UN NUOVO ALGORITMO DI ROUTING PER RETI WIRELESS DI TIPO ROOFTOP

Mario Giammarco 1617871

8 luglio 2003

Indice

Capitolo 1

INTRODUZIONE

I progressi della tecnologia aprono spesso nuovi scenari. Attualmente si sta espandendo e sta suscitando molto interesse la tecnologia "wireless". In particolare le nuove schede per il collegamento in rete ad onde radio a basso costo hanno permesso la creazione di reti alternative ad Internet anche di medie dimensioni.

Queste reti vengono chiamate "rooftop" in base alla loro principale caratteristica: le antenne dei dispositivi di rete sono in genere posizionate sui tetti delle abitazioni dei partecipanti di queste comunità wireless.

Poiché la portata delle antenne è limitata può capitare che per fare comunicare due nodi distanti occorra attraversare diversi nodi che servano "da ponte".

È quindi fondamentale per le sopracitate comunità avere a disposizione un sistema di routing e di assegnazione univoca degli indirizzi dei nodi.

Sfortunatamente gli algoritmi già esistenti per Internet non si adattano efficacemente a questa particolare situazione.

È stato quindi progettato, implementato e valutato in via simulativa, un algoritmo di routing specifico che, utilizzando backtracking e la tecnica degli "shortcut", riesce a trovare dei percorsi qualitativamente validi (che cioè attraversano meno nodi possibili per giungere a destinazione) senza sovraccaricare la rete.

A completamento dell'algoritmo viene inoltre presentato un sistema di assegnazione univoca degli indirizzi.

L'algoritmo, per ragioni di portabilità, è stato implementato nel linguaggio Java e tramite il simulatore Javasim è stato confrontato con altri ottenendo risultati positivi.

I restanti capitoli della tesi sono così strutturati:

- nel secondo capitolo viene introdotto il problema del routing e vengono mostrati i principali algoritmi in uso su Internet;
- nel terzo capitolo si parla del modello di rete wireless più diffuso attualmente con i relativi algoritmi;
- nel quarto capitolo si introduce il nuovo protocollo di routing IP versione sei;
- nel quinto capitolo viene spiegato dettagliatamente il modello di reti wireless rooftop evidenziandone i problemi anche attraverso un esempio errato di approccio algoritmico;
- nel sesto capitolo si presenta l'algoritmo oggetto del seguente lavoro di tesi;
- nel settimo capitolo si mostrano i dettagli dell'implementazione di riferimento dell'algoritmo;
- nell'ottavo capitolo viene valutato l'algoritmo in via simulativa e ne vengono presentati i risultati;
- per finire nel nono capitolo si traggono alcune conclusioni e si presentano alcune idee per eventuali sviluppi futuri.

Capitolo 2

IL ROUTING SU INTERNET

Il routing sostanzialmente è un problema di teoria dei grafi. Più specificamente dati un insieme di nodi collegati tra loro da archi pesati il routing consiste nel trovare il percorso di costo minimo fra ogni coppia di nodi. La difficoltà nel realizzare un algoritmo di routing deriva dai requisiti di scalabilità richiesti e quindi dalle relative conseguenze: l' algoritmo deve essere di tipo distribuito e ogni router non può ovviamente avere una visione completa della rete. Non esiste al momento un algoritmo di routing "ideale" che possa quindi operare bene su ogni modello di rete esistente; al contrario per ogni situazione sono stati realizzati specifici algoritmi, come vedremo nei capitoli successivi.

2.1 Il modello attuale di Internet

Nella figura 2.1 si può vedere il modello attuale di internet che è discretamente più complesso del modello iniziale che aveva una topologia molto più semplice, praticamente ad albero. In esso possiamo notare che vi sono diversi "backbone" (le dorsali ad alta capacità di trasmissione) a cui si collegano diversi AS (autonomous systems). Un AS rappresenta qualsiasi organizzazione collegata ad internet: imprese, enti statali, università, ISP (rivenditori di collegamenti ad internet) e altri. Vi sono AS collegati a più dorsali (es: grandi aziende che desiderano collegamenti ridondanti) e so-

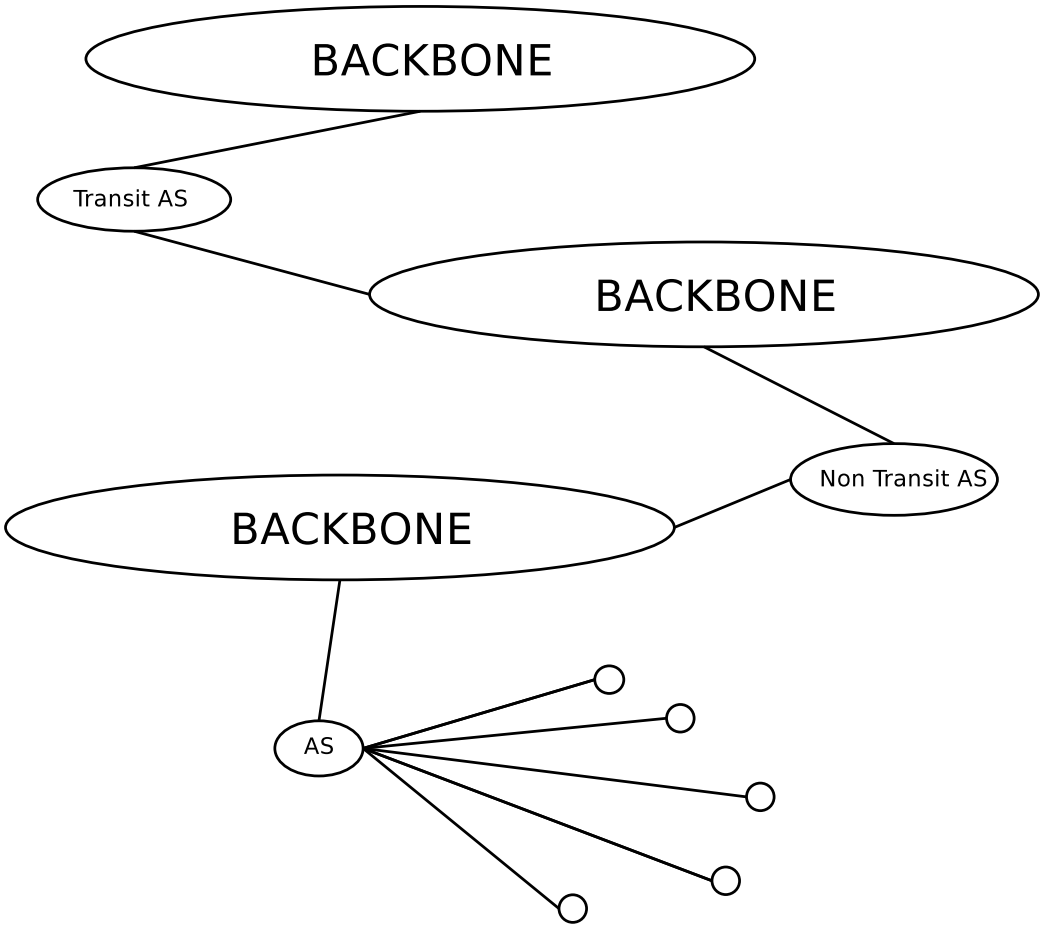

Figura 2.1: Internet attuale

no detti "multihomed", alcuni di questi consentono di fare da tramite al traffico fra le dorsali e vengono definiti "transit" AS. Riguardo al suddetto modello si possono fare le seguenti considerazioni:

- Data l'enorme quantità di computer collegati attualmente in internet (diversi milioni) il problema è troppo complesso per essere affrontato con un singolo algoritmo; come vedremo la complessità è stata ridotta utilizzando un sistema gerarchico;

- È presente in internet un'autorità centrale che assegna gli indirizzi ai richiedenti; in questo modo si garantisce l'univocità di ogni singolo

indirizzo e una distribuzione degli indirizzi che favorisca l'algoritmo di routing utilizzato;

- il routing è reso più difficile da considerazioni "politiche": per esempio alcuni AS pur essendo collegati a più backbone non consentono di far transitare traffico attraverso di loro (vengono definiti "non-transit AS";

- gli algoritmi attuali assumono che la maggior parte degli AS si colleghino ad internet tramite un solo percorso che viene chiamato "default route";

- le dorsali hanno una larghezza di banda molto più elevata dei singoli network quindi possono reggere una quantità di traffico maggiore.

2.2 La netmask

Ad ogni computer collegato ad internet viene fornito un indirizzo IPv4 univoco[1] di 32 bit. Questo indirizzo non si considera come un tutt'uno ma viene suddiviso in due parti un prefisso "network" e una parte "host". Per specificare come dividere le due parti viene utilizzata la "netmask", un indirizzo di 32 bit di cui i bit messi ad uno indicano la parte network, mentre quelli a zero la parte host dell'indirizzo ip. Inizialmente i bit a uno nella netmask potevano essere non contigui, ma in seguito ad una importante riforma di qui parliamo fra poco si è stabilito la contiguità dei bit; in tal caso la netmask si può rappresentare in forma più compatta con un singolo numero che indica quanti bit sono settati.

Quindi il problema iniziale di gestire con un algoritmo di routing milioni di computer collegati fra di loro è ora scomposto in due più semplici: all'interno di ogni AS si usano algoritmi di tipo "intra-domain" che considerano solo gli indirizzi interni (cioè quelli il cui prefisso corrisponde a

[1] in realtà l'indirizzo IP viene assegnato alla scheda di rete e vi possono essere eccezioni all'univocità

quelli posseduti dall'AS), mentre le richieste di destinazioni esterne vengono dirottate verso un computer "gateway" dotato di un collegamento esterno all'autonomous system. Il gateway (che userà un algoritmo di tipo "inter-domain") non deve conoscere il route per ogni possibile destinazione (che sono milioni), ma solamente per ogni prefisso (attualmente qualche decina di migliaia), sarà compito poi del gateway dell'AS remoto, una volta ricevuti i pacchetti, di instradarli alla destinazione finale. Praticamente il gateway vede ogni AS come una singola entità mentre ovviamente al suo interno l'AS contiene un numero elevato di computer e router.

Il compito di assegnare gli indirizzi in maniera da ridurre gli sprechi e cercare di contenere il numero di prefissi è assegnato ad un'autorità centrale. Gli indirizzi ip inizialmente venivano assegnati in lotti da 256 host (classe C), 65536 host (classe B) o 16 milioni di host (classe A). Questi lotti si dicono "provider-indipendent" in quanto vengono assegnati in maniera univoca ad un AS che li mantiene anche nel caso cambi il collegamento fisico ad internet (per es.: cambio di provider). Dati i forti sprechi che questa soluzione comportava (a chi richiedeva per esempio cinquecento indirizzi veniva dato un lotto di classe B sprecando quindi più di sessantamila indirizzi), attualmente si usa un nuovo metodo chiamato CIDR (classless interdomain routing).

Il compito del CIDR è quello di utilizzare meglio lo spazio di indirizzi e di cercare di contenere l'aumento del numero di prefissi interdomain.

Per far questo opera nel seguente modo:

- i lotti di indirizzi non vengono più assegnati ai singoli AS ma ai service provider che forniscono loro la connessione internet (vengono quindi definiti "provider aggregatable"); in questo modo un AS che cambia il collegamento fisico ad internet cambia anche i suoi indirizzi;

- i lotti non sono più di soli tre grandezze ma possono assumere le dimensioni di tutte le potenze del due (quindi la netmask può variare liberamente da zero a trentuno).

In questo modo si segue la topologia attuale di internet: ai backbone infatti si collegano direttamente solo grandi aziende o provider che poi rivendono la connettività ad aziende di dimensioni minori. A questi AS collegati direttamente ai backbone vengono assegnati lotti contenenti milioni di indirizzi con quindi un prefisso molto corto. Di questi prefissi ovviamente ce ne saranno pochi (per es.: di prefissi con netmask /8 ce ne possono ovviamente essere solo 256). Gli AS poi possono spezzare questi grandi lotti in sotto lotti delle dimensioni richieste dai loro clienti (con granularità appunto migliore delle sole tre classi A B C), con prefissi più lunghi, ma questi ultimi non devono essere pubblicati all'esterno dell'AS.

Per chiarire meglio il tutto con un esempio ipotizziamo un AS a cui venga assegnato il prefisso 100 con netmask 8 (in maniera compatta 100/8). I clienti a lui collegati potranno avere prefissi personali del tipo 100.200/16 o similari, ma il 100 deve rimanere fisso quindi nessuno potrà avere 101/8 (in realtà vi è un eccezione come vedremo fra poco). In tal modo agli altri AS collegati direttamente allo stesso backbone per indirizzare questo AS e tutti i suoi discendenti basta usare un unico prefisso 100/8.

Il CIDR è riuscito sia ad utilizzare meglio lo spazio di indirizzi IPV4, ma anche per un certo periodo a contenere l'aumento del numero di prefissi. Sfortunatamente negli ultimi anni il numero di prefissi necessari ha reiniziato ad aumentare. Si è infatti diffuso il fenomeno del multihoming, cioè vari AS si collegano ad internet attraverso più di un provider (per esigenze di ridondanza).

Ora il discorso fatto poco fa sfortunatamente perde in validità come il seguente esempio dimostra. Se l'AS "X" si collega a internet tramite due provider, AS "Y" dotato di prefisso 100/8 e AS "Z" dotato di prefisso 101/8 non può ovviamente avere degli indirizzi ip con due prefissi diversi. Mettiamo quindi che scelga di avere il prefisso 100.10/16, ottenuto da Y. A questo punto Z dovrà "pubblicare" sul backbone non solo il prefisso 101/8 ma anche 100.10/16 per indicare che X può essere raggiunto anche tramite lui.

In questo modo sfortunatamente si causa un aumento considerevole del numero di prefissi pubblicati in un backbone, la cosa che appunto si è

cercato di evitare col CIDR.

2.3 Gli algoritmi intradomain

Questi algoritmi sono tipicamente utilizzati all'interno di un AS e il loro compito è non solo quello di trovare un percorso ma, possibilmente, di trovare quello di costo inferiore, dove per costo si intende una combinazione dei seguenti fattori:

- il numero di router attraversati per giungere a destinazione;
- la larghezza di banda e la latenza dei router attraversati;
- il carico di lavoro presente nei router attraversati.

Trovare quindi una giusta metrica per stabilire il costo di un percorso non è un compito banale e ci sono volute numerose prove e simulazioni per raggiungere i risultati contenuti in "revised ARPANET routing metric" [KZ89].

2.3.1 Routing Information Protocol (RIP)

L'algoritmo su cui si basa RIP [Mal98] è definito di tipo "distance vector" in quanto ogni router mantiene una tabella con la distanza in "hop" (numero di router attraversati) dagli altri router. All'inizio questa tabella conterrà solo i diretti vicini (con distanza uno); in seguito ogni router invia la tabella ai propri vicini che la utilizzano per completare e aggiornare la loro.

In condizioni statiche (nessun router viene aggiunto o tolto) l'algoritmo converge rapidamente, cioè in breve tempo tutti i router acquisiscono una topologia corretta della rete. Nel caso in cui la topologia della rete cambi può succedere che due (o più) router abbiano una visione diversa della rete; in tal caso inizieranno a scambiarsi i loro distance vector, ma sfortunatamente può capitare che si instauri un ciclo senza fine e l'algoritmo quindi non si stabilizzi (problema del "count to infinity"). Esistono

varie tecniche per contenere questa divergenza dell'algoritmo per esempio limitare la massima distanza a 16 hop (ma così facendo si limita anche la grandezza massima della rete a 16 hop); ma risulta evidente che l'algoritmo non riesce a stabilizzarsi in tempi brevi e per questo motivo attualmente sulla maggior parte dei router sono utilizzati algoritmi del tipo "link-state"

2.3.2 Open Shortest Path First (OPSF)

L'algoritmo su cui si basa OSPF [Moy89] è detto di tipo link-state. Ogni router comunica agli altri solo informazioni di cui può appurarne la consistenza di persona: in pratica comunica solamente lo stato dei collegamenti (link) con i suoi diretti vicini. L'informazione viene propagata accuratamente attraverso tutta la rete e sono presi appositi accorgimenti per fare in modo che ogni router disponga al più presto possibile dell'informazione più recente.

Una volta ottenuti tutti i link-state il router può creare una mappa accurata della rete nella sua memoria e su questa utilizzare i classici algoritmi sui grafi, in particolare quello di Dijkstra del percorso più breve.

Grazie al fatto che vengono trasmesse solo informazioni "certe" questo algoritmo converge sempre rapidamente, a differenza di quelli basati su distance vector.

2.4 Gli algoritmi interdomain

2.4.1 Border Gateway Protocol (BGP)

Il compito del BGP è veramente difficile in quanto questo protocollo serve ai router intradomain per trovare il percorso per tutte le destinazioni possibili.

Fortunatamente grazie al CIDR questo compito si riduce a trovare il percorso per tutti i prefissi possibili, che sono comunque ben oltre cinquantamila allo stato attuale.

