POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Corso di laurea in Ingegneria Informatica

COMPARAZIONE DI METODI PER IL BROADCAST NELLE WSN IN SIMULAZIONE E SCENARI REALI

Relatore: Prof. Gianpaolo CUGOLA

Tesi di laurea di:
Alberto MARINELLI
Matr. 720951

Anno Accademico 2009 - 2010

Desidero ringraziare per primi i miei genitori,
che mi hanno cresciuto, educato, stimolato
e infine dato i mezzi per raggiungere questo importante traguardo.
Questa tesi di laurea è anche loro.

Sono riconoscente anche ai parenti e agli amici,
che con il loro incoraggiamento mi hanno sostenuto
durante quest'ultimo passo della mia carriera universitaria.

Un ringraziamento particolare va a Gianpaolo Cugola,
che mi ha guidato sapientemente in questo lavoro
lasciandomi libero di esprimere le mie potenzialità.

Infine rivolgo un pensiero grato ad Alberto Leva,
che con la sua esperienza e disponibilità
molto mi ha insegnato.

Indice

Elenco delle figure **v**

Elenco delle tabelle **vii**

Sommario **viii**

1 Introduzione **1**

1.1 Obiettivi del lavoro 3

1.2 Struttura del documento 4

2 Il broadcast nelle reti wireless **6**

2.1 Metodi basati sulle probabilità 10

2.1.1 Gossip 10

2.1.2 Gossip basato sul numero dei vicini 10

2.1.3 Gossip basato sul numero di repliche ricevute 11

2.1.4 Smart Gossip 11

2.2 Metodi basati sull'area aggiuntiva coperta 12

2.2.1 Metodo basato sul contatore 12

2.2.2 Metodo basato sul contatore (adattivo) 13

2.2.3 Metodo basato sulla distanza 13

2.2.4 Opportunistic Flooder 14

2.2.5 Metodo basato sulla distanza e sul numero di hop . . . 14

2.2.6 Joint Distance-Counter Threshold Broadcast Algorithm (JDCT) 15

2.2.7 Metodo basato sulla posizione 15

2.2.8 Metodo basato sulla posizione (adattivo) 16
2.3 Metodi basati sulla conoscenza dei vicini 16
2.3.1 Irrigator protocol 17
2.3.2 Fireworks protocol 18
2.3.3 Robust Broadcast Algorithm (RBP) 18
2.3.4 Flooding based on One-hop Neighbor Information and Adaptive Holding (FONIAH) 19
2.3.5 Self pruning 19
2.3.6 Delayed Flooding with Cumulative Neighborhood (DFCN) 20
2.3.7 Metodo basato sulla copertura dei vicini 20
2.3.8 Dominant pruning 21
2.3.9 Efficient Flooding Scheme Based on 1-hop Information 22
2.3.10 Scalable Broadcast Algorithm (SBA) 22
2.3.11 Scalable Broadcast Algorithm adattivo (SBA adattivo) 23
2.3.12 Multipoint relaying 23
2.3.13 Ad Hoc Broadcast Protocol (AHBP) 24
2.3.14 Ad Hoc Broadcast Protocol adattivo (AHBP adattivo) 25
2.3.15 Metodo basato sui Connected Dominating Sets (CDS) 25
2.3.16 Metodo basato sui nodi interni 26
2.3.17 Lightweight and Efficient Network-Wide Broadcast (LENWB) 26
2.3.18 Ring Algorithm 27
2.4 Metodi basati sui cluster 27
2.4.1 Metodo 1 basato sui cluster (SI-CDS) 28
2.4.2 Metodo 2 basato sui cluster (backbone SI-CDS) 29
2.4.3 Metodo 3 basato sui cluster (backbone SD-CDS) 29
2.5 Metodi basati su alberi 30
2.5.1 TreeCast 30
2.5.2 Metodo collision-free basato su alberi 30
2.6 Altri metodi 31
2.6.1 Pure machine learning 31
2.6.2 Inter-protocol learning 32

3 I metodi scelti **33**
3.1 OppFlooder 39
3.2 DistFlooder 40
3.3 JDCTFlooder 42
3.4 SBAFlooder 44
3.5 AHBPFlooder 46
3.6 Considerazioni riguardo i metodi scelti 50
3.6.1 Le due categorie a confronto 50
3.6.2 DistFlooder, OppFlooder e JDCTFlooder a confronto . 52
3.6.3 SBAFlooder e AHBPFlooder a confronto 53

4 Ricerche di minimo teorico **55**
4.1 La ricerca del MCDS 55
4.2 L'algoritmo di ricerca proposto: MinBroadcast 58
4.2.1 Ottimizzazione locale: eliminazione dei nodi dominati . 60
4.2.2 Ottimizzazione globale: eliminazione dei sottoalberi ridondanti 61
4.2.3 Descrizione dell'algoritmo 62
4.2.4 Implementazione dell'algoritmo 69

5 Esperimenti con il simulatore **71**
5.1 Il simulatore utilizzato: OMNeT++ 71
5.2 Obiettivi e scenari delle simulazioni 73
5.3 Parametri generali delle simulazioni 75
5.4 Scelta dei parametri dei metodi 77
5.4.1 OppFlooder 78
5.4.2 DistFlooder 79
5.4.3 JDCTFlooder 81
5.4.4 SBAFlooder 84
5.4.5 AHBPFlooder 86

6 Esperimenti in scenari reali **90**
6.1 Le reti sperimentali utilizzate: Motelab e Indriya 91
6.2 Le prove di connettività 93

6.3 Parametri delle simulazioni per il confronto con i casi reali . . 95
6.4 Obiettivi e scenari degli esperimenti 97
6.5 Parametri generali degli esperimenti e parametri dei metodi . 98

7 Risultati **101**
7.1 I risultati delle ricerche di minimo teorico 103
7.2 I risultati delle simulazioni: comparazione dei metodi scelti . . 106
7.2.1 I risultati negli scenari con nodi fissi 107
7.2.2 I risultati negli scenari con nodi mobili 115
7.2.3 L'impatto della mobilità sui metodi scelti 122
7.2.4 Considerazioni sulla comparazione dei metodi scelti . . 124
7.3 I risultati degli esperimenti reali: comparazione con le simulazioni 125

8 Conclusioni **133**

Bibliografia **I**

Elenco delle figure

4.1 Una rete d'esempio, il grafo e i due MCDS 57

5.1 L'impatto di T_{max} in OppFlooder 79
5.2 L'impatto di T_{max} in DistFlooder 80
5.3 L'impatto di $RSSI_{max}$ in DistFlooder 81
5.4 L'impatto di T_{max} in JDCTFlooder 82
5.5 L'impatto di $RSSI_{max}$ in JDCTFlooder 83
5.6 L'impatto di $RSSI_{max}$ e C in JDCTFlooder 83
5.7 L'impatto di T_{max} in SBAFlooder 84
5.8 L'impatto di T_c in SBAFlooder 85
5.9 L'impatto di T_c e T_t in SBAFlooder 86
5.10 L'impatto del parametro aggiuntivo $RSSI_c$ in AHBPFlooder (nodi fissi) 87
5.11 L'impatto del parametro aggiuntivo $RSSI_c$ in AHBPFlooder (nodi mobili) 87
5.12 L'impatto di T_{max} in AHBPFlooder 88
5.13 L'impatto di T_c in AHBPFlooder 89
5.14 L'impatto di T_c e T_t in AHBPFlooder 89

