disconnesso **or** ricevuta `Lost` da qualche figlio) **then**
28. $lost\_dests \leftarrow \{TD_g.lista\_dest_k\ \forall TD_g.discendenti_k$ disconnesso$\} \cup \{$`Lost`$.\{dest\ perse\}\ \forall$ `Lost` ricevuta$\}$;
29. `send Lost` ($\mathcal{G}$, $lost\_dests$) al mio genitore;
30. **if** (persi tutti i figli) **then** azzera valori in TD_i; **if** ($\neg$ $IaminG$) **then** reset $T_{refresh}$;
31. }

$\mathcal{R}$:
32. *Check-Tree* {
33. **if** (qualche figlio $TD_{\mathcal{R}}.discendenti_k$ disconnesso) **then** aggiungi a $S_{\mathcal{R}}$ $TD_{\mathcal{R}}.lista\_dest_k$;
34. **if** (ricevuta `Lost` da qualche figlio) **then** aggiungi a $S_{\mathcal{R}}$ `Lost`.$\{dest\ perse\}$;
35. }

Rilevamento dei cicli

La procedura *Loop-Solve* viene eseguita al posto della linea di codice 11 precedente; essa descrive la tecnica per la scelta del nodo genitore da parte di una SM che abbia già un padre nell'albero e riceva un nuovo pacchetto di `Build` da un nodo differente.

11.0 *Loop-Solve* {
11.1 **if** (nuova `Build` arriva da nuovo padre) **then**
/* sia k t.c. $p_i =$ `Build`.$TD\_sorgente.discendenti_k$ */
11.2 **if** ($TD_i.num\_hop <$ `Build`.num_hop) **then**
11.3 scarta `Build`; `send Lost` ($\mathcal{G}$, `Build`.$TD\_sorgente.discendenti_k.lista\_dest_k$) al nuovo padre; END;
11.4 **else** $\forall j$: `send Lost` ($\mathcal{G}$, $TD_i.discendenti_j.lista\_dest_j$) al vecchio padre;
11.5 }

Superamento degli smarrimenti di pacchetti

La procedura *Complain* è eseguita da un nodo ricevente che osservi lo scadere del proprio $T_{refresh}$ e non riceva alcun nuovo pacchetto di `Build`, per avvisare $\mathcal{R}$ della propria disconnessione. Di seguito è riportato lo pseudocodice della procedura per una SM p_i e per $\mathcal{R}$.

<u>host p_i:</u>

Complain {
    when scade $T_{refresh}$ **do**
        repeat {
            `send` *bubble*($\mathcal{G}$, i, $T_{refresh}$); aspetta per un tempo $T_{refresh}$ una risposta da R;
            case (ricevuto ack): $T_{refresh} \leftarrow$ stima di $T_{\mathcal{R}}$;
            case (ricevuta `Build`): set $T_{refresh}$; *Build-Tree*;
            case (ricevuto `multicast` ($\mathcal{G}$, msg, $T_{refresh}$)): `Deliver` msg; set $T_{refresh}$;
                /* pacchetto `Build` perso; situazione instabile */
            case (timeout): $T_{refresh} \leftarrow T_{refresh} - \frac{T_{refresh} - T_{min}}{2}$;
        } **until** (ricevuta `Build`);
}

<u>$\mathcal{R}$:</u>

Complain {
    when ricevuta *bubble*($\mathcal{G}$, i, T_i) **do**
        if ($p_i \notin S_{\mathcal{R}}$) **then** aggiungi p_i a $S_{\mathcal{R}}$; /* pacchetto `Lost` andato perso */
        if ($T_{\mathcal{R}} > T_i$) **then** `send` ack a p_i specificando $T_{\mathcal{R}}$;
        else non fare niente; /* un multicast m arrivera' tra poco e un pacchetto di `Build`
                    sara' mandato verso p_i */
}

Bibliografia

[1] Acharya A., Badrinath B. R. "A Framework for Delivering Multicast Messages in Networks with Mobile Hosts". *ACM/Baltzer Journal of Wireless Networks, Special Issue on* Routing in Mobile Communication Networks, Vol. 1, pages 199–219, 1996.

[2] Acharya A., Badrinath B. R. "Delivering Multicast Messages in Networks with Mobile Hosts". *Proc. 13th Intl. Conf. on Distributed Computing Systems*, page 292, May 1993.

[3] Aiello R., Pagani E., Rossi G. P. "An Efficient Algorithm for Group Communication". *Proceedings of the 5th IEEE Symposium on Parallel and Distributed Processing*, pages 226–232, Dallas, TX, Dec. 1-4 1993.