Per il BGP è inoltre impossibile calcolare il percorso ottimale, in quanto le informazioni del costo di un percorso ottenute dagli AS non sono standardizzate fra di loro (per es.: un costo mille per un AS potrebbe significare un ottimo percorso, mentre per un altro AS un percorso inaccettabile).

Il compito del BGP è complicato anche da considerazioni "politiche" in quanto certi AS possono permettere il traffico attraverso di loro solo di provider "amici".

Per tutti questi motivi il BGP non usa né il distance vector né il link state, ma invia a tutti i router partecipanti l'elenco completo di ogni percorso per arrivare ad ogni destinazione, riuscendo in tal modo ad evitare la formazione di qualsiasi tipo di loop.

Capitolo 3

IL ROUTING AD HOC

Nel corso degli anni sono emerse nuove tipologie di rete [CM99] sufficientemente diverse da Internet da rendere necessario sviluppare dei nuovi algoritmi di routing "ad hoc" per questi casi. In particolare ad esempio in caso di guerre o calamità naturali sarebbe di grande utilità poter dotare ogni soldato o vigile del fuoco di un dispositivo di comunicazione in grado di funzionare senza bisogno di infrastrutture preesistenti e soprattutto in grado di raggiungere altri dispositivi situati più lontani della sua portata radio.

3.1 Il modello di rete wireless

Si delinea quindi il modello presente nella figura 3.1 con le seguenti caratteristiche:

- i dispositivi sono forniti con una radio di capacità limitata che gli permette di raggiungere direttamente solo i loro vicini entro un certo raggio;
- i dispositivi sono mobili quindi le loro posizioni mutano nel tempo;
- la mobilità implica che l'alimentazione sia probabilmente a batterie, quindi deve essere tenuto in considerazione il loro consumo;

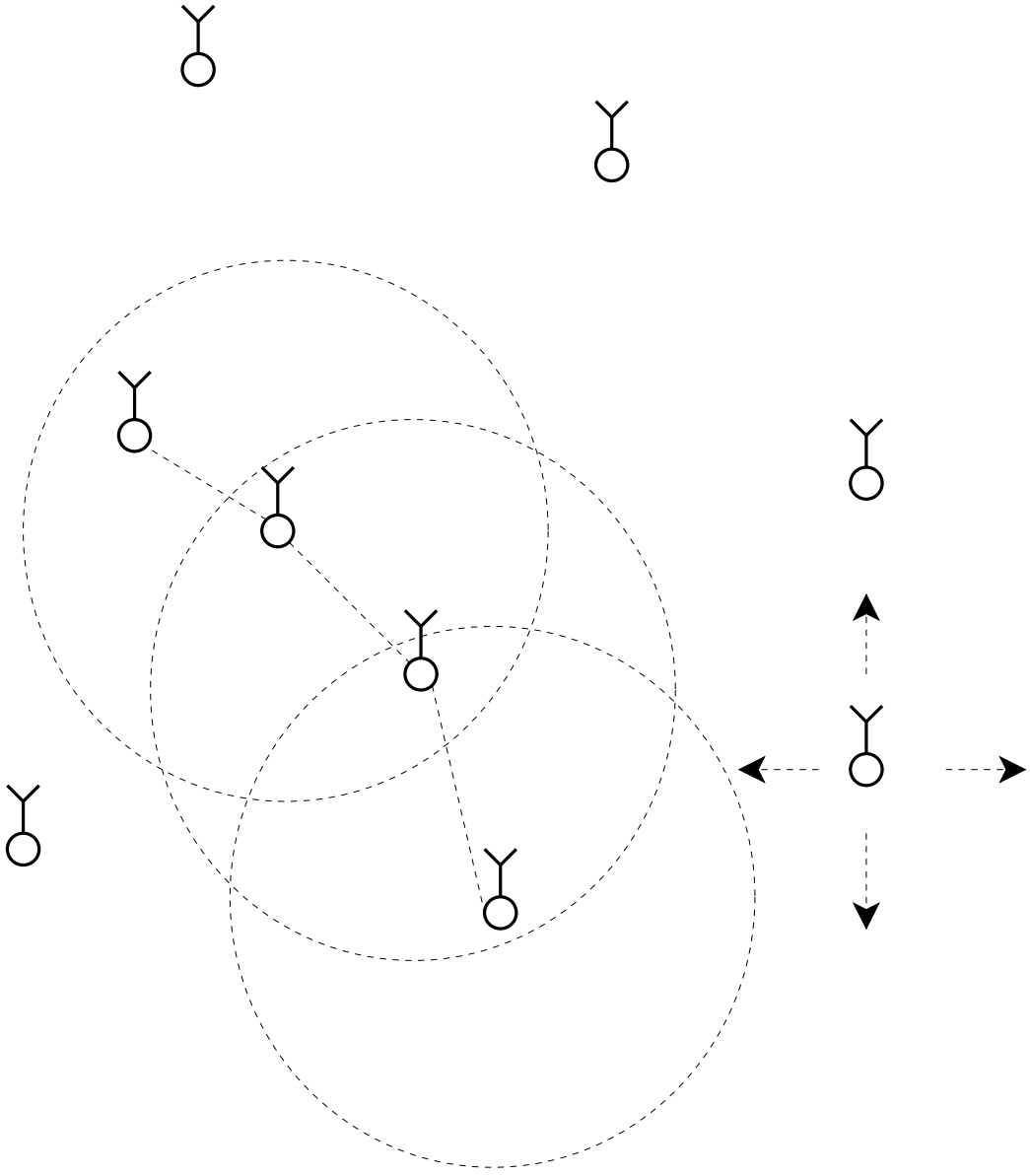

Figura 3.1: Mobile Area NETworks

- ogni dispositivo svolge sia le funzioni di host che quelle di router;
- tutti i dispositivi hanno la stessa larghezza di banda (non esiste quindi un backbone ad alta capacità);
- ogni router ha collegamenti multipli con gli altri quindi il fenomeno del "multihoming" è molto diffuso;
- la capacità di memoria e di calcolo di ogni singolo partecipante al network può essere di dimensioni ridotte.

La ricerca nel campo di queste Mobile Area Networks (MANET) è molto fertile, svariati gruppi di ricerca hanno studiato approcci per risolvere i vincoli qui sopra descritti.

Nelle successive sezioni sarà mostrata una selezione dei più significativi algoritmi. Saranno tralasciati gli algoritmi obsoleti, quelli di cui non esistono implementazioni né simulazioni e, in caso di algoritmi simili, sarà mostrato quello più "famoso".

Per ulteriori approfondimenti si può consultare anche la lista (non completa) presente in [wik].

3.2 Gli algoritmi proattivi

Questa classe di algoritmi si inspira alle idee di quelli utilizzati in Internet, in particolare a quelli di tipo link-state.

3.2.1 Fisheye Routing Protocol (FSR)

Descrizione

L'algoritmo FSR [Sun] è un algoritmo di tipo link-state che si basa sull'interessante osservazione che per ogni nodo è meglio avere una vista "ad occhio di pesce" dell'intera rete, cioè nel caso in cui la rete sia così vasta da non poter avere un'informazione completa, è meglio avere un'informazione più dettagliata e più aggiornata dei nodi vicini (in termini di numero di archi attraversati).

In questo modo man mano il pacchetto di dati si sposta dalla sorgente per andare verso la destinazione incontrerà nodi che hanno una visione via via sempre peggiore del nodo sorgente (che non interessa più) mentre d'altro canto avranno una visione sempre migliore della destinazione (che è l'unica cosa che conta).

L'algoritmo procede in questo modo: i pacchetti di aggiornamento dello stato dei link (spostamento o spegnimento) non vengono inviati ad ogni singolo evento che accade (come fa OSPF), ma vengono inviati a determinati intervalli (quindi rappresentano una specie di riassunto), e gli intervalli a cui vengono inviati sono più frequenti per i nodi vicini e via via sempre meno frequenti per i nodi lontani.

Considerazioni

Il FSR è uno dei migliori algoritmi di routing proattivi, anzi si può dire l'unico algoritmo di questa classe a presentare sufficienti caratteristiche di scalabilità riguardo al numero di nodi e sopratutto al traffico di rete; in più, non presenta la latenza iniziale dovuta alla ricerca del percorso tipica degli algoritmi reattivi.

3.3 Gli algoritmi reattivi

A differenza degli algoritmi proattivi quelli reattivi creano i percorsi "on demand", cioè solo quando vengono effettivamente richiesti. Infatti, poiché nelle reti MANET i vari percorsi da un host all'altro cambiano in continuazione a causa della mobilità dei dispositivi, risulta inutile e controproducente mantenere in anticipo tutti i route possibili per ogni destinazione in quanto:

- al momento dell'effettivo utilizzo di un percorso precalcolato questo potrebbe essere già obsoleto;
- il sovraccarico di traffico dati sulla rete per mantenere route non utilizzati può diventare insostenibile.

3.3.1 Ad Hoc On demand Distance Vector (AODV)

Descrizione

L'AODV [Per97] è uno degli algoritmi nato in seno all' Internet Engineering Task Force (IETF) e per questo motivo uno dei più vicini alla standardizzazione; inoltre di questo algoritmo sono già state realizzate varie implementazioni.

Il funzionamento dell'algoritmo è il seguente: quando viene richiesto un route il nodo sorgente propaga in tutta la rete ("flood") una richiesta per il nodo destinazione. Quando questa richiesta arriva al nodo destinazione o a un nodo che abbia un'informazione sufficientemente aggiornata questa viene rimandata al nodo sorgente tramite il percorso inverso di quello effettuato sino a quel momento.

Per essere certi di costruire percorsi senza loop e che contengano informazioni aggiornate ogni nodo mantiene un numero di sequenza che viene incrementato. Ogni richiesta viene identificata univocamente dall'ip del nodo sorgente e dal suo numero di sequenza; ogni nodo risponde con le sue informazioni riguardo alla destinazione solo ed esclusivamente se il numero di sequenza che lui ricorda per la destinazione è maggiore o uguale del numero di sequenza della destinazione contenuto nella richiesta.

Considerazioni

L'algoritmo, a causa del flood ad ogni ricerca di route, funziona adeguatamente per reti di piccole dimensioni (cinquanta o cento nodi al massimo), dove offre un basso overhead dovuto al fatto che i percorsi vengono cercati solamente on-demand.

Non è garantito che i percorsi siano i più corti, e, a causa del funzionamento on demand, vari pacchetti dati vengono persi finché l' algoritmo non trova un percorso per la destinazione.

Vi è la possibilità che si formi qualche loop ma di breve durata.

3.3.2 Dynamic Source Routing (DSR)

Descrizione

Il DSR [JM96] sfrutta una possibilità presente anche nello standard ipv4 ma raramente utilizzata: quella di poter inserire in ogni pacchetto ip l'elenco completo dei nodi da attraversare per raggiungere la destinazione.

Questa scelta piuttosto radicale serve ad evitare che si possano formare cicli nel percorso e permette ad ogni nodo di essere informato sui percorsi per varie destinazioni semplicemente "spiando" i pacchetti che passano attraverso di lui.

Infatti l'algoritmo prevede che i vari nodi mantengano in memoria un numero considerevole di percorsi in maniera da ridurre il numero di richieste di nuovi route.

La fase di richiesta route avviene in maniera simile all' AODV con un flood iniziale al quale risponde chi è in possesso di informazioni aggiornate.

Anche il DSR è stato proposto dall'IETF ed è vicino alla standardizzazione.

Considerazioni

Anche questo algoritmo non scala all' aumentare della dimensione della rete in quanto usa il flood iniziale per cercare la destinazione.

Anche se apparentemente può sembrare comportarsi meglio del AODV non bisogna dimenticare che ogni pacchetto dati verrà gravato dall' overhead di dover contenere anche il percorso completo sorgente destinazione, quindi anche per questo motivo l'algoritmo non scala all' aumentare del numero di nodi della rete.

In compenso in questo algoritmo è garantita l' assenza totale di loop.

3.3.3 Temporally Aligned Routing Algorithm (TORA)

Descrizione

Il TORA [PC97] basa il suo funzionamento su una precisa sequenza temporale di eventi per cui i nodi su cui gira questo algoritmo devono disporre di orologi sincronizzati, per esempio col "Global Positioning System" [Alo] con un protocollo di rete come Network Time Protocol.

L' algoritmo cerca di costruire un grafo diretto aciclico (DAG) dalla sorgente alla destinazione che viene considerata come radice, quindi con altezza zero, mentre i gli altri nodi del DAG hanno un'altezza pari al numero di nodi da attraversare per arrivare alla radice.

In questo modo l'algoritmo trova potenzialmente più di un percorso per arrivare a destinazione, quindi la manutenzione dei percorsi in caso di guasto di un nodo o del suo spostamento è semplificata in quanto nella maggior parte dei casi non bisogna ripartire con la richiesta di un nuovo percorso, ma utilizzare uno di quelli rimasti.

Considerazioni

Sebbene una volta trovato un percorso multiplo l' algoritmo necessiti di meno manutenzione, è stato osservato che il tempo necessario all' algoritmo per creare il DAG è notevole e ne compromette le prestazioni [BMJ+98].

Anche in questo caso vi è la possibilità della formazione di loop, ma di breve durata.

3.4 Gli algoritmi geografici

Questa classe di algoritmi si basa sulla possibilità di conoscere la posizione geografica di ogni dispositivo connesso alla rete e quindi la possibilità di capire quanto si è vicini alla destinazione e in che direzione occorre muoversi.

Esistono diversi metodi per ottenere le coordinate: all'esterno si può usare il GPS [Alo], all'interno gli ultrasuoni o radiofrequenze [BP00] , oppure si possono utilizzare anche metodi geometrici [CHH01].

Sebbene gli algoritmi geografici possano sembrare a prima vista potenzialmente "migliori" di tutti quelli qui elencati, bisogna far notare il loro principale difetto: occorre predisporre un servizio (simile al DNS) che associ l' indirizzo ip di un host alle sue coordinate geografiche attuali. Compito non semplice in quanto è ovvio che questo algoritmo dovrà essere distribuito e che ad ogni spostamento di un dispositivo la sua associazione ip/indirizzo dovrà essere aggiornata.

Esiste un algoritmo che realizza un' efficiente database distribuito: il Grid Location Service [LJD+00]. I costi di mantenimento delle associazioni sono dell' ordine di $O(\log(N))$ con N il numero di nodi della rete.

È bene notare quindi che ai costi di ogni singolo algoritmo di questa classe dovranno essere aggiunti i costi dell' ip resolution.

3.4.1 Location Aided Routing (LAR)

Descrizione

Il LAR [KV98] è un algoritmo di routing derivato dal DSR.

È stato migliorato il flood iniziale per la ricerca di un percorso: nel caso in cui si sia a conoscenza della posizione (anche approssimativa) della destinazione, si mandano i pacchetti di richiesta route verso quella direzione, cercando di coprire un' area tanto più ampia quanto più è imprecisa la conoscenza della posizione.

Considerazioni

La possibilità di eseguire un flood mirato permette migliori prestazioni rispetto al DSR e all'AODV, ma la necessità di mantenere aggiornata la tabella con le associazioni da IP a locazione fisica, compensa i miglioramenti ottenuti.

3.4.2 Greedy Perimeter Stateless routing (GPSR)

Descrizione

Il GPSR [KK00] è uno degli algoritmi più significativi della categoria in quanto riesce a sfruttare al meglio i vantaggi offerti dalla localizzazione geografica dei nodi della rete: ogni nodo non deve mantenere nessuna informazione se non l'elenco aggiornato dei nodi vicini.

L'algoritmo funziona in questo modo: ogni nodo, quando riceve un pacchetto, lo smista al suo vicino con la distanza minore verso la destinazione. Non occorre mantenere nessuno stato né scambiarsi informazioni di routing di qualsiasi genere.

Sfortunatamente può capitare di trovarsi in un massimo locale e quindi di non avere nessun nodo che permetta di avvicinarsi ulteriormente a destinazione.

In questo caso l'algoritmo commuta in modalità "perimetro", per cercare di girare attorno al perimetro della zona vuota e come regola per scegliere il nodo successivo usa quella della "mano destra", cioè sceglie il nodo collegato al primo arco trovato in senso antiorario a partire dall'arco da dove il pacchetto è arrivato.

Questo metodo non entra in loop se il grafo è planare, cioè se non sono presenti sul grafo archi che si intersecano.

Nel caso in cui il grafo non sia planare occorre renderlo tale, ed il GPRS riesce a tale scopo utilizzando sempre e solo i diretti vicini di un nodo.

Considerazioni

Il GPSR è un algoritmo che, sfruttando la localizzazione geografica nel modo migliore, riesce ad ottenere una buona scalabilità sia all'aumento delle dimensioni della rete che all'aumento della mobilità dei nodi. Inoltre in ogni nodo le informazioni da mantenere sono $O(M)$ con M il numero dei vicini e quindi sono costanti rispetto alla grandezza della rete.