6.1 Risultati di connettività su Motelab e Indriya 95
6.2 Risultati di connettività su simulatore, Motelab e Indriya . . . 97

7.1 Confronto tra metodi e ricerche di minimo teorico 105
7.2 Risultati simulazioni - scenario base (nodi fissi) 108
7.3 Risultati simulazioni - densità variabile (nodi fissi) 110

7.4 Risultati simulazioni - area variabile (nodi fissi) 112
7.5 Risultati simulazioni - frequenza invio variabile (nodi fissi) . . 114
7.6 Risultati simulazioni - scenario base (nodi mobili) 116
7.7 Risultati simulazioni - densità variabile (nodi mobili) 118
7.8 Risultati simulazioni - area variabile (nodi mobili) 119
7.9 Risultati simulazioni - frequenza invio variabile (nodi mobili) . 121
7.10 Risultati simulazioni - velocità nodi variabile 123
7.11 Comparazione simulatore/esperimenti reali - densità variabile (OppFlooder, JDCTFlooder e SBAFlooder) 127
7.12 Comparazione simulatore/esperimenti reali - frequenza invio variabile (OppFlooder, JDCTFlooder e SBAFlooder) 129
7.13 Comparazione simulatore/esperimenti reali - densità variabile (risultati percentuali rispetto a OppFlooder) 130
7.14 Comparazione simulatore/esperimenti reali - frequenza invio variabile (risultati percentuali rispetto a OppFlooder) 131

Elenco delle tabelle

2.1 I metodi presentati 9

5.1 Distanze e RSSI nel simulatore 72
5.2 I parametri su cui sono basati gli scenari di simulazione 75
5.3 I parametri comuni a tutte le simulazioni 76
5.4 I parametri dei metodi nelle simulazioni 77

6.1 Le specifiche tecniche dei nodi TelosB 92
6.2 Le potenze radio scelte con le prove di connettività 94
6.3 Densità e area (per reti simulate) corrispondenti alle potenze radio scelte 96
6.4 I parametri su cui sono basati gli scenari degli esperimenti reali 98
6.5 I parametri comuni a tutti gli esperimenti 98
6.6 I parametri dei metodi negli esperimenti reali 99

7.1 Risultati delle ricerche di minimo teorico 104

Sommario

La presente tesi di laurea riguarda il broadcast in una particolare tipologia di reti wireless, chiamata Wireless Sensor Network (WSN). L'attenzione è focalizzata su queste reti sia in assenza sia in presenza di mobilità. In seguito a un'ampia trattazione dello stato dell'arte, vengono selezionati e dettagliatamente descritti alcuni metodi per il broadcast, uno dei quali, chiamato Opportunistic Flooder, è da noi proposto e appositamente pensato per operare nelle WSN mobili. I metodi scelti sono stati comparati per mezzo di un'estesa campagna di simulazioni. Alcuni di questi metodi sono anche stati confrontati su reti sperimentali. Inoltre l'efficienza dei metodi selezionati è stata valutata utilizzando i risultati teorici ricavati con un algoritmo, presentato in questo lavoro, capace di ottenere il numero minimo di trasmissioni necessarie a coprire una rete.

I risultati ottenuti hanno dimostrato che Opportunistic Flooder, in tutte le situazioni considerate, è un valido metodo per eseguire il broadcast nelle reti WSN ed è più efficiente delle tecniche con cui è stato comparato. Il confronto tra le simulazioni e gli esperimenti reali ha confermato che l'ambiente di simulazione utilizzato offre risultati aderenti a quelli ottenuti nelle reti reali. Infine i risultati di minimo teorico hanno messo in evidenza che c'è ancora spazio per qualche miglioramento.

Capitolo 1

Introduzione

La diffusione delle reti wireless è stata caratterizzata da una notevole crescita negli ultimi anni e, al giorno d'oggi, gli ambiti in cui queste reti vengono utilizzate sono numerosi e spesso molto diversi tra loro. Il presente lavoro prende in esame una particolare tipologia di rete wireless, chiamata *Wireless Sensor Network* (WSN).

Una WSN è una rete formata da un certo numero di dispositivi — in seguito chiamati semplicemente nodi — che sono connessi tra loro mediante una connessione wireless ad hoc, ovvero senza che sia presente un'infrastruttura predefinita. Le caratteristiche principali di una WSN sono le seguenti: modeste prestazioni computazionali dei nodi, limitata banda di trasmissione e ridotte disponibilità energetiche. Lo scopo principale delle WSN è raccogliere un qualche genere di informazione dall'ambiente circostante e trasmettere tale informazione ad altri componenti della rete che possono esserne interessati, oppure usare l'informazione per prendere determinate decisioni o inviare particolari comunicazioni, come per esempio un allarme. Il vantaggio principale delle WSN è non richiedere un lungo e costoso processo di installazione e cablaggio; inoltre, l'introduzione della connettività senza fili predispone implicitamente le WSN a operare in contesti mobili.

Le WSN mobili costituiscono un caso particolare delle reti di sensori wireless e presentano un'ulteriore peculiarità, ovvero la dinamicità della topologia della rete. È facile intuire che questo nuovo aspetto sia piuttosto rilevante e

debba essere tenuto in seria considerazione nello studio e nella realizzazione di una WSN da utilizzare in condizioni di mobilità.

Grazie ai progressi tecnologici, in particolare per quanto riguarda la miniaturizzazione dei circuiti elettronici e la riduzione dei consumi energetici, sia dei microcontrollori sia delle radio, l'uso delle WSN è stato introdotto in molti campi applicativi, quali quello militare, medico-sanitario e ambientale. In generale, le WSN hanno assunto una notevole importanza nell'ambito del monitoraggio di cose, animali e anche persone, sia in assenza sia in presenza di mobilità. Alcuni esempi di applicazioni sono: il monitoraggio strutturale di edifici [50] o tubature [39]; il monitoraggio di vulcani [47] oppure di habitat naturali [23, 28] o ancora di animali al pascolo [36]; infine il monitoraggio di persone anziane o malate [34, 45].

Una'altra tipologia di rete wireless, che condivide svariate caratteristiche con le WSN mobili, è chiamata *Mobile Ad Hoc Network* (MANET). Anche queste reti, infatti, sono distinte da aspetti quali le limitazioni energetiche e trasmissive e ovviamente dalla mobilità. L'utilizzo delle MANET però non è mirato a una funzione specifica, come accade per le WSN, e anche le tecnologie utilizzate per l'implementazione possono essere più varie. Tuttavia, viste le similarità presenti, molti algoritmi e protocolli proposti per le MANET possono essere adatti anche alle WSN.

Una delle operazioni di base che possono essere eseguite su una generica rete e in particolare su una rete wireless come una WSN o una MANET è il broadcast di rete, che nel resto del lavoro chiamiamo semplicemente "broadcast" per brevità. Il broadcast è il processo attraverso cui un nodo della rete invia un messaggio a tutti gli altri. Questa basilare operazione è un blocco costitutivo di molti protocolli più complessi, come ad esempio quelli di routing, che servono a instradare un messaggio affinché esso raggiunga solo uno o più nodi interessati. In tali protocolli, come ad esempio il Dynamic Source Routing (DSR) [12], l'Ad-hoc On-Demand Distance Vector (AODV) [32], lo Zone Routing Protocol (ZPR) [9] e il Context and Content-Based Routing (CCBR) [5], il broadcast è utilizzato per creare e mantenere i percorsi tra differenti nodi all'interno della rete.