[4] Aiello R., Pagani E., Rossi G. P. "Design of a Reliable Multicast Protocol". *Proc. IEEE Infocom '93*, pages 75–81, San Francisco, CA, Mar. 28- Apr. 1 1993.

[5] Aiello R., Pagani E., Rossi G. P. "Causal Ordering in Reliable Group Communications". *Proceedings of the ACM SIGCOMM'93 Conference. Computer Communication Review, Vol. 23, No. 4*, pages 106–115, San Francisco, CA, Sep. 13-17 1993.

[6] Amir Y., Moser L. E., Melliar-Smith P. M., Agarwal D. A., Ciarfella P. "The Totem Single-Ring Ordering and Membership Protocol". *ACM Trans. on Computer Systems, Vol. 13, No. 4*, pages 311–342, Nov. 1995.

[7] Anceaume E., Charron-Bost B., Minet P., Toueg S. "On the Formal Specification of Group Membership Services". *Cornell University Technical Report*, 1995.

[8] Badrinath B. R., Acharya A., Imielinski T. "Designing Distributed Algorithms for Mobile Computing Networks". *To appear in Computer and Communications*, 1996. ftp://athos.rutgers.edu/pub/acharya/algos_journal.ps.Z.

[9] Badrinath B. R., Acharya A., Imielinski T. "Impact of Mobility on Distributed Computations". *ACM Operating Systems Review, Vol. 27, No. 2*, pages 15–20, Apr. 1993.

[10] Badrinath B. R., Imielinski T. "Replication and Mobility". *Proc. 2nd IEEE Workshop on Management of Replicated Data*, pages 9–12, Nov. 1992.

[11] Bagrodia R., Liao W. "Maisie: A language for the design of efficient discrete-event simulations". *IEEE Trans. on Software Engineering, Vol. 20. No. 4*, pages 225–238, 1994.

[12] Ballardie T., Francis P., Crowcroft J. "Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing". *Proc. SIGCOMM'93*, pages 85–94, Sep. 1993.

[13] Barbará D., Garcia-Molina H. "Replicated Data Management in Mobile Environments: Anything New Under the Sun ?". *Proc. IFIP Conf. on Applications in Parallel and Distributed Computing*, Caracas, Venezuela, Apr. 1994.

[14] Bassak G. "Brainy, Brawny Batteries". *Byte. Special Issue on "Mobile Computing"*, page 131, Jun. 1995.

[15] Bertino E., Pagani E., Rossi G. P. "Fault-Tolerance and Recovery in Mobile Computing Systems". "Recovery in Database Management Systems", ch.24, Eds. V. Kumar and M. Hsu:661–696, Prentice-Hall, 1998.

[16] Bertino E., Pagani E., Rossi G.P. "Transaction Management in Mobile Computing Systems". *Proc. II Workshop "Sistemi Distribuiti: Algoritmi, Architetture e Linguaggi"*, Bertinoro, Sep. 23-25 1997.

[17] Birman K., Schiper A., Stephenson P. "Lightweight Causal and Atomic Group Multicast". *ACM Transactions on Computer Systems, Vol. 9, No. 3*, pages 272–314, Aug. 1991.

[18] Calvert K. L., Zegura E. W., Donahoo M. J. "Core Selection Methods for Multicast Routing". *Proc. ICCCN'95*, 1995.

[19] Chandra T. D., Hadzilacos V., Toueg S. "The Weakest Failure Detector for Solving Consensus". *Cornell Technical Report No. 92-1293*, 1992.

[20] Chandra T. D., Hadzilacos V., Toueg S., Charron-Bost B. "On the Impossibility of Group Membership". *Cornell University Technical Report. To appear in Proc. PODC'96*, 1995.

[21] Chandra T. D., Toueg S. "Unreliable Failure Detectors for Reliable Distributed Systems". *Cornell Technical Report No. 93-1374*, 1993.

[22] Chiang C.-C., Gerla M. "Routing and Multicast in Multihop, Mobile Wireless Networks". *Proc. IEEE ICUPC'97*, Oct. 1997.

[23] Chiang C.-C., Gerla M., Zhang L. "Shared Tree Wireless Network Multicast". *Proc. IEEE IC3N'97*, 1997.

[24] Chiang C.-C., Gerla M., Zhang L. "Forwarding Group Multicast Protocol (FGMP) for Multihop, Mobile Wireless Networks". *ACM/Baltzer Journal of Cluster Computing: Special Issue on Mobile Computing, Vol. 1, No. 2*, 1998.