Per contro presenta i seguenti difetti:

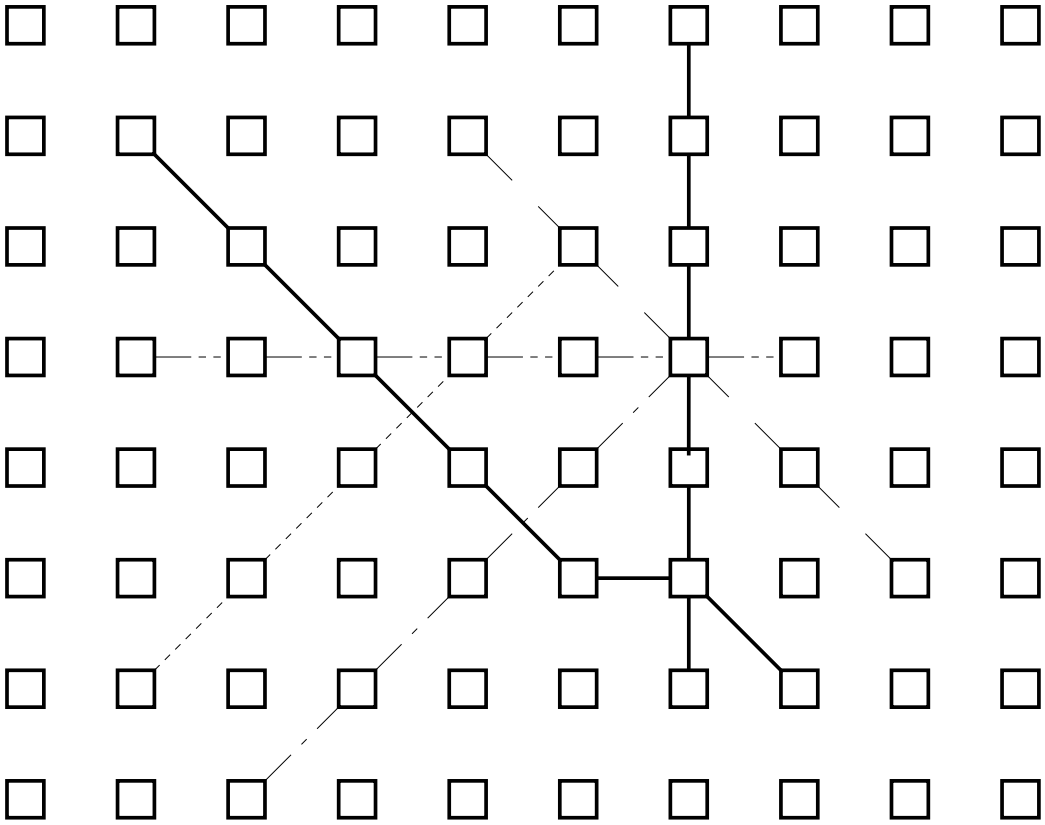

Figura 3.2: Traffico in una mesh

- i percorsi trovati possono non essere ottimali sia nel numero di nodi attraversati, sia nel costo degli archi;
- l'algoritmo di planarizzazione può non funzionare nel caso in cui il nodo che lo calcola non riesca a collegarsi a tutti i suoi vicini a causa di ostruzioni fisiche (muri, alberi ecc.);
- come si vede in figura 3.2 , GPRS tende a far concentrare il traffico al centro della mesh.

3.5 Gli algoritmi gerarchici

Gli algoritmi di questo tipo cercano di ottenere una maggiore scalabilità (quindi di poter utilizzare reti wireless di più ampie dimensioni) impo-

stando una gerarchia di host che si dividono la responsabilità del routing.

A differenza del modello di Internet dove l'assegnazione dei ruoli nella gerarchia viene effettuata da un'autorità centrale in maniera manuale, questi algoritmi si autoconfigurano e cercano di trovare una gerarchia che rispetti la topologia della rete in quel momento. Inoltre la maggior parte di questi algoritmi, pur essendo gerarchica usa uno spazio di indirizzamento piatto (senza cioè per esempio netmask e prefissi come in Internet)

3.5.1 Clusterhead Gateway Switch Routing (CGSR)

Descrizione

Il CGSR [CWLG97] è un algoritmo di routing gerarchico a due livelli con creazione automatica dei gruppi di nodi (cluster) tramite un metodo chiamato "least clusterhead change" (LCC).

In ogni cluster viene automaticamente eletto un nodo detto "clusterhead", mentre i nodi che appartengono contemporaneamente a due (o più) cluster vengono definiti "gateway".

L'algoritmo utilizza il metodo "distance vector", ma ogni nodo ha una tabella differente a seconda del ruolo che svolge.

I nodi semplici mantengono una tabella di un solo elemento: il loro clusterhead.

Il clusterhead mantiene nella sua tabella gli indirizzi degli altri nodi del suo stesso tipo.

Il routing funziona così: il nodo invia il pacchetto al suo clusterhead che lo invia tramite un nodo gateway verso un clusterhead più vicino alla destinazione. Quindi il nodo segue un percorso del tipo clusterhead → gateway → clusterhead → gateway ... fino al raggiungimento dell'ultimo clusterhead e quindi della destinazione.

Considerazioni

L'algoritmo riduce sia l'overhead dovuto al flood dei pacchetti DV che la dimensione dei pacchetti stessi.

L'overhead dovuto al mantenimento dei cluster non è eccessivo grazie al metodo LCC.

Gli svantaggi sono dovuti al fatto che i clusterhead ricevono un traffico dati superiore a quello degli altri nodi e in più la lunghezza dei percorsi non è ottimale, in quanto non viene scelto il percorso più breve, ma quello passante per clusterhead e gateway.

3.5.2 Multimedia Mobile Wireless Networks (MMWN)

Descrizione

Il MMWN [RS98] è una soluzione completa per reti wireless senza infrastrutture che fornisce oltre al routing anche servizi "Quality Of Service".

L' approccio è gerarchico a più livelli: i nodi si auto organizzano in cluster (livello zero) e poi i cluster si raggruppano ricorsivamente in cluster via via di più grandi dimensioni (livello uno due ecc.). In caso che un cluster contenga troppi elementi (dove gli elementi possono essere nodi o clusterdi livello $n - 1$) è previsto che si autodivida in due, mentre se diventa di dimensioni ridotte si autoaggrega ad altri.

In ogni cluster ci sono uno o più nodi che sono eletti a switch per smistare il traffico verso gli altri cluster: in maniera intelligente i nodi eletti a switch sono quelli che si trovano sul "bordo" fra due cluster; infatti visto che il traffico dovrebbe comunque passare fra di loro è meglio dare direttamente a loro il compito di gestirlo.

Ogni nodo appartiene ad una gerarchia di cluster dal livello zero in poi. Il suo indirizzo univoco si indica con una serie puntata dei cluster a cui appartiene (per es.: A.B.C.D). Ovviamente tenuto conto del fatto che la lunghezza dell' indirizzo del nodo è variabile e che l' indirizzo del nodo può cambiare del tempo (in quanto i nodi si muovono e possono cambiare quindi cluster di appartenenza) occorre un servizio che associ l'indirizzo MMWN attuale del nodo al suo indirizzo ip.

Il servizio deve gestire gli aggiornamenti delle posizioni dei nodi e di ricerca di un nodo, e viene svolto in maniera ricorsiva dagli stessi switch.

Ogni switch ha una visione gerarchica della rete, più dettagliata nei riguardi del suo cluster e sotto cluster e meno dettagliata degli altri. Calcola il percorso attraverso una variante dell'algoritmo di Dijkstra e lo inserisce nel pacchetto stesso (per es.: A.B.C $\rightarrow$ A.B.D $\rightarrow$ A.B $\rightarrow$ A $\rightarrow$ E $\rightarrow$ F). Il percorso non è completamente definito alla partenza (infatti al di fuori del cluster di appartenenza vengono indicati solo i nomi dei cluster di livello max), ma viene precisato meglio man mano che il pacchetto dalla sorgente si avvicina alla destinazione.

Nel caso in cui dalla sorgente alla destinazione passino una grande quantità di pacchetti, per risparmiare l' overhead del source routing e per garantire la qualità del servizio viene creato un circuito virtuale dalla sorgente alla destinazione.

Nel caso di interruzione del circuito virtuale (frequente in caso di mobilità dei nodi) sono previste addirittura 3 metodi per cercare di ricostruirlo:

handoff dove lo switch cerca un diretto vicino che possa sostituire il nodo guasto;

local reroute dove lo switch genera una richiesta di nuovo route verso il primo switch del circuito ancora funzionante;

end reroute dove viene inviato alla sorgente un messaggio per ricreare completamente il route e il relativo circuito.

Considerazioni

L'algoritmo si pone obiettivi ambiziosi che sono in parte raggiunti. La parte più critica è il servizio di associazione: è infatti molto costoso in termini di sovraccarico della rete mantenere una struttura aggiornata e distribuita degli abbinamenti fra indirizzo gerarchico e indirizzo ip.

3.5.3 Terminode Routing (TR)

Descrizione

Il Terminode Routing [BBG02] è un algoritmo di routing gerarchico a due livelli con interessanti tecniche euristiche di ottimizzazione dei percorsi in caso di reti "sparse" cioè con varie zone in cui sono assenti nodi.

Per le destinazioni vicine alla sorgente viene utilizzato il Terminode Local Routing (TLR) che è un classico algoritmo proattivo che non usa nessuna informazione geografica.

Se la destinazione non è compresa nei nodi controllati dal TLR entra in azione il Terminode Remote Routing (TRR) che è un algoritmo geografico basato sul GPRS con l'aggiunta delle "ancore".

Un'ancora è un punto sul territorio (scelto da un operatore umano) che segna un percorso preferenziale per il passaggio dei pacchetti.

Il TRR funziona quindi in questo modo: inizialmente viene trovato un percorso con il metodo GPRS; viene poi effettuato un confronto fra la lunghezza del percorso trovato e una stima della lunghezza di un percorso ideale. Se il percorso è molto più lungo del previsto viene effettuata una nuova ricerca passando però attraverso i punti consigliati dalle ancore.

In questo ultimo caso si usa una forma di source routing approssimativo: nei pacchetti spediti vengono indicati solamente i punti obbligatori da attraversare (le ancore) e si lascia ai singoli nodi attraversati la decisione locale di quale vicino scegliere.

Considerazioni

L'algoritmo presenta alcune interessanti soluzioni, in quanto comprende che il principale problema del GPSR è che non sempre fornisce il percorso più breve: quindi fornisce un metodo per capire quando il percorso è eccessivamente lungo e un metodo per trovarne uno migliore.

Capitolo 4

IPV6

In seguito allo sviluppo enorme di Internet in questi ultimi decenni sono esarcerbati alcuni limiti del protocollo IPV4 e nello stesso tempo sono emerse nuove necessità.

In particolare un indirizzo ip di 32 bit permette di distinguere poco più di 2 miliardi di indirizzi che, anche se fossero tutti sfruttabili (e, come detto nel paragrafo 2.2 non lo sono), non bastano per garantire un indirizzo ip per ogni abitante del pianeta. E le previsioni per il futuro sono che ogni abitante del pianeta possederà più di un personal computer e anche altri dispositivi dotati di ip (telefoni cellulari ecc.).

Inoltre sono nate nuove esigenze di sicurezza (cifratura dati, reti private virtuali) e di qualità del servizio (convergenza delle reti telefoniche in quelle a pacchetto) che non sono facilmente gestibili col protocollo IPV4.

Quindi, dopo una lunga gestazione, è stato ideato il nuovo protocollo IP versione sei allo scopo di risolvere tutte queste nuove esigenze.

4.1 Indirizzo IP e auto-configurazione

Per "tagliare la testa al toro" la dimensione del nuovo indirizzo ip è stata portata non a 64 bit ma bensì a 128 bit, in modo da avere i seguenti vantaggi:

- la sicurezza che non vi saranno più carenze di ip anche in caso di assegnazioni di più ip allo stesso dispositivo o di non utilizzazione totale dello spazio ip;

- la possibilità di inglobare nell'indirizzo ip l'intero indirizzo di livello due (per es.: tutti i 48 bit dell'indirizzo ethernet), in modo da semplificare la configurazione delle reti;

- una maggiore versatilità nella creazione di indirizzamenti gerarchici;

- la possibilità di effettuare sperimentazioni, come per esempio l'inserimento delle coordinate geografiche di un dispositivo all'interno dell'indirizzo ip.

Il protocollo "Address Resolution Protocol" (ARP) dell'IPv4 è stato abolito; ora ogni dispositivo IPv6 all'attivazione si crea da solo un indirizzo valido per la rete fisica a cui è collegato chiamato "link local address" (LLA) [TN98].

Un LLA è costituito da 64 bit di prefisso chiamato Link Local Prefix (LLP) e da 64 bit di indirizzo in formato EUI-64.

La parte EUI-64 viene creata automaticamente dal dispositivo utilizzando l'indirizzo hardware della sua interfaccia di rete, per esempio i 48 bit dell'indirizzo ethernet.

Il LLA viene comunicato agli altri host presenti nella rete col protocollo Neighbor Discovery Protocol (NDP) [NNS98] per assicurarsi dell'univocità dell'indirizzo e per fare in modo che eventuali router presenti possano comunicare indirizzi aggiuntivi all'host.

4.2 Mobile IP

Per mobile ip si intende lo scenario in cui un host collegato ad Internet possa muoversi e quindi scollegarsi dalla rete originaria e ricollegarsi da un altro punto, mantenendo però tutti i suoi collegamenti con i suoi host remoti, che non devono accorgersi che l'host si è spostato.

La soluzione proposta per l' Internet attuale basata su ipv4 è subottimale ed è stata scelta in quanto non è possibile cambiare il funzionamento di tutti i router già presenti in Internet.

Il funzionamento è questo: all'host viene assegnato un indirizzo iniziale ("home address") dalla prima rete a cui si collega; nelle reti che supportano il mobile ip vi è almeno un router che supporta le seguenti funzionalità aggiuntive e viene chiamato "home agent".

Quando l'host si muove e ottiene un nuovo indirizzo ip dalla nuova rete in cui è collegato lo comunica all'home agent e crea un tunnel con esso.

L'home agent nella rete originaria fa le veci dell'host e mantiene per lui tutte le connessioni esistenti. Ogni volta che riceve pacchetti ip li rispedisce all'host alla sua posizione attuale attraverso il tunnel prima creato.

L'inefficienza di questa soluzione si spiega con il seguente esempio detto "il problema della triangolazione del routing": il route diretto fra due host A e B può essere migliore del route della triangolazione A $\rightarrow$ home agent di A $\rightarrow$ B, ma il mobile ip prevede solo il secondo caso.

Per contro nella nuova versione prevista per l'ipv6 [PJ96] si è appunto tenuto in considerazione questa inefficienza ed è prevista la possibilità per l'home agent di A di informare il router a cui è collegato B del nuovo indirizzo di A in modo che i pacchetti possano seguire il percorso diretto da A a B.

4.3 Proposte per indirizzamento geografico

Le "abbondanti dimensioni" del nuovo indirizzo IPv6 hanno permesso ai ricercatori la possibilità di studiare indirizzamenti alternativi al CIDR.

In particolare è stata considerata l'opportunità di codificare le coordinate geografiche di un nodo all'interno del suo indirizzo IP, in maniera tale di fornire un indirizzamento indipendendente dai provider (come accadeva nell'era pre CIDR).

Sono state presentate quindi due proposte, la prima [Leo02] assegna 96 bit per rappresentare le tre coordinate (latitudine, longitudine e altezza dal suolo) garantendo così una precisione millimetrica sulla posizione di un nodo.

La seconda [Hai02] invece utilizza solamente 44 bit per le coordinate (solamente latitudine e longitudine) in maniera tale da preservare 64 bit per includere l'indirizzo EUI-64 del nodo.

Capitolo 5

LE RETI ROOFTOP

I progressi nel campo informatico hanno abbassato notevolmente il costo dei dispositivi per creare collegamenti wireless fra computer. Questo ha permesso a gruppi di "entusiasti" (radioamatori, professionisti in informatica, reti, telecomunicazioni) di iniziare a collegarsi fra di loro e formare delle comunità wireless.

Nel mondo sono emerse numerose comunità di questo tipo [coma]; i loro partecipanti sono stati contattati per ricevere informazioni sui loro obiettivi, sulle loro necessità e sulle scelte tecniche da loro intraprese.

Gli obiettivi di questi gruppi in genere sono:

- la possibilità di fare arrivare un collegamento ad internet ad alta velocità in zone dove le tradizionali soluzioni wired non sono adatte o convenienti (aree rurali);
- la possibilità di comunicare con altre persone e scambiare dati, senza dover collegarsi a pagamento ad internet;
- la promessa di migliori prestazioni di un collegamento diretto (specialmente bassa latenza) rispetto ad un collegamento indiretto tramite provider;
- la migliore comprensione della tecnologie di rete, e la possibilità di sperimentare nuove idee in prima persona;

- la creazione di una rete alternativa ad Internet più libera e meno controllata dalle autorità.