Noto che il broadcast non viene solo usato fine a se stesso, ma spesso è

un componente ausiliario per l'esecuzione di compiti più sofisticati, risulta particolarmente evidente la rilevanza di questa operazione e quanto sia importante che la sua esecuzione avvenga nella maniera più efficiente possibile, soprattutto per quanto riguarda il traffico generato.

1.1 Obiettivi del lavoro

Nella presente tesi di laurea, l'attenzione è focalizzata sul broadcast nelle reti wireless, con un particolare riguardo alle WSN, sia in contesti fissi sia mobili, e quindi all'efficienza con cui tale operazione viene eseguita. Il lavoro mira a raggiungere un duplice obiettivo:

- confrontare una nostra tecnica per il broadcast nelle WSN mobili, chiamata in questo lavoro Opportunistic Flooder e brevemente presentata all'interno del protocollo di routing CCBR [5], con un gruppo di altri metodi, opportunamente selezionati dalla letteratura scientifica, tramite:
  a. una serie di simulazioni, adeguatamente scelte per coprire un ampio spettro di situazioni, rappresentative di possibili scenari reali;
  b. un insieme di test eseguiti su reti sperimentali;
  c. i risultati di minimo teorico, per quanto riguarda l'efficienza, ottenuti grazie a un algoritmo appositamente realizzato.
- verificare, in generale, la corrispondenza tra i risultati ottenuti in teoria, simulazione e scenari reali; si ritiene particolarmente importante il confronto tra simulazioni ed esperimenti reali poiché, come dimostrato in [7], nel mondo reale anche tecniche semplicissime possono esibire un comportamento inaspettato, segno rivelatore di una complessità, spesso non tenuta in considerazione, che può influire notevolmente sulle prestazioni di un protocollo.

La scelta di utilizzare un simulatore di reti per eseguire la comparazione è principalmente dovuta al fatto che un ambiente simulato permette di eseguire

molti esperimenti, variando senza sforzo la topologia e gli altri parametri della rete, la mobilità dei nodi e così via. Inoltre le simulazioni, essendo eseguite in un ambiente controllato per definizione, sono ripetibili e ciò consente di confrontare le tecniche nelle stesse identiche situazioni.

Molte delle qualità positive dell'ambiente simulato si perdono passando al caso reale, rendendo di fatto l'esecuzione degli esperimenti più complicata e a volte più difficile da interpretare. Tuttavia, visto che i vari metodi di broadcast — il medesimo discorso può essere esteso a molti altri ambiti — devono poi essere utilizzati in reti reali, è sicuramente importante verificare se il comportamento esibito in simulazione corrisponde alla realtà, in modo da valutare l'affidabilità dell'ambiente di simulazione e quindi poterlo usare con cognizione di causa.

1.2 Struttura del documento

Il presente lavoro è organizzato come segue.

Il capitolo 2 introduce la questione del broadcast nelle reti wireless, evidenziando le principali difficoltà e l'idea fondamentale da perseguire per ottenere una tecnica efficiente. Quindi viene presentato, sia attraverso una schematica tabella sia con una breve descrizione, un folto gruppo di metodi per l'esecuzione del broadcast, opportunamente divisi in categorie.

Nel capitolo successivo, il terzo, compare innanzitutto il percorso che ha portato alla necessaria scelta di un numero limitato di tecniche da confrontare con Opportunistic Flooder. Tutte i metodi scelti per la comparazione sono stati quindi dettagliatamente descritti. La sezione termina con alcune considerazioni inerenti i metodi scelti.

Il quarto capitolo riguarda la ricerca dei risultati teorici necessari a valutare, in assoluto, l'efficienza delle tecniche considerate. La prima parte del capitolo presenta questa ricerca, descrivendone le caratteristiche ed evidenziando i principali problemi, mentre nella seconda viene proposto un algoritmo appositamente creato per eseguire tali ricerche, facendo particolare attenzione alle ottimizzazioni inserite e alla descrizione accurata del suo funzionamento.

Il capitolo 5 concerne l'esecuzione degli esperimenti con il simulatore. Per prima cosa viene introdotto il simulatore utilizzato in questo lavoro, quindi sono presentati gli obiettivi delle simulazioni e gli scenari scelti per compiere gli studi necessari. La parte conclusiva del capitolo riguarda i parametri generali delle simulazioni e la scelta dei parametri per ogni singolo metodo selezionato.

Il capitolo seguente presenta invece i vari aspetti relativi ai test compiuti su reti sperimentali. Nella prima parte vengono presentate le due reti utilizzate, poi l'attenzione si sposta sulle prove di connettività eseguite per effettuare gli esperimenti sulle due reti e sul simulatore in condizioni più simili possibile. Infine vengono presentati gli obiettivi di questa fase sperimentale del lavoro, gli scenari considerati e i parametri relativi sia agli esperimenti sia ai singoli metodi.

Il settimo capitolo raccoglie i risultati prodotti nel corso di questo lavoro. In particolare vengono presentati quelli delle ricerche di minimo teorico, della campagna di simulazioni per la comparazione dei metodi scelti e degli esperimenti effettuati sulle reti sperimentali. Tutti questi risultati sono analizzati, commentati e utilizzati per giungere agli obiettivi che sono stati presentati nel paragrafo precedente.

L'ultimo capitolo del lavoro è riservato alle conclusioni, in cui viene effettuato il resoconto dell'attività svolta e sono evidenziati i principali risultati raggiunti. Infine si propongono possibili lavori futuri, mirati ad approfondire alcuni degli argomenti trattati.

Capitolo 2

Il broadcast nelle reti wireless

La trasmissione di messaggi a tutti i componenti di una rete wireless, operazione comunemente chiamata broadcast, è stata ampiamente trattata in letteratura, soprattutto in seguito alla pubblicazione dell'articolo scritto da S. Ni, Y. Tseng, Y. Chen e J. Sheu [25], in cui viene presentato il problema chiamato *broadcast storm*. Questo problema, dovuto in primo luogo al mezzo trasmissivo utilizzato nelle reti wireless, ovvero le radiofrequenze, che generalmente è condiviso da tutti i componenti della rete, è strettamente legato alla tecnica utilizzata per l'esecuzione del broadcast.