[25] Chlamtac I., Weinstein O. "Distributed 'Wave' Broadcasting in Mobile Multi-Hop Radio Networks". *Proc. 7th Intl. Conf. on Distributed Computing Systems*, pages 82–89, 1987.

[26] Chlamtac I., Weinstein O. "The Wave Expansion Approach to Broadcasting in Multihop Radio Networks". *IEEE Transactions on Communications, Vol. 39, No. 3*, pages 426–433, Mar. 1991.

[27] Cho K., Birman K. P. "A Group Communication Approach for Mobile Computing". *Proc. Mobile Computing Workshop*, Santa Cruz, CA, Dec. 1994.

[28] Corson M. S., Ephremides A. "A Distributed Routing Algorithm for Mobile Wireless Networks". *Journal of Wireless Networks, Vol. 1, No. 1*, pages 61–81, Feb. 1995.

[29] Deering S. E. "Multicast Routing in a Datagram Internetwork". *Ph.D. dissertation, Stanford University*, Dec.1991.

[30] Deering S. E., Cheriton D. R. "Multicast Routing in Datagram Internetworks and Extended LANs". *ACM Trans. on Computer Systems, Vol. 8, No. 2*, pages 85–110, May 1990.

[31] Deering S., Estrin D., Farinacci D., Jacobson V., Liu C.-G., Wei L. "The *PIM* Architecture for Wide-Area Multicast Routing". *IEEE/ACM Trans. on Networking, Vol. 4, No. 2*, page 153, Apr. 1996.

[32] Diot C., Dabbous W., Crowcroft J. "Multipoint Communication: A Survey of Protocols, Functions and Mechanisms". *IEEE Journal on Selected Areas in Communications, Vol. 15, No. 3*, pages 277–290, Apr. 1997.

[33] Dolev D., Malki D., Strong R. "A Framework for Partitionable Membership Service". *Technical Report CS95-4*, Inst. of Computer Science, Hebrew University, Jerusalem, 1995.

[34] Donahoo M. J., Zegura E. W. "Core Migration for Dynamic Multicast Routing". *Proc. ICCCN'96*, 1996.

[35] Edney J. "802.11: The New Wireless Standard". *Data Communications International, Vol. 24, No. 12*, page 77, Sep. 1995.

[36] Fischer M., Lynch N., Paterson M. S. "Impossibility of Distributed Consensus with One Faulty Process". *Journal of the ACM, Vol. 32, No. 2*, pages 374–382, Apr. 1985.

[37] Floyd S., Jacobson V., McCanne S., Liu C.-G., Zhang L. "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing". *Proc. SIGCOMM'95, ACM Computer Communication Review, Vol. 25, No. 4*, pages 342–356, 1995.

[38] Gaber I., Mansour Y. "Broadcast in Radio Networks". *Proc. SODA'95*, 1995.

[39] Garay J. A. "Reaching (and Maintaining) Agreement in the Presence of Mobile Faults". *Proceedings WDAG '94*, pages 253–264, Oct. 1994.

[40] Gerla M., Tzu-Chieh Tsai J. "Multicluster, Mobile, Multimedia Radio Network". *ACM/Baltzer Journal of Wireless Networks, Vol. 1, No. 3*, pages 255–265, 1995.

[41] Gilhousen K. S., Jacobs I. M. et al. "On the Capacity of a Cellular CDMA System". *IEEE Trans. Veh. Tech., Vol. 40*, pages 303–312, 1991.

[42] Haas Z. J. "Panel Report on Ad Hoc Networks - MILCOM'97". *ACM Mobile Computing and Communications Review, Vol. 2, No. 1*, pages 15–18, Jan. 1998.

[43] Hadzilacos V., Toueg S. "A Modular Approach to Fault-Tolerant Broadcasts and Related Problems". *WDAG '93 Tutorial*, 1993.

[44] Hadzilacos V., Toueg S. "Fault-Tolerant Broadcasts and Related Problems". Distributed Systems:97–145, S. Mullender, Addison-Wesley, 1993.

[45] Imielinski T., Badrinath B. R. "Mobile Wireless Computing: Challenges in Data Management". *Communications of the ACM, Vol. 37, No. 10*, page 18, 1994.

[46] Imielinski T., Badrinath B. R. "Mobile Wireless Computing: Solutions and Challenges in Data Management". *Rutgers University Tech. Rep. DCS-TR-296/WINLAB-TR-49*, 1994.

[47] Imielinski T., Badrinath B. R. "Querying Locations in Wireless Environments". Wireless Communications: Future Directions:85–108, J. M. Holtzmann, D. J. Goodman. Kluwer Academic Publishers, 1993.