5.1 Il modello

Il modello è simile a quello indicato in figura 2.1 tenendo però conto delle seguenti considerazioni:

- i nodi possono essere collegati fra di loro con un collegamento da punto a multi-punto, in tal caso usano un'antenna omnidirezionale;
- i nodi possono essere anche collegati con un collegamento punto a punto, per esempio con due schede wireless e antenne direzionali oppure con collegamenti laser o infrarosso;
- la mobilità dei nodi è nulla, in quanto le antenne o gli apparati laser vengono posizionati con accuratezza sul terreno (o sui tetti degli edifici) per ottenere i migliori risultati di comunicazione in termini di distanza e qualità del segnale;
- l'affidabilità dei nodi è media: non sono in genere costruiti in maniera "fault tolerant", però vengono in genere lasciati accessi ventiquattro ore su ventiquattro;
- in genere un nodo è collegato direttamente con i nodi geometricamente più vicini a lui; questo però non sempre è vero sia per motivi orografici (muri, alberi, ecc.) che per motivi "umani" (per es.: i proprietari dei due nodi non si conoscono)
- la dimensione della rete (in termini di numero di nodi) può crescere fino a raggiungere dimensioni considerevoli, in quanto gli utilizzatori intendono realizzare un sostituto di Internet;
- non c'è un backbone ad alta capacità di trasmissione verso cui far convergere tutto il traffico, al contrario tutti i nodi hanno larghezza di banda e latenza omogenee;

- la larghezza di banda di ogni nodo è ridotta: nessun nodo può gestire un numero di elevato di connessioni contemporanee (per es.: server http);
- la maggior parte del traffico avviene fra nodi vicini (in numero di hop);
- i collegamenti fra nodi sono bidirezionali;
- ogni nodo è collegato direttamente con vari altri quindi il fenomeno del multihoming è diffuso;
- non c'è un' organizzazione centrale che assegna indirizzi ip o che prenda decisioni sull' aggiunta o la rimozione dei nodi;
- alcuni nodi possono offrire un collegamento ad Internet, ma in genere di bassa capacità e non continuativo.

5.2 Gli algoritmi

Non esiste nessun algoritmo specifico per reti wireless rooftop. In tutti i casi vengono utilizzati gli algoritmi per reti MANET, sebbene questi ultimi siano specifici per reti di piccole dimensioni e non mirino ad ottenere il percorso migliore a causa della mobilità dei nodi, mobilità che nelle reti rooftop è assente.

La maggior parte delle comunità utilizza AODV, probabilmente perché ne esiste già un'implementazione funzionante sotto Linux.

La comunità di Seattle negli USA addirittura ha organizzato la sua rete fisica in maniera gerarchica per usare OPSF.

5.3 Considerazioni

Se ci troviamo nella situazione in figura 5.1 con una rete densa, e, per un momento, non consideriamo il possibile traffico nei nodi, possiamo ovvia-

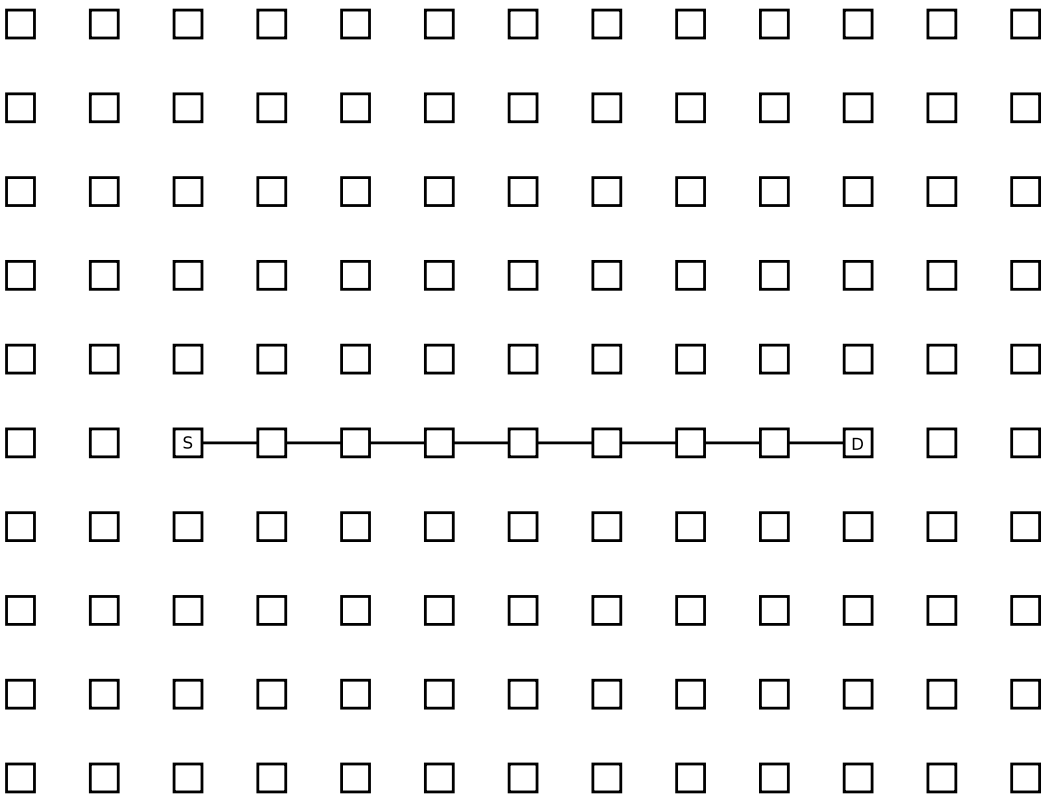

Figura 5.1: Rete densa

mente dire che il percorso più breve è quello che comprende i nodi nella retta che congiunge la partenza con la destinazione.

Le difficoltà sopraggiungono nella situazione in figura 5.2, dove fra i due nodi c'è un "vuoto" che impedisce di stabilire un percorso in linea retta fra i due. In questo caso il percorso migliore è quello composto dai due spezzoni di rette SY ed YD indicate in figura.

Se osserviamo un attimo come si comportano gli algoritmi geografici e gerarchici, notiamo per esempio che il GPSR arriva fino al nodo X poi gira attorno all' ostacolo e infine arriva a destinazione, ma il percorso trovato non è ottimale.

L'algoritmo Terminode invece trova il percorso ottimale, a condizione però che un operatore umano posizioni un'ancora nel punto Y.

L'algoritmo AODV, trova anche lui il percorso ottimale, ma ad un alto prezzo: non avendo visione topologica della rete deve chiedere a tutti i

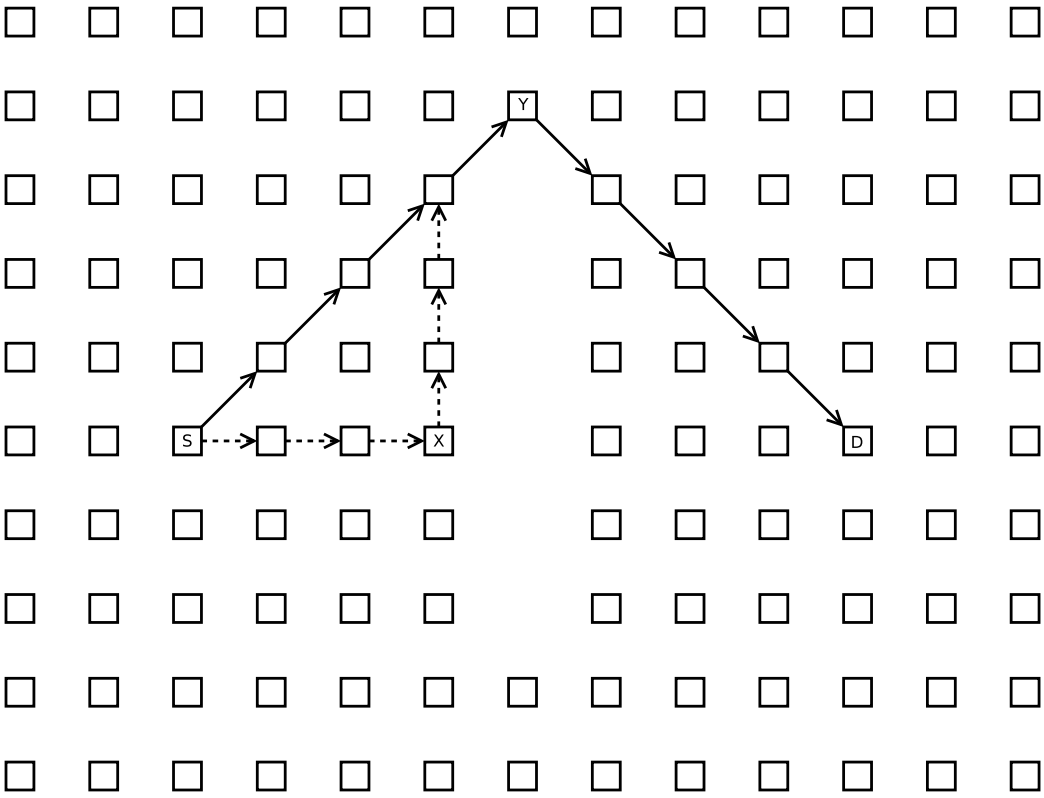

Figura 5.2: Rete sparsa

nodi della rete se conoscono il percorso per la destinazione.

Possiamo quindi capire senza ulteriori esempi che è fondamentale, se si sceglie di utilizzare un algoritmo di tipo geografico, trovare un metodo per evidenziare e quindi aggirare i vuoti nella rete, qualunque dimensione questi abbiano.

Inoltre se si desidera realizzare un algoritmo proattivo con reti di grandi dimensioni occorrerà prendere in considerazione l'idea di utilizzare un indirizzamento gerarchico (per es.: netmask).

5.4 Un primo tentativo

Inizialmente si era pensato di realizzare un algoritmo proattivo gerarchico con le seguenti caratteristiche (vedi figura 5.3):

- le coordinate del nodo devono essere codificate dentro l'indirizzo ip (e questo è possibile farlo con sufficiente precisione solo usando i 128 bit dell' indirizzo ipv6); poiché i nodi non si muovono in questo modo si può fare a meno del servizio di localizzazione e del relativo overhead;

- se le coordinate vengono codificate in maniera opportuna la netmask può indicare una zona quadrata del territorio tanto più grande quanto il netmask è piccolo (nella figura un netmask /64 indica l'area grigio chiara, /62 quella grigio scura);

- il routing è gerarchico a N vie, ogni nodo ha una visione completa delle aree vicine (con il netmask più selettiva), mentre ha una visione "riassuntiva" di quelle più distanti;

- i nodi che si trovano ai bordi fra una zona e l'altra svolgono il compito di router (in quanto comunque il traffico passerebbe attraverso loro);

- ogni router appartiene ad un livello, di numero più elevato a seconda della grandezza delle due aree che mette in comunicazione (ipotiz-

ziamo di partire dal livello uno per l' area coperta da un netmask /64);

- i router raccolgono informazioni sui nodi e gli archi presenti nella loro area di competenza e ne preparano un sunto da inviare alle altre aree.

Se osserviamo l'esempio in figura 5.4 notiamo come il nodo S, analizzando "dall'alto" la rete, riesca ad accorgersi quale sia il percorso migliore (che non è quello sulla linea congiungente S con D). A questo punto basta che invii i pacchetti verso il router R che avrà una visione migliore della rete nelle sue vicinanze (analogamente a quanto avviene in FSR) e saprà scegliere il percorso migliore verso la destinazione.

5.4.1 I difetti

Negli esempi visti fino ad adesso il funzionamento di questa soluzione proattiva geografica e scalabile ad N livelli sembra ineccepibile. Questo accade perché gli esempi sono stati scelti con cura per massimizzare la relazione fra "distanza geografica" e "distanza in numero di hop", e rappresentano inoltre una visione statica della rete.

Nei seguenti esempi mostreremo cosa accade quando due nodi, seppure vicini geograficamente, siano separati da un considerevole numero di nodi intermedi.

5.4.2 Nodi lontani

Come si vede in figura 5.5 i router A e B e i relativi nodi appartengono tutti allo stesso quadrante, ma per comunicare fra di loro devono attraversare altri quadranti questo causa:

- la propagazione del traffico di routing verso i nodi del quadrante uno attraverso un percorso più lungo del dovuto, quindi la rete viene sovraccaricata;

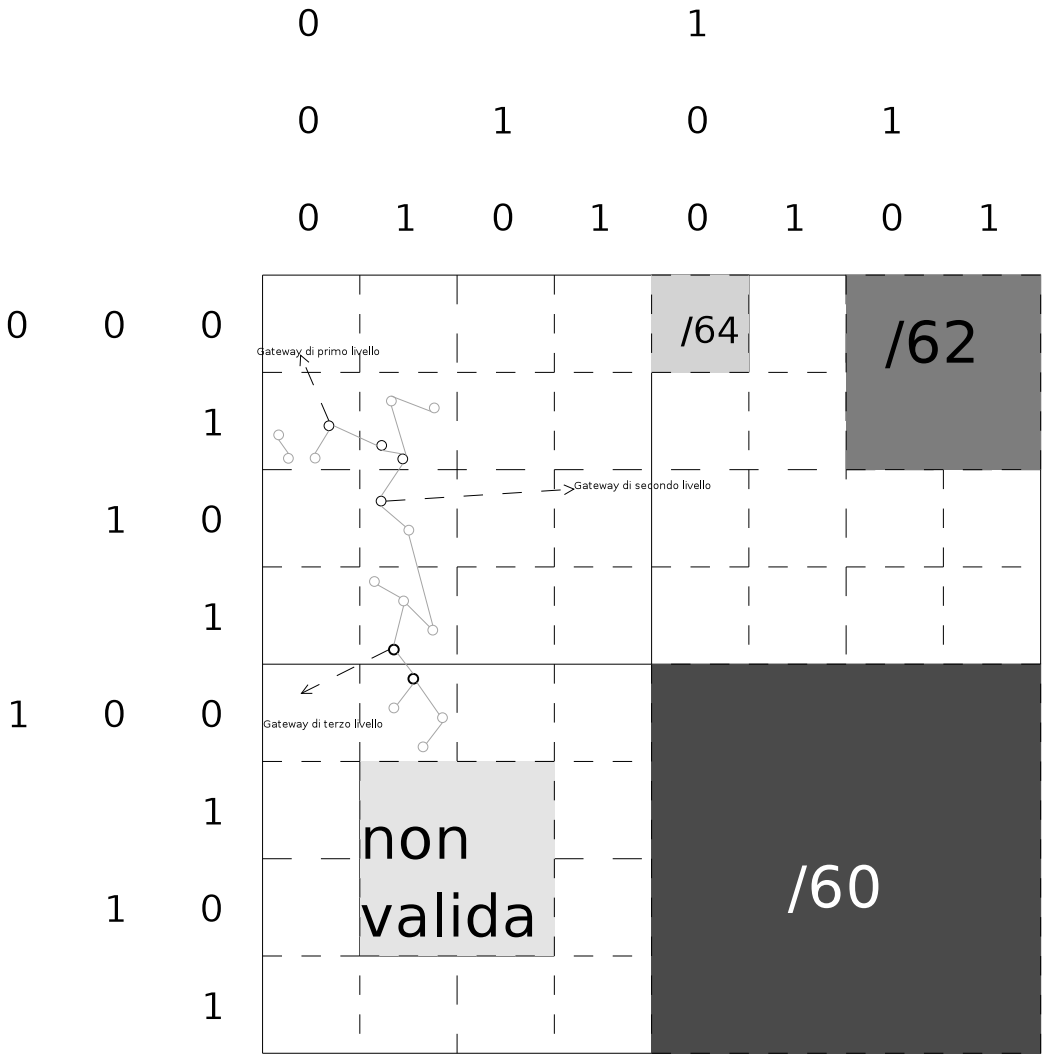

Figura 5.3: Visione geografico/gerarchica

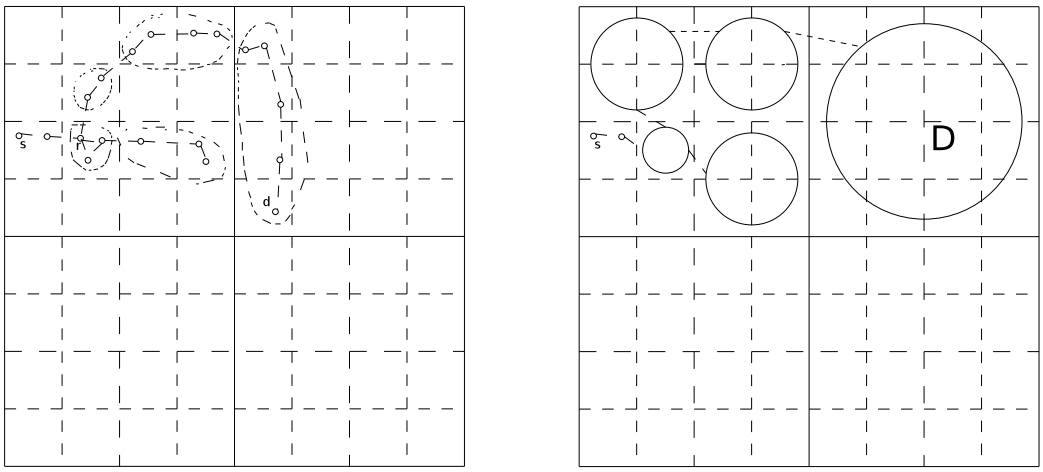

Figura 5.4: Visione gerarchica dal nodo S

- il malfunzionamento dell' idea base dell' algoritmo in quanto i nodi del quadrante uno non sanno come trattare e a chi inviare le informazioni di routing "interne" dei nodi del quadrante zero.