Il modo più semplice con cui è possibile disseminare un messaggio all'interno di una rete è chiamato *flooding*. Il suo funzionamento è piuttosto semplice: alla prima ricezione di un particolare messaggio ogni nodo ritrasmette immediatamente il messaggio stesso, mentre eventuali repliche vengono scartate. Per la sua semplicità, il flooding sembrerebbe essere un'ottima soluzione per eseguire il broadcast; inoltre il flooding ha una caratteristica decisamente importante in reti che possono essere contraddistinte da una componente dinamica non trascurabile, sia essa la mobilità dei nodi, eventuali guasti o entrambe le cose: è una tecnica molto robusta, che promette una buona affidabilità in un ampio spettro di situazioni. Per questo motivo, per esempio in [26], il flooding è stato proposto come possibile soluzione in presenza di mobilità elevata. Tuttavia questa tecnica è la più colpita dal problema del broadcast storm. Il punto debole più evidente del flooding è costituito dalle

trasmissioni ridondanti: tutti i nodi ritrasmettono una volta ogni messaggio ricevuto, perciò ogni nodo riceve lo stesso messaggio da tutti i suoi vicini e quindi, in una generica rete, più volte. Una conseguenza dell'elevato numero di trasmissioni è, senza dubbio, un'elevata possibilità che avvengano conflitti sul canale di trasmissione. Nell'ambiente di nostro interesse, ovvero quello delle WSN, si ricorre generalmente a radio CSMA/CA[1]: nel caso migliore un nodo che vuole trasmettere rileva che il canale è occupato dalla trasmissione di un altro nodo e si mette in attesa, mentre nel peggiore due o più nodi trasmettono contemporaneamente, causando una collisione, senza aver la possibilità di rilevare il problema e di fatto rendendo impossibile la ricezione di alcuni dei messaggi trasmessi. Si noti che, nel caso del flooding, la probabilità di collisione è incrementata dal fatto che nodi adiacenti ricevono quasi contemporaneamente un messaggio trasmesso da un loro vicino e tentano di ritrasmetterlo immediatamente.

Alla luce di quanto appena detto, la soluzione del problema del broadcast storm risiede nella riduzione delle trasmissioni ridondanti e ciò può essere attuato inibendo opportunamente la ritrasmissione di alcuni messaggi da parte di un certo numero di nodi. Gli stessi autori dell'articolo sul problema del broadcast storm presentano, sempre in [25], alcuni possibili meccanismi che permettono di scegliere se un nodo deve partecipare o meno al broadcast di un messaggio e nel corso degli anni sono state escogitate numerose tecniche mirate a ridurre il suddetto problema, in parte già raccolte in alcune interessanti panoramiche e comparazioni [15, 49].

Al momento attuale possiamo distinguere i metodi per il broadcast presentati nella letteratura in cinque macro categorie, a seconda del meccanismo e delle informazioni utilizzate. Le categorie sono le seguenti: metodi basati sulle probabilità, sull'area aggiuntiva coperta, sulla conoscenza dei vicini, su cluster e infine su alberi. A queste cinque categorie ne aggiungiamo una sesta, in cui raccogliamo alcuni protocolli che si basano su presupposti e tec-

[1] Carrier Sense Multiple Access with Collision Avoidance, ovvero accesso multiplo con ascolto della portante per evitare collisioni; è una tecnica che prevede l'ascolto del canale prima di effettuare una trasmissione: se il canale risulta libero la trasmissione viene effettuata immediatamente, altrimenti si attende un tempo casuale e poi si ripete la procedura.

niche differenti rispetto agli altri e che non sono collocabili nelle cinque classi sopramenzionate. Si noti inoltre che non tutti i metodi sono precisamente inseribili in una sola categoria; in questo caso viene scelta quella ritenuta più adeguata.

La restante parte di questo capitolo presenta, per ogni categoria, una raccolta non esaustiva di metodi che ne fanno parte. Si noti che questa intende essere una semplice presentazione, senza alcuna pretesa di spiegare tutti i metodi nel dettaglio: in generale, lo scopo è quello di fornire un'idea del funzionamento delle varie tecniche e lasciare poi al lettore interessato la scelta di quali approfondire attraverso gli articoli di riferimento; fanno eccezione solo i metodi più semplici, che è possibile descrivere in maniera ragionevolmente accurata in poche righe.

Le tecniche presentate in seguito sono raccolte nella tabella 2.1, divise per categorie. Per ogni tecnica sono specificate le informazioni utilizzate, la necessità di scambiare messaggi in aggiunta (MA in tabella) a quelli di broadcast, il fatto che richiedano l'uso di dispositivi GPS, il grado di adattabilità a differenti densità dei nodi nella rete (AD) senza che sia necessario l'intervento umano (per esempio per la modifica di alcuni parametri) e quanto sono adatte per operare in contesti mobili (MOB). Si noti che lo scopo della tabella è raccogliere le informazioni più significative in maniera che siano facilmente confrontabili. Inoltre, per guanto riguarda l'adattabilità rispetto alla diversa densità dei nodi e la capacità di operare in contesti mobili si fa uso di una semplice valutazione, eseguita dall'autore di questo lavoro sulla base delle informazioni disponibili: in particolare, l'assenza dell'asterisco equivale a una valutazione negativa, la presenza di un solo asterisco (*) a una valutazione discreta e infine la presenza di due asterischi (**) a una valutazione buona. Si noti infine che due particolari metodi, essendo notevolmente differenti dal resto delle tecniche, non permettono di inserire dati specifici nella tabella e si è scelto di segnalare questo fatto introducendo un punto di domanda (?).

NOME METODO	INFORMAZIONI UTILIZZATE	MA	GPS	AD	MOB
Metodi basati sulle probabilità					
Gossip	nessuna	no	no		**
Gossip2	vicini (1 hop)	sì	no	*	*
Gossip3	# repliche msg	no	no	*	**
Smart Gossip	vicini (1 hop), mittente hop precedente	sì/no	no	**	*
Metodi basati sull'area aggiuntiva coperta					
Metodo basato sul contatore	# repliche msg	no	no	*	**
Metodo basato sul contatore (adattivo)	# repliche msg, vicini (1 hop)	sì	no	**	*
Metodo basato sulla distanza	distanza mittente	no	no	*	**
Opportunistic Flooder	distanza mittente	no	no	**	**
Metodo basato sulla distanza e sul # di hop	distanza mittente, # hop attraversati msg	no	no	*	**
JDCT	distanza mittente, # repliche msg	no	no	*	**
Metodo basato sulla posizione	posizione mittente	no	sì	*	**
Metodo basato sulla posizione (adattivo)	posizione mittente, vicini (1 hop)	sì	sì	**	*
Metodi basati sulla conoscenza dei vicini					
Irrigator protocol	vicini (1 hop)	sì	no	**	*
Fireworks protocol	vicini (1 hop)	sì	no	*	*
RBP	vicini (1 hop)	sì	no	**	
FONIAH	vicini (1 hop), posizione vicini	sì	sì	**	*
Self pruning	vicini (1 hop), vicini mittente	sì	no	**	*
DFCN	vicini (1 hop), vicini mittente	sì	no	**	*
Metodo basato sulla copertura dei vicini	vicini (2 hop)	sì	no	**	**
Dominant pruning	vicini (2 hop)	sì	no	**	*
Efficient flooding scheme based on 1-hop information	vicini (1 hop), posizione vicini	sì	sì	**	*
SBA	vicini (2 hop)	sì	no	**	*
SBA (adattivo)	vicini (2 hop)	sì	no	**	*
Multipoint relaying	vicini (2 hop)	sì	no	**	*
AHBP	vicini (2 hop), percorso msg	sì	no	**	*
AHBP (adattivo)	vicini (2 hop), percorso msg	sì	no	**	**
Metodo basato sui CDS	vicini (2 hop), percorso msg	sì	no	**	*
Metodo basato sui nodi interni	vicini (2 hop)	sì	sì/no	**	*
LENWB	vicini (2 hop)	sì	no	**	*
Ring Algorithm	vicini (2 hop)	sì	no	**	*
Metodi basati sui cluster					
Metodo SI-CDS	vicini (1 hop)	sì	no	**	*
Metodo backbone SI-CDS	vicini (2 o più hop)	sì	no	**	*
Metodo backbone SD-CDS	vicini (2 o più hop), insieme di copertura testa mittente	sì	no	**	*
Metodi basati su alberi					
TreeCast	vicini (1 hop), ID albero	sì	no	**	*
Metodo collision-free	vicini (2 hop)	sì	no	**	
Altri metodi					
Pure machine learning	svariate, necessarie per l'addestramento	?	?	**	*
Inter-protocol learning	svariate, a seconda delle tecniche coinvolte e dell'addestramento	?	?	**	*