[48] Imielinski T., Badrinath B. R. "Data Management for Mobile Computing". *SIGMOD RECORD, Vol. 22, No. 1*, pages 34–39, Mar. 1993.

[49] Jing J., Bukhres O., Elmagarmid A. "Distributed Lock Management for Mobile Transactions". *Proc. 15th Intl. Conf. on Distributed Computing Systems*, pages 118–125, Vancouver, British Columbia, Canada, Jun. 1995.

[50] Jung P., Baier P. W., Steil A. "Advantages of CDMA and Spread Spectrum Techniques over FDMA and TDMA in Cellular Mobile Radio Applications". *IEEE Trans. Veh. Tech., Vol. 42*, pages 357–364, 1993.

[51] Kleinrock L. "Nomadic Computing - An Opportunity". *ACM Computing Communication Review, Vol. 25, No. 1*, pages 36–40, Jan. 1995.

[52] Lam D., Cui Y., Cox D. C., Widom J. "A Location Management Technique to Support Lifelong Numbering in Personal Communications Services". *ACM Mobile Computing and Communications Review, Vol. 2, No. 1*, pages 27–35, Jan. 1998.

[53] Macker J. P., Corson M. S. "Mobile Ad Hoc Networking and the IETF". *ACM Mobile Computing and Communications Review, Vol. 2, No. 1*, pages 9–14, Jan. 1998.

[54] Moser L. E., Melliar-Smith P. M., Agrawala V. "Processor Membership in Asynchronous Distributed Systems". *IEEE Trans. on Parallel and Distributed Systems, Vol. 5, No. 5*, pages 459–473, May 1994.

[55] Moy J. "Multicast Extensions to OSPF". *RFC 1584, work in progress*, Mar. 1994.

[56] Mummert L. B., Ebling M. R., Satyanarayanan M. "Exploiting Weak Connectivity for Mobile File Access". *Operating Systems Review, Vol. 29, No. 5*, pages 143–155, Dec. 1995.

[57] Pagani E., Rossi G. P. "On-demand Multicast Routing in Multi-Hop Wireless Networks". *Technical Report, Dip. Scienze dell'Informazione, Università degli Studi di Milano*, 1998. Submitted to ICDCS'99.

[58] Pagani E., Rossi G. P. "Providing Reliable and Fault Tolerant Broadcast Delivery in Mobile ad-hoc Networks". *To appear in the Special Issue "Mobile ad-hoc Networking" of the ACM/Baltzer Journal of Mobile Networks and Applications (MONET)*, 1999.

[59] Pagani E., Rossi G. P. "A Set of Multicast Primitives for Fault Tolerant Distributed Systems". *Journal of High Speed Networks, Vol. 4, No. 3*, pages 299–316, IOS Press, Sep. 1995.

[60] Pagani E., Rossi G.P. "Reliable Broadcast in Mobile Multihop Packet Networks". *Proc. 3rd ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'97)*, pages 34–42, Budapest, Sep. 26-30 1997.

[61] Perkins C. E., Bhagwat P. "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers". *ACM SIGCOMM'94*, pages 234–244, 1994.

[62] Ramanathan R., Steenstrup M. "Hierarchically-Organized, Multi-hop Mobile Wireless Networks for Quality-of-Service Support". *Journal of Mobile Networks and Applications, Special Issue 'Mobile Multimedia Communications', Vol. 3, No. 1*, pages 101–119, Jan. 1998.

[63] Reischuk R. "A New Solution for the Byzantine Generals Problem". *Information and Control, Vol. 64*, pages 23–42, 1985.

[64] Salamone S. "Radio Days". *Byte. Special Issue on "Mobile Computing"*, page 107, Jun. 1995.

[65] Santoro N., Widmayer P. "Time is not a Healer". *Proceeding of the 6th Annual Symposium on the Theoretical Aspects of Computer Science (STACS'89)*, Vol. 349 of Lecture Notes in Computer Science:304–313, Feb. 1989.

[66] Shields C., Garcia-Luna-Aceves J.J. "The Ordered Core Based Tree Protocol". *Proc. IEEE Infocom'97*, Apr. 1997.

[67] Short J., Bagrodia R., Kleinrock L. "Mobile Wireless Network System Simulation". *Proc. ACM Mobile Communication Networking Conference*, Nov. 1995.

[68] Tanenbaum A. S. *Computer Networks*. Prentice-Hall, 1988.

[69] Welch G. F. "A Survey of Power Management Techniques in Mobile Computing Operating Systems". *Operating Systems Review, Vol. 29, No. 4*, pages 47–55, Oct. 1995.