5.4.3 Route fantasma

Anche se si riescono a risolvere i problemi della sezione precedente e si riescono a mettere in comunicazione fra di loro nodi dello stesso quadrante ma non collegati fra di loro ci si imbatte comunque in un problema ben più grave indicato a destra in figura 5.5.

Come si può ben vedere la sintesi delle informazioni che viene passata ai livelli più alti della rete suggerisce che vi sia un percorso fra il nodo a e b cosa che effettivamente non esiste.

Non si è trovata una soluzione efficiente a questo problema. Si è pensato di utilizzare una parte dello spazio di indirizzi per distinguere più gruppi di nodi appartenenti alla stessa zona geografica.

Ma in tal caso emergono problemi di gestione su chi assegna questi indirizzi ai nodi e, in caso di variazione di topologia della rete, gli indirizzi devono variare di conseguenza (perché non più associati solamente alla posizione): non è quindi più possibile utilizzarli direttamente come indirizzi ipv6 e bisogna creare un servizio di locazione come in MMWN.

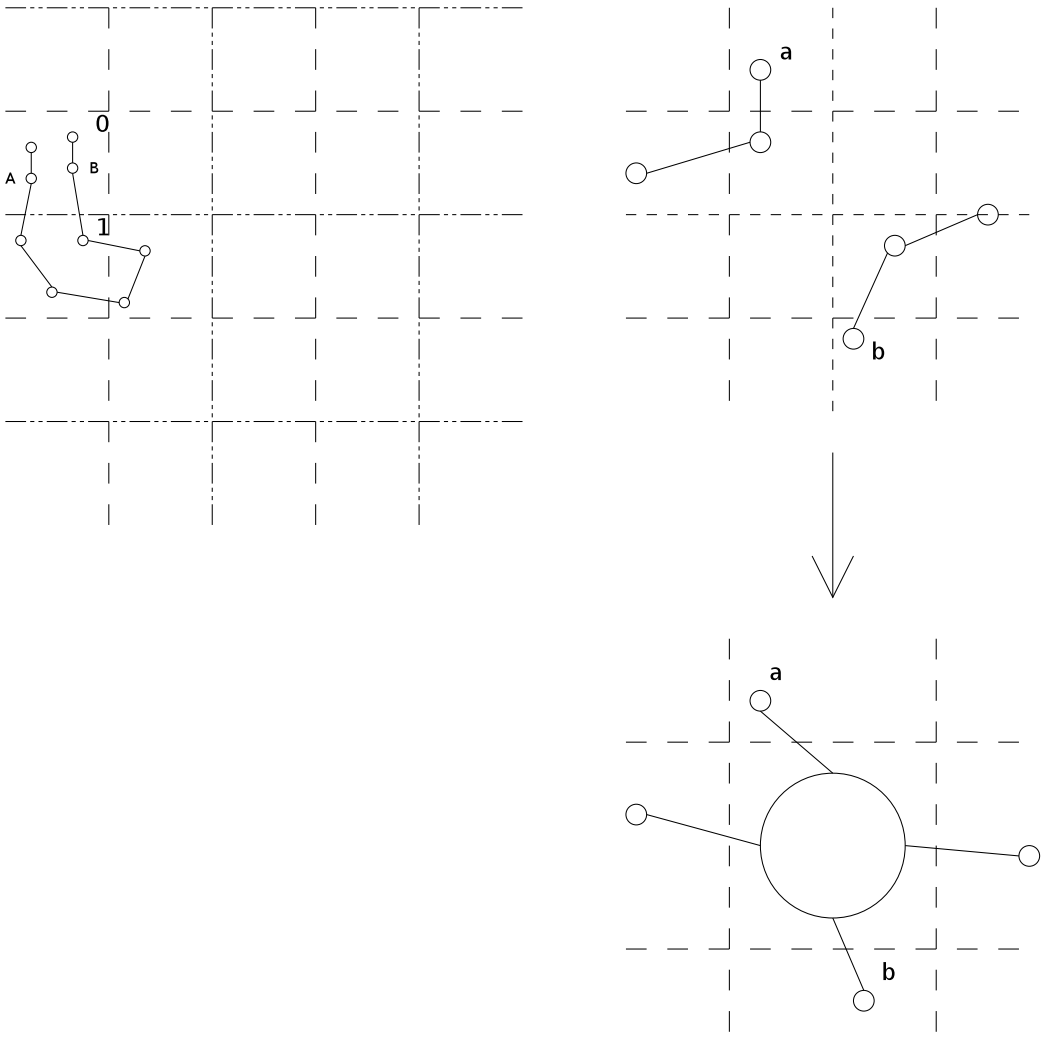

Figura 5.5: Problemi del routing gerarchico

5.4.4 Multihoming

Utilizzando un approccio gerarchico con l' indirizzamento pure gerarchico (netmask) ci si ritrova in una situazione molto simile all'Internet tradizionale e ci si ritrova il problema del multihoming, qui notevolmente amplificato dal fatto che la topologia della rete non assomiglia minimamente ad un albero, ma ad una griglia (mesh), con praticamente tutti gli host dotati di più di un collegamento.

5.4.5 Conclusioni

L' approccio proattivo/gerarchico multilivello, con creazione automatica delle gerarchie, pur essendo a prima vista particolarmente attraente, si rivela con diversi vizi di fondo, che, anche se fossero risolvibili, ne fanno comunque perdere gli iniziali vantaggi.

Rimane ora da sperimentare una alternativa, magari basata su un approccio reattivo come vedremo nei successivi capitoli.

Capitolo 6

SHORTCUT ROUTER

In questo capitolo sarà presentato un nuovo approccio per il routing specifico per reti wireless esclusivamente di tipo rooftop (non MANET quindi).

Le necessità per le comunità che utilizzano questo tipo di reti sono sufficientemente diverse per richiedere uno specifico algoritmo e non un adattamento di altri già esistenti.

In particolare si richiede un algoritmo che:

- scali all'aumentare del numero di nodi della rete, in quanto si prevedono in futuro reti rooftop di grandi dimensioni;
- fornisca non solo un percorso, ma possibilmente un percorso di buona qualità (di lunghezza o costo minimo).

Poiché la larghezza di banda di ogni singolo nodo non è elevata invece non è richiesto un algoritmo che scali all'aumentare del numero di route.

6.1 L'idea

L'approccio scelto è quello reattivo e geografico, in maniera simile all'algoritmo GPRS.

A grandi linee l'algoritmo si comporta in questa maniera: si inizia con una fase greedy di ricerca percorso, e nel caso in cui si finisca in un massimo locale viene eseguito un "backtracking geometrico".

Una volta trovato un percorso valido l'algoritmo effettua una fase di ottimizzazione del percorso dove identifica i tratti del percorso non lineari e cerca di ottimizzarli, sostituendoli con percorsi in linea retta (le "scorciatoie", da cui il nome).

L'algoritmo si differenzia dal GPSR nel fatto che quest'ultimo non ha neppure una fase iniziale di ricerca route necessaria in questo caso per avere in seguito le informazioni necessarie per ottimizzare il percorso; inoltre si è scelto di rimpiazzare la modalità perimetro del GPSR con un "backtracking" per evitare i problemi descritti nel paragrafo 3.4.2.

Poichè i nodi non si muovono, si possono codificare le loro posizioni geografiche all'interno dei loro indirizzi IP, evitando in questo modo anche il costo del servizio di localizzazione.

In questo modo si risolve anche il problema di assegnare un indirizzo univoco ad ogni nodo senza bisogno di un autorità centrale in quanto evidentemente la posizione geografica di un nodo è univoca.

6.2 La fase di ricerca percorso

In seguito alla richiesta di un nuovo percorso, viene preparato un pacchetto di richiesta route, inizialmente vuoto, per tenere traccia del percorso effettuato (e quindi non cadere in possibili loop) e dei punti "scorciatoia" (che vedremo in seguito).

Tale pacchetto viene inviato in direzione della destinazione; per fare ciò si sceglie il vicino con la distanza geometrica minore dal nodo finale, dando la precedenza ai vicini che non fanno parte del percorso già visitato.

In questo modo di procedere può capitare che, una volta scelto il vicino, ci si trovi ad una distanza maggiore rispetto alla destinazione; in tal caso l' algoritmo marca il nodo corrispondente al massimo locale. Quando poi, nel proseguire la ricerca, si trova un nodo più vicino alla destinazione rispetto al massimo locale, lo si aggiunge alla lista dei "punti scorciatoia".

Può anche accadere di trovarsi in un "vicolo cieco", cioè in un nodo i cui vicini appartengano tutti al percorso già visitato: in tal caso si marca

il nodo come "da evitare" (in tal modo non sarà più inserito in futuro in nessuna lista dei vicini) e si continua come al solito scegliendo il vicino con la distanza inferiore alla destinazione (ignorando quindi il fatto che tutti i vicini facciano parte di un percorso già seguito), e cancellando la parte di percorso che ha portato a quel nodo.

Una volta raggiunta la destinazione viene preparato un pacchetto di risposta ed inviato a ritroso nel percorso trovato, sia per confermare il percorso ai singoli nodi che per attivare la ricerca delle scorciatoie.

6.3 Le scorciatoie

In seguito alla ricerca del percorso ci troviamo con una lista di punti definiti "punti scorciatoia". Questi punti indicano dove l'algoritmo riesce ad avvicinarsi verso la destinazione.

Questi punti indicano però un'altra cosa molto più importante: il luogo esatto dove l'algoritmo è riuscito ad aggirare un "ostacolo" (inteso come zona con assenza di nodi).

E, per aggirare un ostacolo, l'algoritmo ha sicuramente proceduto in maniera non lineare, passando probabilmente per più nodi del necessario.

Quindi l'algoritmo, per ottimizzare il percorso trovato, opera in questo modo: per ogni punto scorciatoia n fa partire una ricerca percorso avente come destinazione il punto scorciatoia $n + 1$.

6.4 Casi particolari

In questa sezione riprenderemo alcuni dettagli tralasciati nei precedenti paragrafi.

In alcuni casi verranno proposte più soluzioni alternative; come vedremo nel capitolo 7 alcuni membri delle comunità wireless stanno partecipando con suggerimenti e richieste al miglioramento dell'algoritmo proposto.

6.4.1 Destinazione non trovata

In caso che la destinazione non esista o che esista ma sia irraggiungibile l'algoritmo termina la ricerca dopo un numero finito di tentativi. Tale numero può essere fissato a priori come costante oppure può essere calcolato euristicamente in base alla distanza fra sorgente e destinazione e quindi al presunto numero di nodi che sarebbero necessari per un collegamento in linea d'aria.

6.4.2 Controlli periodici

Nel caso in cui non vi sia un collegamento in linea retta fra sorgente e destinazione, o che il costo del percorso trovato non sia soddisfacente, si può periodicamente inviare una nuova ricerca di percorso.

La frequenza di tale ricerca è stata stabilita ad ogni ora, questo in attesa di riscontri pratici delle varie comunità.

6.4.3 Interruzione del percorso

Ogni nodo controlla periodicamente lo stato dei vicini, mediante l'invio di un pacchetto "hello". Nel caso in cui si renda conto che uno dei suoi vicini non risponda, e che questo faccia parte di un percorso attivo, provvede a far partire una nuova ricerca di percorso per quella destinazione.

In alternativa è stato richiesto che il nodo che si accorge del guasto non prenda iniziative, ma che invii un pacchetto di "route error" al nodo partenza del percorso, che provvederà alla ricerca di un nuovo route.

6.4.4 Latenza creazione route

Come detto in precedenza i pacchetti dati iniziano a transitare nel percorso trovato solamente alla fine della fase di "conferma percorso", durante la quale viene aggiornato il "forwarding table" di tutti i nodi con le informazioni sul percorso trovato e sul costo.

Questo però comporta che il primo pacchetto dati possa transitare solamente dopo il tempo necessario per trovare il percorso completo.

In maniera simile al GPRS è però ipotizzabile una variante dell'algoritmo in cui i pacchetti dati vengano inviati assieme al pacchetto di richiesta route, in tal caso basta che i nodi interessati inseriscano nel loro forwarding table le informazioni parziali del percorso trovato fino a quel momento.

6.4.5 Evitare nodi sovraccarichi

Come si è visto nel paragrafo 3.4.2 in una rete conformata a griglia gli algoritmi (in particolar modo quelli geografici) tendono a concentrare tutto il traffico al centro della rete. È quindi possibile ipotizzare una variante dell'algoritmo in cui in collegamenti sovraccarichi vengano considerati come scollegati in modo che l'algoritmo li aggiri con il backtracking.

6.5 Esempi

Per chiarire ulteriormente il funzionamento vengono ora presentati esempi funzionamento dell'algoritmo in alcune topologie comuni.

6.5.1 Primo esempio

Il caso ipotizzato in figura 6.1 è uno dei più comuni: la rete densa (tutti i nodi possono essere collegati fra di loro con percorsi approssimanti una linea retta).

In figura non è stato indicato, ma si sottointende che ogni nodo abbia un collegamento wireless con gli otto nodi vicini a lui.

Il nodo S vuole inviare dei pacchetti al nodo D quindi inizia la procedura di ricerca percorso. Viene preparato un pacchetto di richiesta route e viene inviato al nodo più vicino alla destinazione che in questo caso è il nodo 1.

Il pacchetto segue quindi il percorso $2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$.

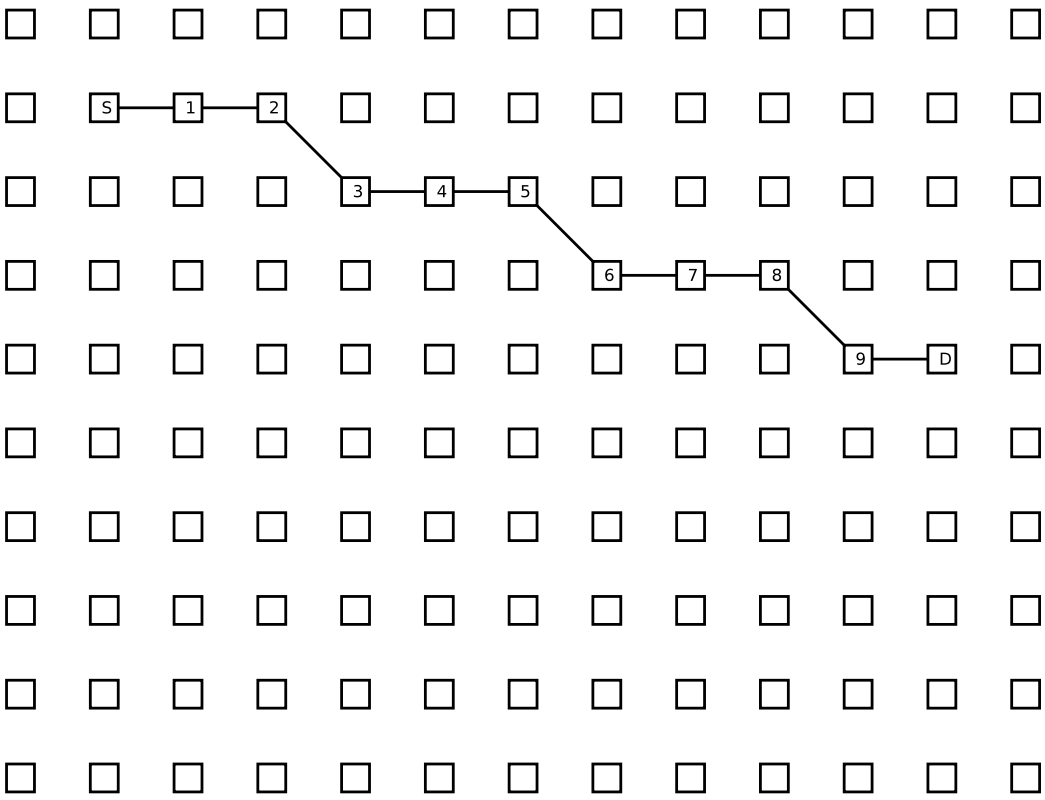

Figura 6.1: Esempio 1

Giunti al nodo 9 si nota che il nodo D è nella lista dei vicini, quindi è terminata la fase di ricerca e inizia la fase di conferma percorso.

Viene quindi preparato il pacchetto di risposta e viene poi inviato a ritroso verso il nodo S.

Il primo nodo ad essere raggiunto dal pacchetto di risposta è quindi il nodo 8: nella sua forwarding table viene aggiunta l'informazione che, per raggiungere il nodo D occorre passare dal nodo 9.

Quindi quando il pacchetto di risposta raggiunge il nodo S tutti nodi attraversati avranno il forwarding table aggiornato adeguatamente.