Tabella 2.1: I metodi presentati

2.1 Metodi basati sulle probabilità

Le tecniche prese in considerazione in questa sezione sono quelle in cui ogni nodo decide di ritrasmettere ogni messaggio in base a una certa probabilità. Questa probabilità può essere stabilita a priori, rendendo di fatto inutile ogni conoscenza sulla topologia o lo stato della rete, oppure può variare in base ad alcune informazioni. Come consigliato in [25] è sempre preferibile far in modo che le ritrasmissioni siano precedute da un piccolo ritardo casuale, in modo da limitare conflitti e collisioni sul canale.

2.1.1 Gossip

Il metodo probabilistico più semplice, chiamato Gossip, prevede che ogni nodo, alla prima ricezione di un particolare messaggio, lo ritrasmetta con una probabilità p prestabilita. Non basandosi su alcuna informazione riguardante la rete, questa tecnica risulta essere molto semplice, tuttavia la probabilità p fissata non la rende adatto a reti in cui cambia, ad esempio, la densità dei nodi: infatti un valore di p basso può essere l'ideale per una rete densa, in cui si vuole evitare che molti nodi ritrasmettano, ma non è adatto a una rete sparsa, poiché in tal caso diventa inaccettabile la possibilità che non tutti i nodi della rete ricevano tutti i messaggi. Si noti che se $p = 1$ allora questo metodo è perfettamente equivalente al flooding.

2.1.2 Gossip basato sul numero dei vicini

Questa variante del Gossip, presentata in [10] sotto il nome di Gossip2, cerca di far fronte alle limitazioni del Gossip nelle reti in cui la densità dei nodi non è costante. Rispetto al Gossip originale, i parametri non sono più uno, ma quattro: p_1, equivalente a p; k, ovvero il numero di hop dal nodo di partenza in cui $p_1 = 1$ (questo parametro è aggiunto per evitare che il broadcast si arresti immediatamente); p_2, una probabilità maggiore di p_1; n, cioè una soglia sul numero di nodi vicini. L'idea è che ogni nodo segnali ai suoi vicini di ritrasmettere con probabilità p_2 se il loro numero è minore di n, altrimenti viene usata la probabilità p_1. Gli autori non specificano in che modo ogni no-

do informi i suoi vicini della necessità di ritrasmettere con probabilità pari a p_2. Una possibilità potrebbe essere sfruttare gli stessi messaggi aggiuntivi che è necessario trasmettere periodicamente per conoscere il numero dei vicini: ogni nodo, nota l'indicazione dei suoi vicini, decide quindi se ritrasmettere con probabilità p_1 o p_2 in base al mittente del messaggio di broadcast ricevuto. Un'altra possibilità è inglobare direttamente l'informazione in ogni messaggio di broadcast. Per quanto riguarda i parametri, gli autori indicano 4 come buon compromesso per k, mentre gli altri devono essere scelti a seconda della topologia della rete. Si noti che, come già accennato, ogni nodo ottiene il numero dei suoi vicini mediante lo scambio di opportuni messaggi aggiuntivi.

2.1.3 Gossip basato sul numero di repliche ricevute

In [10] viene anche presentata un'alternativa migliorativa di Gossip2, con il nome Gossip3. In questo caso, oltre alla probabilità di ritrasmissione p e al parametro k, già presentati per Gossip2, viene introdotto un terzo parametro m, soglia sul numero di repliche ricevute dai nodi adiacenti. In pratica ogni nodo che riceve un messaggio, qualora non lo ritrasmetta in base al classico funzionamento del Gossip, tiene conto di quante repliche di tale messaggio riceve dai suoi vicini prima di un timeout ragionevole e se questo numero è inferiore a m ritrasmette il messaggio con $p = 1$. Gli autori consigliano l'utilizzo di $m = 1$. Va evidenziato il fatto che, al contrario di Gossip2, non è necessario che ogni nodo abbia alcuna informazione sul numero dei vicini.

2.1.4 Smart Gossip

Smart Gossip [16] è un protocollo di broadcast capace di adattarsi alla topologia della rete e che richiede come input la sola percentuale di ricezione desiderata. Come nel caso del Gossip, ogni nodo ritrasmette con una probabilità p, che questa volta però dipende dalla topologia della rete e può essere differente per ogni nodo. In particolare, a ogni nodo viene assegnata una misura di importanza per l'esecuzione del broadcast: tale misura varia a seconda di quanto i suoi vicini, per ricevere un messaggio, necessitano che

esso lo ritrasmetta e pertanto ne consegue che la probabilità di ritrasmissione p è direttamente proporzionale a questa misura e viene calcolata con un'apposita formula. Ogni nodo, secondo gli autori, può ottenere le informazioni necessarie sulla topologia locale della rete semplicemente basandosi sui messaggi di broadcast ricevuti da altri nodi, non richiedendo quindi lo scambio di appositi messaggi di controllo. Qualora la rete esibisca un comportamento dinamico accentuato tuttavia è probabile che tali messaggi di controllo siano necessari, a meno che la frequenza dei messaggi di broadcast non sia adeguatamente elevata. Da questa breve presentazione risalta che Smart Gossip permette di superare la principale limitazione di Gossip ed evita di dover selezionare manualmente parametri la cui scelta può non essere semplice.

2.2 Metodi basati sull'area aggiuntiva coperta

In questa categoria raccogliamo i metodi che basano la decisione sulla ritrasmissione di un messaggio sull'area aggiuntiva che ogni nodo è in grado di coprire ritrasmettendo tale messaggio. Più quest'area è piccola più è probabile che la ritrasmissione sia ridondante e quindi da evitare. Questo fatto è analizzato e discusso approfonditamente in [25].

2.2.1 Metodo basato sul contatore

Questo primo metodo si basa sul fatto che, come mostrato in [25], l'area aggiuntiva attesa coperta da ogni nodo diminuisce all'aumentare del numero di nodi suoi vicini che ritrasmettono il messaggio. L'idea è quindi usare, su ogni nodo, un contatore c per ogni messaggio, che sia incrementato ogni volta che questo viene ricevuto e su cui sia basata la scelta di ritrasmettere o meno per mezzo di una soglia C preimpostata. In particolare il funzionamento del metodo è il seguente: alla prima ricezione di un messaggio si pone $c = 1$ e il messaggio viene inserito in una coda di ritrasmissione con un ritardo casuale; se all'interno di questo tempo di attesa lo stesso messaggio viene nuovamente ricevuto allora $c = c+1$ e se $c = C$ il messaggio viene cancellato dalla coda e quindi non ritrasmesso. Riassumendo, un messaggio viene ritrasmesso solo se,

dopo un ritardo casuale, è stato ricevuto meno di C volte. I valori consigliati per la soglia sono $C < 6$ e in particolare i migliori sembrano essere $C = 3$ e $C = 4$, ma si noti che il valore ottimale di C dipende dalla densità dei nodi nella rete.

2.2.2 Metodo basato sul contatore (adattivo)

Per superare il problema della scelta di C, che si pone usando il metodo basato sul contatore, è possibile ricorrere a una sua estensione [43], in cui ogni nodo è in grado di adattare il valore di C in base al numero dei suoi vicini diretti. Il funzionamento dell'algoritmo è esattamente lo stesso della sua versione base, soltanto che al posto di C è presente la funzione $C(n)$, dove n è il numero dei vicini del nodo. Gli autori ricavano la forma ottimale di questa funzione attraverso una serie di simulazioni e raffinamenti successivi e viene dimostrato come questo adattamento garantisca migliori prestazioni rispetto alla versione base. Tuttavia non bisogna dimenticare che questa variante richiede lo scambio di messaggi aggiuntivi per permettere a ogni nodo di conoscere il numero dei suoi vicini.

2.2.3 Metodo basato sulla distanza

Rispetto al metodo basato sul contatore, in questo caso utilizziamo una differente informazione per approssimare quale sia l'area aggiuntiva coperta da un nodo, ovvero la distanza del nodo da cui viene ricevuto un messaggio di broadcast o una sua replica. Come dimostrato in [25], più questa distanza è piccola minore può essere l'area aggiuntiva coperta. Supponiamo allora di utilizzare lo stesso schema dell'algoritmo basato su contatore, soltanto che, invece del contatore, per ogni messaggio viene mantenuta una variabile d_{min}, in cui viene salvata la minima distanza da cui è stata ricevuta una copia del messaggio e questa variabile viene confrontata con una soglia D; qualora all'aggiornamento di d_{min} si verifichi la condizione $d_{min} < D$ il messaggio viene cancellato dalla coda di trasmissione, altrimenti si continua ad attendere la scadenza del timeout. Per ottenere la distanza del nodo che ha trasmesso è possibile ricorrere alla misura della potenza del segnale relativa al messaggio

ricevuto, fornita dal livello MAC[2] come RSSI[3]. La scelta di D presenta le stesse difficoltà incontrate in quella di C nel metodo basato sul contatore, senza dimenticare che essa è derivata dal valore di RSSI e questo non è uno standard, ma può cambiare a seconda della radio utilizzata.

2.2.4 Opportunistic Flooder

Questa tecnica [5] si basa sempre sulla misura del RSSI e quindi idealmente sulla misura della distanza, ma il suo funzionamento differisce dal metodo appena presentato in più punti. In primo luogo non è presente una soglia sulla distanza, ma la tecnica si impernia su ciò che viene chiamato *delay and cancel*: ogni nodo, alla prima ricezione di un messaggio, calcola un ritardo inversamente proporzionale alla distanza del nodo che lo ha trasmesso e pone il messaggio nella coda di ritrasmissione; se viene ricevuta una replica del messaggio prima dello scadere del ritardo allora il messaggio viene cancellato, altrimenti viene ritrasmesso. In questo modo ogni messaggio dovrebbe essere ritrasmesso solamente dai nodi più lontani, limitando dunque le trasmissioni ridondanti.

2.2.5 Metodo basato sulla distanza e sul numero di hop

Una seconda variante al metodo basato sulla distanza è costituita dalla tecnica presentata in [14] e ora brevemente introdotta. In questo caso le informazioni utili sono ancora la distanza del nodo da cui viene ricevuto un messaggio di broadcast e il numero di hop che tale messaggio ha già compiuto. Alla prima ricezione di un messaggio di broadcast il nodo inizializza una variabile $HC0$ con il numero di hop già compiuti dal messaggio e calcola gli estremi dell'intervallo in cui viene scelto casualmente il ritardo per la ritrasmissione basandosi sulla distanza dal nodo da cui ha ricevuto il messaggio, su una soglia D su tale distanza, sul raggio di trasmissione e su un tempo

[2]Media Access Control, ovvero controllo di accesso al mezzo fisico; è un sottolivello dello standard IEEE 802 e la sua funzione è regolare l'accesso e la condivisione del mezzo trasmissivo.

[3]Received Signal Strength Indicator, un indicatore della potenza del segnale ricevuto.

massimo di attesa predefinito; quindi inserisce il messaggio nella coda per ritrasmetterlo. Qualora venga ricevuta una replica da una distanza inferiore a D e se il numero di hop compiuti dalla replica è maggiore di $HC0$ allora viene cancellata la ritrasmissione. Come per il metodo basato sulla distanza, anche in questo caso è necessario stabilire una soglia, che dipende dalla topologia della rete e influisce sui risultati ottenuti.

2.2.6 Joint Distance-Counter Threshold Broadcast Algorithm (JDCT)

Questo algoritmo di broadcast [17] sfrutta contemporaneamente le informazioni sul numero di messaggi replicati ricevuti e sulla distanza dei nodi adiacenti che li hanno trasmessi. Ogni nodo quindi mantiene sia un contatore c sia la variabile d_{min} per ogni messaggio di broadcast in attesa di essere ritrasmesso e usa le due soglie C e D, che riguardano rispettivamente contatore e distanza. Alla ricezione del primo messaggio e delle sue repliche viene incrementato il contatore, calcolata la distanza del nodo che lo ha trasmesso ed eventualmente aggiornata d_{min}; se $d_{min} > D$ il nodo attende un breve periodo di tempo e se nessun'altra replica viene ricevuta ritrasmette il messaggio, mentre se $d_{min} \leq D$ valuta anche la condizione sul contatore: se $c < C$ il nodo attende un breve periodo di tempo e se nessun'altra replica viene ricevuta ritrasmette il messaggio, diversamente cancella il messaggio dalla coda di ritrasmissione. L'algoritmo promette prestazioni superiori a quelle ottenibili dai metodi basati sul contatore e sulla distanza, grazie a un opportuno utilizzo di entrambe le soglie.

2.2.7 Metodo basato sulla posizione

Supponiamo ora che ogni nodo abbia a disposizione informazioni precise sulla sua posizione e su quella degli altri nodi da cui riceve un messaggio, per esempio grazie all'uso di dispositivi GPS[4]. Note tali informazioni risulta possibile calcolare quale sia l'area aggiuntiva coperta da un'eventuale ritrasmissione

[4]Global Positioning System, un sistema di posizionamento che permette di calcolare la posizione in base alle informazioni ricevute da una costellazione di satelliti artificiali.