6.5.2 Secondo esempio

Nell'esempio in figura 6.2 si osserva come un algoritmo "greedy" con una visione limitata della topologia della rete prenda la strada sbagliata

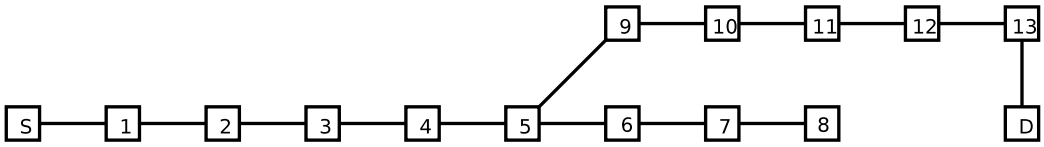

Figura 6.2: Esempio 2

per raggiungere la destinazione. Si può osservare inoltre come l'algoritmo proposto effettui un "backtracking" fino a ritrovare il percorso giusto.

Il pacchetto di ricerca route parte dal nodo S e, dopo qualche iterazione, raggiunge il nodo 5. Fra i due vicini a disposizione (9 e 6) quello scelto è il nodo 6 perché è quello meno distante dalla destinazione.

Ovviamente la strada intrapresa è quella sbagliata e l'algoritmo se ne accorge al nodo 8, dove nota che l'unico vicino a disposizione è il nodo 7 e fa parte del percorso già attraversato.

L'algoritmo marca il nodo 8 come "da evitare", e ritorna al nodo 7.

Qui, di nuovo si accorge che dei due vicini a disposizione uno è "da evitare" e l'altro fa parte del percorso già visitato. Anche il nodo 7 viene marcato "da evitare" e si passa al 6 (stessa sorte) e si arriva al nodo 5.

Ora però la situazione è diversa, in quanto l'unico nodo ammissibile è il nodo 9 e quindi finalmente viene intrapresa la strada giusta.

Una volta raggiunto il nodo 13 viene generato il pacchetto di risposta che viene inviato a ritroso verso il nodo S seguendo però il seguente percorso $13 \rightarrow 12 \rightarrow 11 \rightarrow 10 \rightarrow 9 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow S$.

6.5.3 Terzo esempio

Questo esempio (vedi figura 6.3 è quello più significativo poichè mostra la parte innovativa dell'algoritmo e cioè l'ottimizzazione del percorso trovato tramite le scorciatoie.

Come al solito l'algoritmo parte dal nodo S e si muove avvicinandosi alla destinazione fino al nodo 5. Tale nodo è un massimo locale e si trova infatti sul perimetro di un'area con assenza di nodi.

L'unica novità che negli esempi precedenti abbiamo tralasciato di menzionare è che il nodo di partenza viene aggiunto alla lista dei punti scorciatoia.

L'algoritmo quindi procede fino al nodo 10 dove si rende conto che finalmente può di nuovo avvicinarsi alla destinazione quindi il nodo 10 viene aggiunto alla lista dei "nodi scorciatoia".

Una volta raggiunto il nodo 14 viene preparato ed inviato a ritroso il pacchetto di risposta, e i nodi interessati aggiungono al loro forwarding table l'informazione per raggiungere la destinazione.

Quando il pacchetto di risposta raggiunge il nodo 10 questo si accorge di essere un nodo scorciatoia, ma è l'ultimo della lista quindi non effettua nessuna azione.

Quando invece il pacchetto di risposta arriva al nodo "S" quest'ultimo si accorge di essere un nodo scorciatoia e che nella lista esiste un altro nodo scorciatoia dopo di lui: quindi fa partire una nuova ricerca percorso verso il nodo 10 ma con destinazione reale nodo "D".

La ricerca di quest ultimo percorso avviene in maniera analoga agli esempi precedenti, ma con la sola differenza che questa volta i vari nodi interessati aggiungeranno nella loro forwarding table non solo che il percorso trovato serve per raggiungere il nodo 10, ma anche il nodo "D".

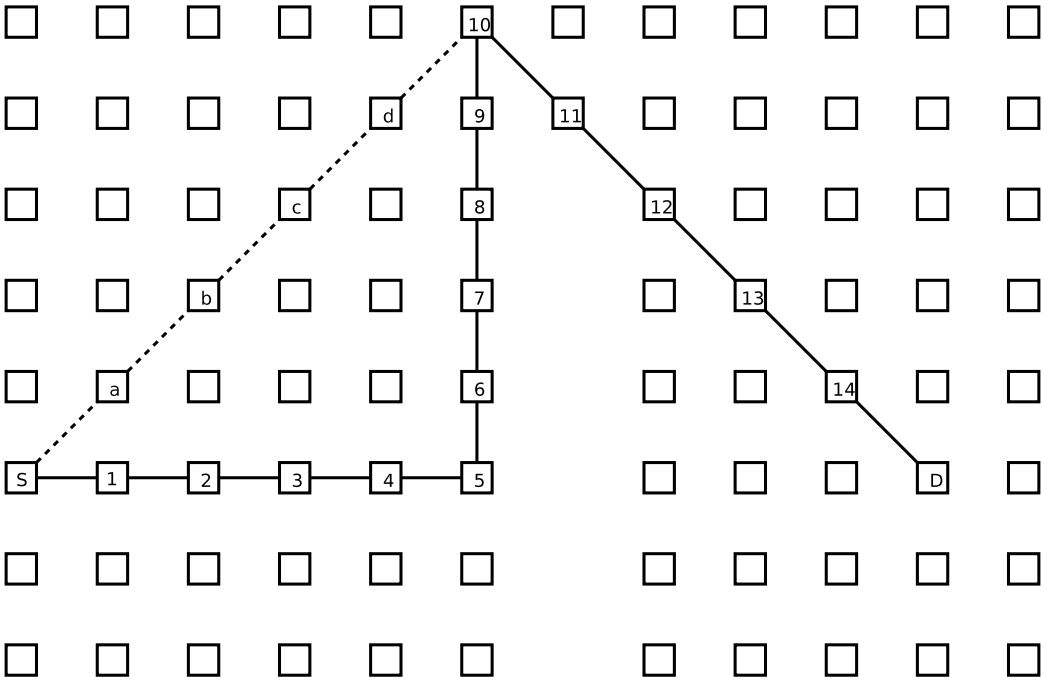

Figura 6.3: Esempio 3

Capitolo 7

IMPLEMENTAZIONE

In questo capitolo si parlerà dell'implementazione dell'algoritmo di routing e verranno spiegate le scelte tecniche effettuate. Tale implementazione permetterà di eseguire dei test e dei confronti con altri algoritmi come vedremo nel capitolo 8.

Ma lo scopo principale per cui è stato scritto il codice è quello di fornire ai partecipanti delle comunità wireless uno strumento funzionante, anche se con capacità basilari, per permettere loro di iniziare a fare delle prove, delle sperimentazioni, e come base per future modifiche e miglioramenti che permettano un utilizzo costante dell'algoritmo nelle reti rooftop.

E gran parte delle scelte tecniche presentate nei paragrafi successivi sono dettate dall'obiettivo appena descritto.

7.1 Scelte di base

Prima di iniziare la scrittura del codice sono state prese due decisioni importanti su linguaggio da utilizzare e sulla tipologia di interazione con il sistema operativo.

Si è deciso che il codice del router dovrà girare al di fuori del "kernel", e dovrà interagire con esso tramite le chiamate di sistema pubbliche come fanno per esempio i "demoni" nel sistema operativo UNIX.

Questo perché un'implementazione esterna presenta i vantaggi qui elencati:

- è adattabile più facilmente a vari tipi di sistemi operativi e di dispositivi hardware;
- il codice risulta in genere più comprensibile e quindi più facilmente (e velocemente) modificabile in caso di nuove esigenze;
- la fase di debug è più semplice in quanto non occorrono strumenti particolari per effettuare il debug all'interno del kernel;
- la stabilità del kernel non viene compromessa da bug contenuti nel codice del router.

Questi vantaggi sono simili a quelli che hanno fatto propendere la maggior parte dei ricercatori allo studio e alla realizzazione di "microkernel" invece che kernel monolitici.

Il linguaggio scelto per l'implementazione è il "Java"; tale decisione è stata presa in seguito alle seguenti considerazioni:

- una delle caratteristiche fondamentali di tale linguaggio è che i programmi possono girare senza modifiche su svariate architetture: non solo su computer ma anche su cellulari e "pda", e anche questi ultimi potrebbero essere utilizzati in futuro nelle reti rooftop;
- la lentezza dell'esecuzione del codice e il consumo di memoria più elevato rispetto a programmi compilati in codice nativo possono essere risolti mediante l'utilizzo del nuovo compilatore "gnu gcj" che genera direttamente codice nativo;
- per il linguaggio Java esistono numerosi strumenti di sviluppo sia proprietari che "open source": editor, debugger, profiler, cvs, UML designing;
- il linguaggio Java è ampiamente diffuso e conosciuto, quindi vi è una maggiore probabilità di ricevere contributi dai programmatori delle comunità wireless rooftop.

7.2 Architettura

La fase di design concettuale è stata svolta su carta. In seguito è stato disegnato il diagramma delle classi a livello implementativo mostrato in figura 7.1.

La classe "Dispatcher" si occupa di ricevere da varie fonti i comandi da eseguire che sono tutti sottoclassi della classe astratta Action e che quindi presentano un'interfaccia comune, come consiglia il "command pattern" [comb].

Dalla figura 7.1 si nota inoltre una forte interdipendenza fra le varie classi del progetto.

In più, come accennato in precedenza, il codice deve poter essere riusabile in tre situazioni:

1. in una rete rooftop reale;
2. collegato ad un simulatore di rete;
3. in un ambiente di test interattivo.

Per queste ragioni si è ritenuto di utilizzare i suggerimenti del pattern "Facade" [faca] [facb], e si è modificata l'architettura come si vede in figura 7.2.

La parte "decisionale" è stata disaccoppiata dalla parte che interagisce con il sistema operativo (o con il simulatore) dalla classe astratta System-Interaction.

In questo modo chi desidera effettuare il "porting" del codice in un nuovo sistema operativo deve solamente scrivere una sottoclasse di System-Interaction che implementi tutti i suoi metodi.

Durante la scrittura del codice sono stati effettuati dei controlli periodici sul rispetto dell'architettura originaria tramite uno strumento di generazione automatica dei diagrammi UML dal codice stesso.

In figura 7.3 si può vedere il diagramma delle classi in uno stato avanzato dei lavori.

Si può notare che il progetto originario è stato rispettato, ciò conferma la validità del progetto stesso.

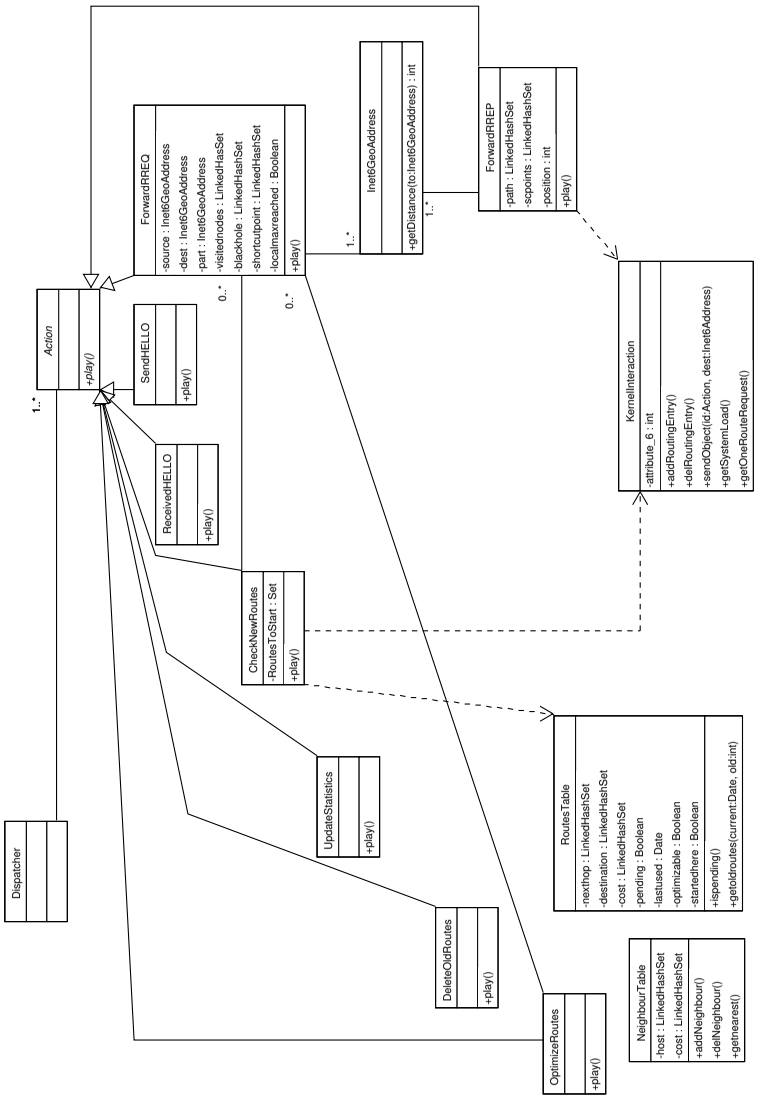

Figura 7.1: Diagramma classi iniziale (livello implementazione)

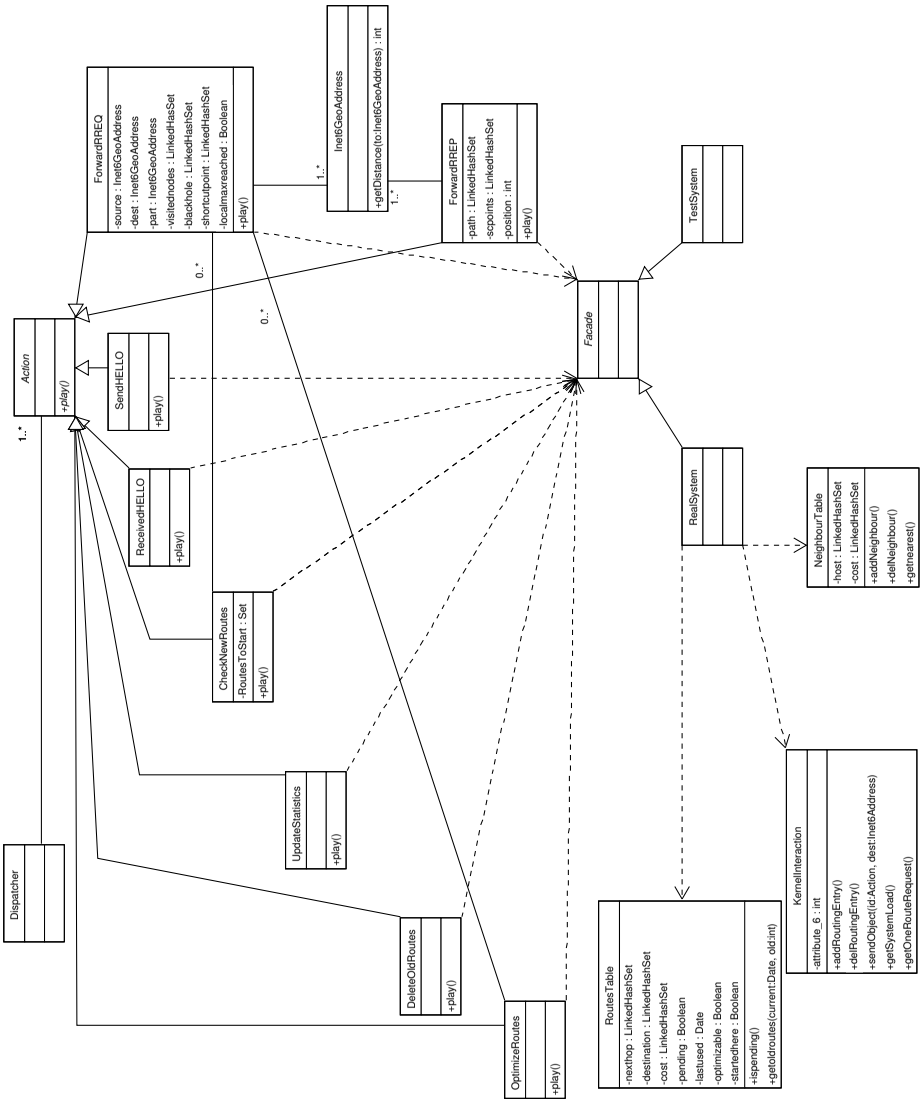

Figura 7.2: Diagramma classi dopo applicazione pattern “Facade”

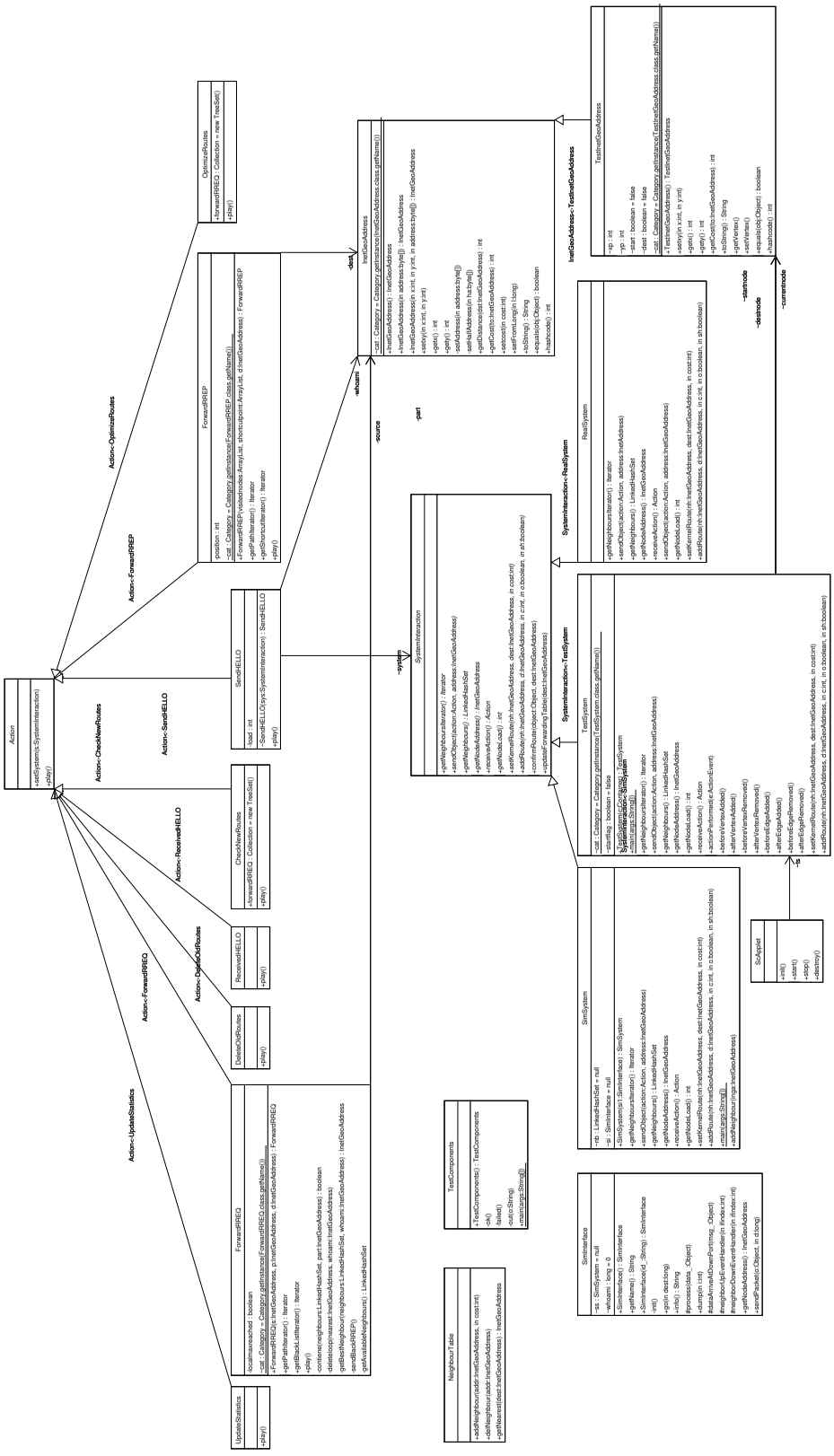

Figura 7.3: Diagramma classi in uno stadio avanzato dei lavori

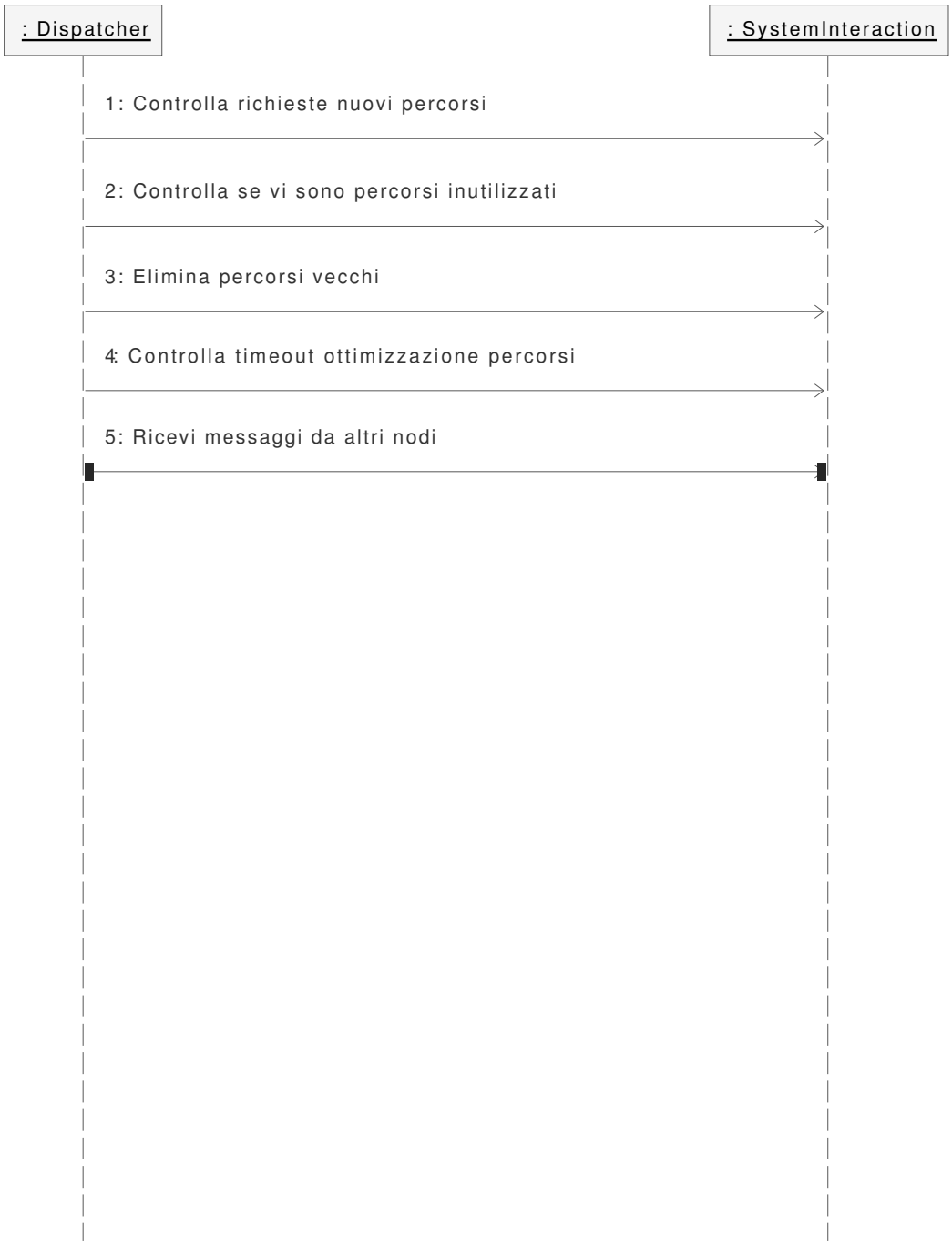

Figura 7.4: Sequenza fondamentale

7.3 Classi Principali

7.3.1 Dispatcher

La classe Dispatcher ha il compito di "direttore d'orchestra", quello cioè di ricevere gli eventi dal sistema (possibilmente già sotto forma di classi Action) e di eseguirli.

Gli eventi sono di vari tipi: richieste di route provenienti da altri nodi, richieste di un nuovo percorso dal sistema operativo del nodo, timeout per eventi periodici di manutenzione.

Visti i vari tipi di eventi possibili e poiché il linguaggio Java supporta nativamente i thread si potrebbe pensare ad una implementazione multi-threaded.

Ma, considerando che l'algoritmo è già complicato in quanto è di natura distribuita, si è seguito il consiglio del pattern "reactor" [Sch], che suggerisce di evitare l'utilizzo dei thread e invece appoggiarsi alla notifica asincrona degli eventi fornita dal sistema operativo.

Quindi la classe Dispatcher esegue in sequenza gli eventi come mostrato nella figura 7.4

7.3.2 ForwardRREQ e ForwardRREP

Le due classi ForwardRREQ e ForwardRREP sono le classi principali, in quanto implementano la fase di ricerca del percorso.

Sul loro funzionamento non c'è molto da aggiungere se non che, nella loro versione attuale, non tengono conto del costo dei nodi, ma solamente della lunghezza del percorso. Ma il codice è già stato predisposto per aggiungere valutazioni più sofisticate per controllare il costo del percorso.

7.3.3 SendHELLO e ReceiveHELLO

Le due classi SendHELLO e ReceiveHELLO servono per mantenere aggiornata la lista dei vicini di un nodo in mancanza di altri metodi (li-

sta statica di vicini inserita in un file di configurazione, oppure notifica automatica dal livello inferiore).

7.3.4 UpdateStatistics

La classe UpdateStatistics si occupa di mantenere aggiornate le statistiche riguardo all'uso di ogni percorso (in modo che quelli non più usati possano essere eliminati) e al carico del nodo (per la versione dell'algoritmo che tiene conto del costo totale del percorso e non solamente dei nodi attraversati).

7.3.5 OptimizeRoutes

La classe OptimiseRoutes effettua periodicamente un nuovo tentativo di ricerca route per i percorsi già esistenti, allo scopo di controllare se ne esistono di migliori.

La frequenza della ricerca è maggiore per i percorsi che contengono scorciatoie.

7.3.6 DeleteOldRoutes

La classe DeleteOldRoutes si occupa di cancellare sia i percorsi non più utilizzati (allo scopo di contenere le dimensioni della tabella) che i percorsi in fase di ricerca per cui non si è ottenuta una risposta entro un tempo limite (così possono essere richiesti nuovamente).

7.3.7 CheckNewRoutes

La classe CheckNewRoutes si occupa di ricevere dal sistema operativo la lista di richieste per nuove destinazioni. Da tale lista elimina poi quelle per cui la richiesta è già stata effettuata ed è in attesa di risposta; con le rimanenti fa partire la richiesta di nuovi percorsi.

7.4 Note implementative

La versione attuale del software è stata interfacciata con il sistema operativo Linux. A tale proposito si desidera mettere in evidenza due casi dove sono state trovate difficoltà non previste nella scrittura del codice.

La prima difficoltà consiste nel fatto che non esiste in Linux una chiamata di sistema per sapere se c'è bisogno di una nuova route.

Infatti se il kernel riceve un pacchetto per una destinazione non presente nel forwarding table semplicemente lo scarta e invia un pacchetto di errore alla sorgente.

Alcuni esperti di kernel hanno suggerito che l'unica soluzione sia o la modifica del kernel o l'implementazione di un modulo apposito: entrambe le soluzioni sono state considerate eccessivamente impegnative e contrarie ai propositi del paragrafo 7.1.

É stata invece utilizzata la possibilità offerta dal kernel di creare dispositivi di rete virtuali. Tali dispositivi, pur essendo configurabili esattamente come un dispositivo ethernet reale (quindi hanno indirizzo IP, netmask ecc.), appaiono invece alle applicazioni come dispositivi a carattere. Quindi ogni pacchetto IP inviato a tali dispositivi può essere ricevuto un byte alla volta dall'applicazione che ha creato l'interfaccia virtuale.

Si è quindi impostato il route di default su questo dispositivo virtuale. Quindi se per un pacchetto esiste già un route speficico questo viene inviato attraverso esso. Altrimenti viene inviato al route di default: il software di routing ne estrae l'indirizzo IP e inizia la ricerca di un nuovo percorso.

La seconda difficoltà invece si presenta quando occorre sapere se un percorso è ancora utilizzato o meno (e quindi eliminarlo dalla relativa tabella). Questa volta invece si è pensato di utilizzare le funzionalità di accounting del traffico del kernel Linux.

Tali servizi mantengono il conto dei pacchetti che attraversano una determinata interfaccia di rete allo scopo di implementare politiche di "pricing". A questo punto basta controllare il numero di byte scaricati: se rimane costante per un sufficiente intervallo di tempo significa che il percorso non è più utilizzato e può essere rimosso.

7.5 Collaborazione

Una volta scritte le parti principali del router si è provveduto a pubblicarlo in un famoso sito per lo sviluppo pubblico del software [sou].

Tale sito offre una serie di servizi per facilitare il lavoro di gruppo dei programmatori: repository CVS, mailing list, forum, bug reporting, e altri.

Si è aspettato a pubblicare il progetto allo scopo di avere una parte di codice funzionante e un "applet" dimostrativa in linguaggio Java in quanto, dopo un'analisi attenta dell'andamento dei progetti presenti su quel sito, è sembrata la scelta migliore.

Infatti si è visto che:

- la presenza di una buona parte del codice conferisce al progetto un'idea di "serietà";
- molti programmatori non si interessano alle scelte architetturali e/o algoritmiche, ma iniziano a contribuire solamente quando è ora di scrivere codice;
- i progetti aperti "al pubblico" troppo presto generano numerose discussioni all'interno dei loro forum riguardo alle scelte architetturali, che non portano poi a nessuna conclusione.

Capitolo 8

SIMULAZIONE

L'algoritmo oggetto di questa tesi è il primo ideato specificatamente per le reti wireless di tipo rooftop, quindi non può essere confrontato con nessun altro in questa categoria, semplicemente perché non ve ne sono.

Si è scelto quindi di confrontarlo con l'algoritmo AODV, anche se pensato per reti wireless di tipo manet in quanto:

- è stato proposto dall'IETF ed è vicino alla standardizzazione;
- ne esistono già diverse implementazioni per Linux e anche varie simulazioni;
- varie comunità rooftop lo usano per i motivi appena citati, in attesa di un algoritmo specifico.

È stata effettuata una serie basilare di test per confrontare la scalabilità dei due algoritmi in attesa di riscontri dalle comunità rooftop su test svolti in casi reali.

8.1 Piattaforma

Per la simulazione dell'algoritmo proposto in questa tesi è stato usato il simulatore JavaSim [jav].

Tale simulatore è stato esteso con due moduli di routing che implementano i due algoritmi confrontati.

Per l'algoritmo oggetto di questa tesi è stata utilizzata l'implementazione descritta nel capitolo 7.

La modifica dell'implementazione per supportare il collegamento con il simulatore è stata semplice, grazie alle scelte progettuali spiegate nel paragrafo 7.2.

È stato necessario realizzare una classe "SimInterface" per estendere il simulatore, e una sottoclasse di "SystemInteraction" che implementa i metodi necessari all'algoritmo per interagire con il sistema (in questo caso col sistema simulato).

Per l'algoritmo AODV è stata utilizzata l'implementazione Java fornita da [Cha02]. Sfortunatamente tale implementazione non è stata pensata per essere adattata a varie situazioni, quindi il processo di modifica per l'utilizzo con Javasim è stato lungo e laborioso.

Per generare la topologia è stato utilizzato il generatore BRITE [MLMB01]; tale software è stato poi modificato per inserire la posizione geografica dei nodi nel loro indirizzo ip, come richiesto dall'algoritmo oggetto di questa tesi.

8.2 Test e Risultati

Sono stati scelti i test per mettere in evidenza le qualità fondamentali di un algoritmo di routing per reti rooftop:

- l'aumento del carico imposto alla rete per lo scambio dei messaggi necessari all'algoritmo all'aumentare del numero di nodi della rete stessa;

- la qualità del percorso trovato in termini di numero di nodi da attraversare.

Riguardo alla qualità del percorso si è pensato di confrontare due versioni dell'algoritmo proposto, una completa, l'altra mancante della fase di ottimizzazione, per vedere quali miglioramenti sono apportati dalla ricerca degli "shortcut".

Con BRITE sono stati generati quattro scenari di dimensioni crescenti.

Per generare gli scenari è stato scelto in BRITE il modello "Barabasi-Albert" incrementale, che è quello ritenuto migliore per rappresentare una rete rooftop in quanto simula la crescita nel tempo di una rete aggiungendo nodi nelle vicinanze di quelli già esistenti.

In ogni scenario è stata fatta partire la ricerca di cinque percorsi in contemporanea.

Al termine della ricerca dei percorsi è stato osservato il numero di byte scambiati fra i nodi per l'attuazione dell'algoritmo di routing.

In seguito è stato inviato un pacchetto dati di prova da ogni sorgente verso la rispettiva destinazione per verificare se i percorsi effettivamente erano corretti e per misurare la lunghezza di ogni percorso.

I risultati si possono osservare nelle figure 8.1 e 8.2.

Nel primo grafico, che mostra l'aumento del traffico richiesto dall'algoritmo di routing all'aumentare del numero di nodi della rete, si può immediatamente vedere che l'overhead dell'algoritmo AODV è molto maggiore di quello del suo concorrente.

Infatti per generare un percorso AODV invia un pacchetto a tutti i nodi della rete quindi la quantità di traffico generata segue una regola di $O(n)$ con n il numero di nodi della rete.