[25]. Sia AC il valore calcolato per l'area aggiuntiva coperta e utilizziamo una soglia A per decidere se il nodo deve ritrasmettere o meno, seguendo lo stesso schema di algoritmo valido per i metodi basati su contatore e distanza: in particolare, a ogni messaggio replicato ricevuto va aggiornato AC tenendo ovviamente conto di tutti le repliche precedenti e se $AC < A$ si cancella il messaggio dalla coda di trasmissione, altrimenti si attende la scadenza del timeout. Questo metodo però presenta un problema non trascurabile: il calcolo di AC, come affermato dagli autori, è computazionalmente impegnativo e ciò può essere un problema soprattutto quando i nodi hanno risorse limitate sia dal punto di vista della potenza di calcolo sia da quello dell'energia a disposizione. È possibile ricorrere, sempre secondo gli autori, a un'alternativa per evitare il complesso calcolo: qualora il nodo sia all'interno del poligono convesso formato dai vicini da cui ha ricevuto una replica allora non è necessario che ritrasmetta anch'esso, poiché l'area aggiuntiva coperta è minima o addirittura nulla, mentre se non si trova all'interno di tale poligono il nodo deve ritrasmettere.

2.2.8 Metodo basato sulla posizione (adattivo)

Così come per il metodo basato sul contatore, anche per quello basato sulla posizione è stato proposta una versione in grado di adattare la scelta di ogni singolo nodo alla topologia locale da lui rilevata. Al posto della soglia costante A viene usata una funzione $A(n)$, ancora una volta dipendente dal numero dei nodi a 1 hop di distanza. In [43] viene proposta una funzione da usare come $A(n)$, ricavata per via sperimentale. Anche in questo caso la variante adattiva richiede lo scambio di messaggi aggiuntivi, necessari a ottenere il numero di nodi vicini.

2.3 Metodi basati sulla conoscenza dei vicini

La presente categoria raccoglie i metodi che basano la decisione di ritrasmissione su un qualche grado di conoscenza dei nodi vicini. In alcuni casi il nodo che riceve il messaggio di broadcast, sfruttando le informazioni sul suo

vicinato, decide se ritrasmettere o meno, mentre in altri è il nodo trasmittente che sceglie quali dei suoi nodi vicini devono a loro volta ritrasmettere il messaggio. In generale, i metodi qui presentati necessitano lo scambio periodico di messaggi di controllo, chiamati in letteratura *hello packets*, aggiuntivi al normale traffico dati. Si noti che la scelta del periodo deve tener conto delle condizioni della rete, soprattutto per quanto riguarda la mobilità o le possibili disconnessioni dei nodi.

2.3.1 Irrigator protocol

In questo primo metodo [27], ogni nodo utilizza la conoscenza dei suoi vicini diretti, ovvero quelli a un solo hop di distanza, per sceglierne casualmente un certo numero c; quindi comunica a tali nodi la sua scelta. Per ottenere tale conoscenza è sufficiente l'utilizzo di un semplice messaggio di controllo contenente il proprio ID, da inviare a tutti i vicini diretti, mentre per comunicare la propria scelta si può usare un secondo messaggio di controllo contenente, in aggiunta, la lista dei c nodi selezionati. La scelta dei c nodi e la conoscenza della selezione effettuata dai propri vicini permette di costruire una topologia virtuale, in cui due nodi sono collegati da un arco se almeno uno dei due ha scelto l'altro. Questa topologia virtuale, di cui ogni singolo nodo ha una conoscenza solo locale, viene usata per la diffusione dei messaggi di broadcast all'interno della rete: ogni nodo, alla ricezione di un messaggio, verifica se il suo identificativo è nella lista del mittente oppure se esso si trova tra i c nodi da lui scelti e in tal caso ritrasmette; in altre parole, è sufficiente che almeno uno tra mittente e ricevente abbia scelto l'altro affinché il ricevente ritrasmetta il messaggio. Gli autori sostengono che se la rete di nodi è connessa allora anche la topologia virtuale creata dai collegamenti tra i nodi scelti con questo metodo è connessa se $c \geq 4.5$.

In [27] viene anche presentata una variante dell'Irrigator, definita Irrigator protocol v2.0. Ciò nasce dalla seguente considerazione: nella prima versione è probabile che il grado della topologia virtuale sia inutilmente superiore a 4.5 poiché i vicini di ogni nodo, in tale topologia, comprendono sia quelli scelti dal nodo stesso sia quelli che lo hanno scelto. Nella v2.0 è presente una

prima fase identica alla versione base del protocollo, solo con un c inferiore (es. $c = 2$); al termine di questa fase ogni nodo verifica se la sua lista di vicini nella topologia virtuale, ossia quella formata sia dai vicini che ha scelto sia da quelli che lo hanno scelto, contiene un numero n di nodi inferiore a $c^* = 4$ e in tal caso inizia una seconda fase in cui sceglie altri $c^* - n$ e comunica loro tale decisione. Così facendo si cerca di ottenere $n \approx c^*$ per tutti i nodi della rete.

2.3.2 Fireworks protocol

Fireworks, sempre proposto in [27], è un protocollo molto semplice che prende spunto sia dall'Irrigator sia da Gossip: il nodo sorgente del messaggio di broadcast deve trasmettere a tutti i suoi vicini mentre qualsiasi altro nodo, alla ricezione del messaggio, trasmette a tutti i suoi vicini con probabilità p, mentre con probabilità $1 - p$ trasmette solo a c vicini scelti casualmente. Per comunicare ai nodi vicini questa scelta è necessario inserire la lista dei c nodi nel messaggio di broadcast.

2.3.3 Robust Broadcast Algorithm (RBP)

Questo algoritmo [38] si differenzia dalla maggior parte degli altri presentati in questa panoramica siccome il suo obiettivo è ottenere una maggiore affidabilità per il singolo broadcast piuttosto che una maggiore efficienza. Questo può essere utile in presenza di applicazioni che richiedono che il tasso di copertura per un messaggio di broadcast sia superiore a una certa percentuale (possibilmente molto elevata). RBP richiede che ogni nodo conosca i suoi vicini diretti e tra questi consideri tali solo quelli raggiunti da un segnale sufficientemente stabile, che quindi non sono ai limiti del raggio di trasmissione. Il broadcast funziona in questo modo: ogni nodo, alla prima ricezione di un messaggio, ritrasmette incondizionatamente e tiene traccia delle ritrasmissioni dei suoi vicini, usandole come ACK[5] impliciti; se queste non superano una soglia dipendente dalla densità dei nodi ritrasmette nuovamente. Un nodo

[5] Abbreviazione di acknowledgment, che in informatica indica un segnale di conferma.

r che riceve i messaggi da un solo mittente s lo avvisa di questa situazione con un apposito messaggio di controllo. Quindi s, in caso di mancato ACK implicito da parte di r, trasmetterà più volte i messaggi successivi, per assicurare che r non li perda e che quindi eventuali porzioni della rete ricevano con maggiori probabilità i messaggi di broadcast.

2.3.4 Flooding based on One-hop Neighbor Information and Adaptive Holding (FONIAH)

Quello presentato in [18] è un metodo ibrido che utilizza sia la conoscenza dei vicini diretti sia quella della posizione di ogni nodo, ricavata con il GPS. Tramite messaggi di controllo ogni nodo viene a conoscenza dei suoi vicini a un hop di distanza e della loro posizione. All'interno di un messaggio di broadcast il mittente deve inserire la sua posizione e la distanza del nodo adiacente a lui più lontano. Un nodo che riceve un messaggio di broadcast deve attendere per un tempo dipendente sia dalla distanza massima inserita dal mittente sia da quella calcolata dal nodo stesso grazie alle informazioni a sua disposizione, ovvero per un periodo inversamente proporzionale alla distanza dal mittente. Le repliche del messaggio, ricevute durante questo tempo di attesa, permettono al nodo di stimare quali dei suoi vicini abbiano ricevuto il messaggio e quindi di decidere se cancellare o meno la ritrasmissione.