L'algoritmo Shortcut invece genera un numero di pacchetti approssimativamente proporzionale alla lunghezza del percorso e quindi indipendente dal numero di nodi della rete. Come infatti si può vedere nel grafico l'overhead per l'algoritmo oggetto di questa tesi nel caso di cento nodi è addirittura più basso che nel caso di cinquanta: questo accade perché la lunghezza media dei percorsi, nel caso della simulazione con cento nodi, è risultata più bassa.

Ovviamente l'overhead in seguito cresce perché all'aumentare dei nodi della rete è probabile che aumenti anche la lunghezza del percorso medio.

Nel grafico si nota inoltre che la versione dell'algoritmo Shortcut senza la ricerca di scorciatoie ha prestazioni lievemente migliori in quanto appunto non effettua le ulteriori ricerche di percorso necessarie per l'ottimizzazione.

Nel grafico in figura 8.2 si può vedere invece la capacità di ogni algoritmo di trovare il percorso più breve possibile.

Com'era prevedibile, in situazione di nodi statici e senza traffico di rete, l'algoritmo AODV trova percorsi mediamente più brevi (e corrispondenti al massimo teorico).

Nel grafico si vede inoltre che l'algoritmo Shortcut col solo backtracking non trova percorsi di buona qualità.

Invece la versione che utilizza anche la tecnica degli shortcut riesce ad ottenere percorsi quasi ottimali, al prezzo di un leggero (e costante) aumento dell'overhead del routing, che era proprio l'obiettivo che ci si era preposti in questa tesi: riuscire ad ottenere buoni percorsi anche in reti di medie dimensioni.

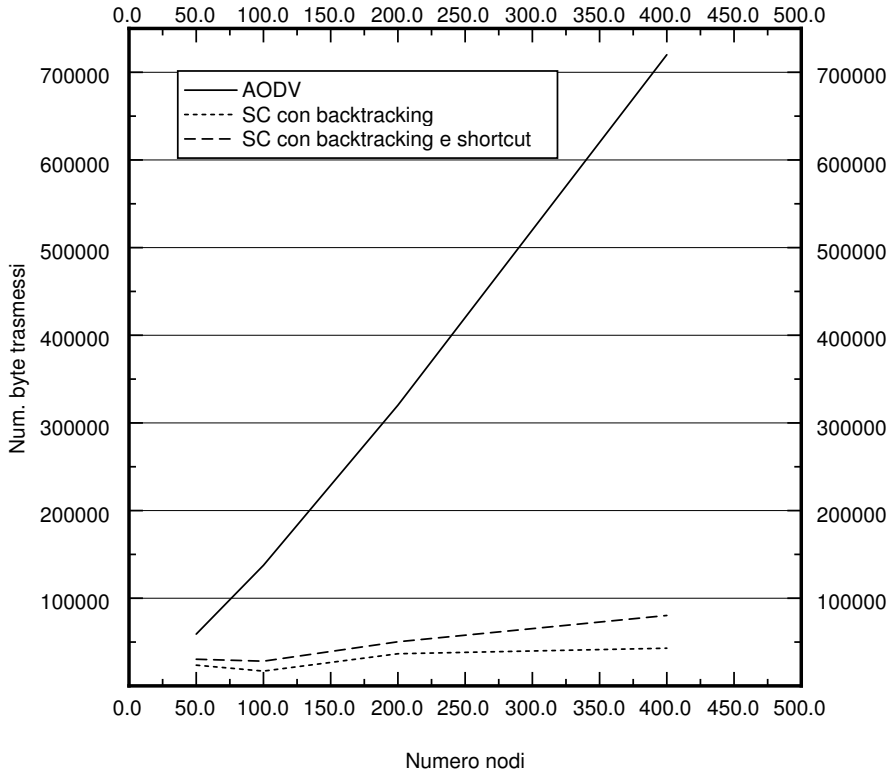

Figura 8.1: Scalabilità dell'algoritmo

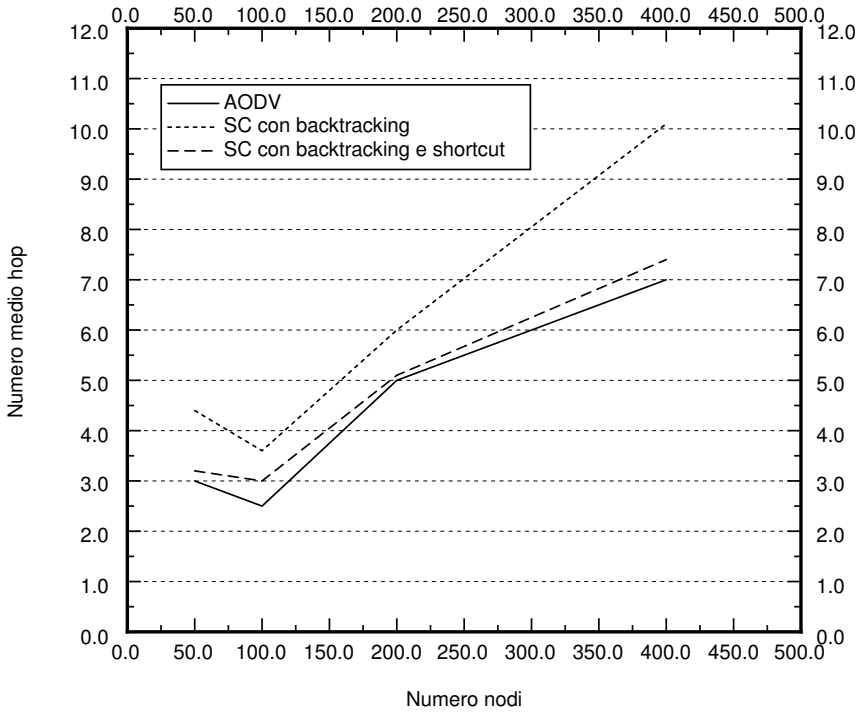

Figura 8.2: Lunghezza media percorsi

Capitolo 9

CONCLUSIONI E SVILUPPI FUTURI

Ragioni economiche e sociali hanno portato alla diffusione di reti alternative ad Internet utilizzanti collegamenti wireless.

Queste reti, denominate "rooftop" e caratterizzate dall'avere nodi fissi necessitano di una soluzione specifica per il problema del routing e per l'assegnazione di indirizzi univoci, in quanto le soluzioni già esistenti per Internet e per le reti di tipo MANET non sono utilizzabili.

Nel corso di questa tesi è stato presentato quindi un algoritmo di routing specifico per reti rooftop con annessa una soluzione per l'assegnazione univoca degli indirizzi IP.

L'algoritmo utilizza un approccio geografico nella ricerca del percorso, con in più una fase di backtracking per aggirare le zone con assenza di nodi.

La caratteristica innovativa che lo distingue da algoritmi simili per reti MANET consiste in una fase aggiuntiva di ottimizzazione del percorso trovato tramite l'uso di "shortcut".

Dell'algoritmo è stata realizzata una versione basilare di riferimento in linguaggio Java; tale versione è stata poi pubblicata su Internet per ricevere suggerimenti e contributi dalle comunità wireless.

La stessa versione è stata poi usata in associazione con il simulatore

Javasim per ottenere una serie di risultati, che dimostrano chiaramente che:

- l'overhead necessario all'algoritmo non cresce all'aumentare del numero di nodi della rete;
- i percorsi trovati sono mediamente vicini come lunghezza a quelli migliori possibili.

Gli sviluppi futuri possibili sono vari e riguardano sia l'implementazione che l'algoritmo stesso.

Innanzitutto è opportuno in collaborazione con i partecipanti delle varie comunità wireless, realizzare test approfonditi dell'algoritmo in situazioni reali allo scopo di:

- trovare i valori migliori degli intervalli per la ricerca periodica di percorsi migliori di quelli già esistenti;
- scegliere la soluzione migliore in caso di interruzione del percorso fra le due già viste: riparazione locale oppure iniziare una nuova ricerca;
- controllare se la tecnica per migliorare la latenza iniziale inviando subito i pacchetti dati assieme al pacchetto ricerca percorso non sovraccarichi troppo la rete;
- verificare se la soluzione di considerare i nodi sovraccarichi come non funzionanti (in maniera quindi che l'algoritmo li eviti) serva effettivamente per distribuire il carico sulla rete e non crei invece percorsi troppo lunghi.

Nel corso del lavoro di tesi non è stata offerta nessuna soluzione per gestire l'interazione fra una rete rooftop ed Internet: è necessario quindi studiare il problema solo apparentemente banale di trovare un route da un nodo di una rete rooftop e un nodo collegato ad Internet e viceversa.

È interessante anche studiare una possibile evoluzione dell'algoritmo in chiave gerarchica analogamente a quanto effettuato dall'algoritmo Terminode.

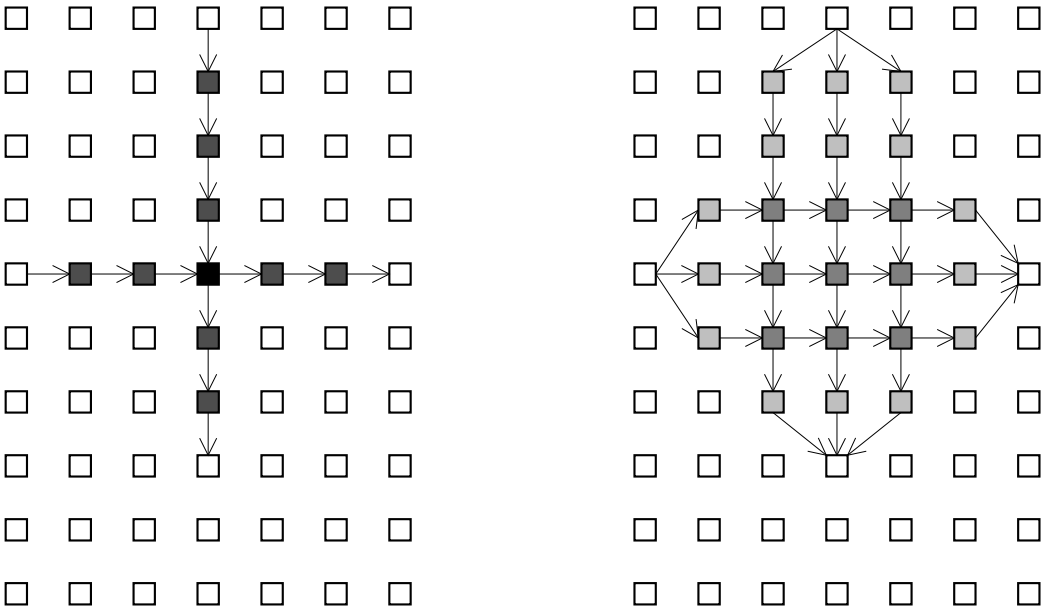

Figura 9.1: Distribuzione del carico (il nodo in nero è sovraccarico)

La fase di ricerca percorso servirebbe in questo caso solamente a dare ai singoli nodi delle indicazioni in linea di massima; ai singoli nodi spetterebbe la decisione finale, presa in base alle informazioni locali di tipo "link state" da loro possedute.

Il vantaggio di questa soluzione è che i singoli nodi potrebbero effettuare un bilanciamento del carico; come si vede in figura 9.1 è meglio per esempio avere tre percorsi paralleli a basso traffico che un singolo percorso saturo.

Per concludere si può notare che i problemi fondamentali da risolvere nelle reti rooftop siano trovare un modo efficiente per evitare le zone vuote e trovare un indirizzamento gerarchico efficiente: ed è in questa direzione che si intende procedere nel lungo termine.

Bibliografia

[Alo] Mohamed-Slim Alouini. Global positioning system: An overview.

[BBG02] Ljubica Blazevic, Jean-Yves Le Boudec, and Silvia Giordano. A scalable routing method for irregular mobile ad hoc networks, 2002.

[BMJ+98] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Mobile Computing and Networking*, pages 85–97, 1998.

[BP00] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *INFOCOM (2)*, pages 775–784, 2000.

[Cha02] Amitabh Chaudhury. Experimental wireless networks. Master's thesis, Rensselaer Polytechnic Institute, 2002.

[CHH01] Srdan Capkun, Maher Hamdi, and Jean-Pierre Hubaux. GPS-free positioning in mobile ad-hoc networks. In *HICSS*, 2001.

[CM99] S. Corson and J. Macker. RFC 2501: Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations, January 1999.

[coma] `http://wirelessanarchy.com/`.

[comb] http://www.javaworld.com/javaworld/javatips/jw-javatip68.html.

[CWLG97] C. Chiang, H. Wu, W. Liu, and M. Gerla. Routing in clustered multihop, mobile wireless networks, 1997.

[faca] http://www.wikipedia.org/w/wiki.phtml?search=facade+pattern&go=Go.

[facb] http://www.cs.ucsb.edu/~cappello/50/lectures/patterns/Facade.html.

[Hai02] T. Hain. An ipv6 provider-independent global unicast address format, 2002.

[Hed88] C. L. Hedrick. RFC 1058: Routing information protocol, June 1988. Updated by RFC1388, RFC1723 [Mal93, Mal94]. Status: HISTORIC.

[jav] http://www.javasim.org/.

[JM96] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[KK00] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Mobile Computing and Networking*, pages 243–254, 2000.

[KV98] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Mobile Computing and Networking*, pages 66–75, 1998.

[KZ89] Atul Khanna and John Zinky. The revised ARPANET routing metric. In *Proc. ACM SIGCOMM '89*, pages 45–56, Austin, TX, September 1989.

[Leo02] David Leonard. A provider-independent spatial unicast addressing scheme for ipv6, 2002.

[LJD+00] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 120–130, August 2000.

[Mal93] G. Malkin. RFC 1388: RIP version 2 carrying additional information, January 1993. Obsoleted by RFC1723, RFC2453 [Mal94, Mal98]. Updates RFC1058 [Hed88]. Updated by RFC2453 [Mal98], STD0056. Status: PROPOSED STANDARD.

[Mal94] G. Malkin. RFC 1723: RIP version 2 — carrying additional information, November 1994. Obsoletes RFC1388 [Mal93]. Obsoleted by RFC2453 [Mal98]. Updates RFC1058 [Hed88]. Updated by RFC2453 [Mal98], STD0056 . Status: DRAFT STANDARD.

[Mal98] G. Malkin. RFC 2453: RIP version 2, November 1998. Obsoletes RFC1388, RFC1723 [Mal93, Mal94]. Status: STANDARD.

[MLMB01] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: Universal topology generation from a user's perspective. Technical Report 2001-003, 1 2001.

[Moy89] J. Moy. RFC 1131: OSPF specification, October 1989. Obsoleted by RFC1247 [Moy91]. Status: PROPOSED STANDARD.

[Moy91] J. Moy. RFC 1247: OSPF version 2, July 1991. See also RFC1246, RFC1245 . Obsoleted by RFC1583 [Moy94]. Obsoletes RFC1131 [Moy89]. Status: DRAFT STANDARD.

[Moy94] J. Moy. RFC 1583: OSPF version 2, March 1994. Obsoleted by RFC2178 [Moy97]. Obsoletes RFC1247 [Moy91]. Status: DRAFT STANDARD.

[Moy97] J. Moy. RFC 2178: OSPF version 2, July 1997. Obsoleted by RFC2328 [Moy98]. Obsoletes RFC1583 [Moy94]. Status: DRAFT STANDARD.

[Moy98] J. Moy. RFC 2328: OSPF version 2, April 1998. See also STD0054. Obsoletes RFC2178 [Moy97]. Status: STANDARD.

[NNS96] T. Narten, E. Nordmark, and W. Simpson. RFC 1970: Neighbor discovery for IP version 6 (IPv6), August 1996. Obsoleted by RFC2461 [NNS98]. Status: PROPOSED STANDARD.

[NNS98] T. Narten, E. Nordmark, and W. Simpson. RFC 2461: Neighbor discovery for IP Version 6 (IPv6), December 1998. Obsoletes RFC1970 [NNS96]. Status: DRAFT STANDARD.

[PC97] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *INFOCOM (3)*, pages 1405–1413, 1997.

[Per97] C. Perkins. Ad hoc on demand distance vector (aodv) routing, 1997.

[PJ96] Charles E. Perkins and David B. Johnson. Mobility support in ipv6. In *Mobile Computing and Networking*, pages 27–37, 1996.

[RS98] Ram Ramanathan and Martha Steenstrup. Hierarchically-organized, multihop mobile wireless networks for quality-of-service support. *Mobile Networks and Applications*, 3(1):101–119, 1998.

[Sch] Douglas C. Schmidt. Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching. pages 529–547.

[sou] `http://scrouter.sourceforge.net/`.

[Sun] Allen C. Sun. Design and implementation of fisheye routing protocol for mobile ad hoc networks.

[TN96] S. Thomson and T. Narten. RFC 1971: IPv6 stateless address autoconfiguration, August 1996. Obsoleted by RFC2462 [TN98]. Status: PROPOSED STANDARD.

[TN98] S. Thomson and T. Narten. RFC 2462: IPv6 stateless address autoconfiguration, December 1998. Obsoletes RFC1971 [TN96]. Status: DRAFT STANDARD.

[wik] `http://www.wikipedia.org/wiki/Ad_hoc_protocol_list`.