2.3.5 Self pruning

In questo metodo [20] ogni nodo deve essere a conoscenza dei suoi vicini diretti, ovvero quelli a un solo hop di distanza. Affinché ciò accada, ogni nodo emette periodicamente un messaggio di controllo con cui comunica ai nodi vicini la sua presenza. Quando un nodo vuole trasmettere un messaggio di broadcast aggiunge a tale messaggio la lista dei suoi nodi vicini. Alla ricezione di un messaggio di broadcast, ogni nodo confronta la sua lista con quella inserita nel messaggio e decide di ritrasmettere solo se la sua lista contiene nodi differenti rispetto a quella ricevuta. Il self pruning quindi impedisce

la ritrasmissione quando questa, almeno in linea teorica, non apporta alcun beneficio.

2.3.6 Delayed Flooding with Cumulative Neighborhood (DFCN)

Questo metodo, presentato in [11], segue un approccio simile a quello del self pruning, ma più elaborato. È richiesta la conoscenza dei soli vicini diretti, ottenibile tramite semplici messaggi di controllo. Al contrario del self pruning però l'eventuale ritrasmissione viene preceduta da un ritardo casuale, che permette al nodo di tenere in considerazione eventuali repliche del messaggio nella decisione di ritrasmissione. Il funzionamento di questa tecnica è il seguente: ogni nodo, alla prima ricezione di un messaggio, crea un insieme K di nodi già raggiunti sulla base delle informazioni aggiunte al messaggio e si mette in attesa per un tempo casuale, mentre alla ricezione di una replica aggiorna l'insieme K con i nuovi nodi eventualmente coperti; al termine del tempo d'attesa il nodo calcola la percentuale di suoi vicini che non hanno ricevuto il messaggio e solo se questa è superiore a una soglia prestabilita (gli autori consigliano 0.4) il messaggio viene ritrasmesso; infine se il numero dei vicini è inferiore a una certa soglia (viene suggerito 4) e il nodo si accorge di un nuovo vicino allora tutti i messaggi in attesa vengono immediatamente spediti.

2.3.7 Metodo basato sulla copertura dei vicini

Questa tecnica, presentata in [43], si differenzia da DFCN per alcuni particolari e aggiunge un'estensione mirata all'adattività. Innanzitutto questo metodo sfrutta la conoscenza dei vicini fino a 2 hop di distanza, ottenuta mediante opportuni messaggi di controllo, invece di inserire informazioni aggiuntive in ogni messaggio di broadcast. Quindi prevede che ogni nodo, alla prima ricezione di un messaggio di broadcast, inserisca tutti i suoi vicini diretti in un insieme di nodi scoperti T e scelga un ritardo casuale per la ritrasmissione; a ogni ricezione dello stesso messaggio, compresa la prima,

vengono rimossi da T tutti i nodi che risultano essere vicini diretti del mittente del messaggio e la ritrasmissione viene cancellata se $T = \emptyset$, altrimenti il messaggio viene ritrasmesso allo scadere del ritardo. Infine viene proposto un comportamento dinamico, dipendente da una stima quantitativa della variazione del vicinato di ogni nodo, per il periodo con cui ogni nodo emette il messaggio di controllo: più precisamente, il periodo dovrebbe essere inversamente proporzionale a questa stima e variare all'interno di due valori predefiniti. Si noti che, utilizzando un intervallo dinamico per l'emissione dei messaggi di controllo, è necessario che questo periodo venga comunicato ai nodi adiacenti, affinché questi possano aggiornare le informazioni sulla loro topologia locale tenendone conto.

2.3.8 Dominant pruning

Gli autori di [20] propongono anche il dominant pruning, che differisce dal self pruning per due punti fondamentali: primo, la conoscenza di ogni nodo è estesa ai vicini fino a due hop di distanza; secondo, non è il nodo che riceve il messaggio di broadcast a decidere se ritrasmetterlo, ma è quello che lo trasmette a scegliere quali nodi tra i suoi vicini debbano ritrasmettere. Per mantenere una conoscenza dei vicini fino a due hop di distanza è necessario che i nodi inseriscano nei messaggi di controllo la propria lista dei vicini diretti. In base a tale conoscenza ogni nodo, all'atto di trasmettere un messaggio di broadcast, deve decidere quali dei suoi vicini diretti devono a loro volta trasmettere e quindi aggiungere la lista dei loro identificativi al messaggio. L'idea di base è che questa lista permetta di raggiungere tutti i nodi a due hop di distanza e al contempo sia più piccola possibile, per ridurre il numero di trasmissioni. Trovare la lista più piccola è però un problema complesso e quindi si ricorre a un algoritmo greedy che seleziona il vicino diretto che permette di raggiungere il maggior numero di nodi a due hop di distanza ancora scoperti e lo aggiunge alla lista, ripetendo l'operazione fino a copertura completa.

2.3.9 Efficient Flooding Scheme Based on 1-hop Information

Questa tecnica [21] segue l'idea del dominant pruning, ovvero ogni nodo che trasmette un messaggio di broadcast calcola quali tra i suoi vicini debbano a loro volta ritrasmettere, ma basa la scelta su diverse informazioni. È richiesta la conoscenza solo dei vicini diretti, ma di questi è necessario sapere anche la posizione (ad esempio tramite GPS). Utilizzando la posizione dei nodi adiacenti è possibile calcolare con precisione i nodi che devono ritrasmettere per garantire la copertura dell'intera rete, pur non essendo a conoscenza dei vicini a due hop di distanza. Gli autori provano che il numero di nodi scelti sia localmente minimo (e può essere ulteriormente ottimizzato conoscendo l'intero insieme scelto dal mittente del messaggio) e che l'algoritmo di scelta abbia la complessità minore possibile. Viene proposto inoltre un algoritmo che, sfruttando un aggiornamento incrementale, permette a ogni nodo di reagire efficientemente a eventuali modifiche nella topologia.

2.3.10 Scalable Broadcast Algorithm (SBA)

Un'altra tecnica che si basa sulla conoscenza dei nodi vicini fino a due hop di distanza è SBA [29] e il suo funzionamento è molto simile a quello del metodo basato sulla copertura dei vicini, presentato in precedenza (2.3.7). La conoscenza dei vicini fino a 2 hop, insieme all'identità del nodo da cui viene ricevuto il messaggio di broadcast, è sufficiente a chi riceve per sapere se una sua eventuale ritrasmissione coprirebbe nodi aggiuntivi. Anche in questo caso la conoscenza sui vicini è acquisita mediante messaggi di controllo che contengono l'identificativo del mittente e la lista dei suoi vicini. Il funzionamento dell'algoritmo è il seguente: quando un nodo riceve un messaggio di broadcast è a conoscenza di tutti i suoi vicini già raggiunti dalla stessa trasmissione e opta per la ritrasmissione solo se rimangono ancora vicini scoperti; qualora decida di ritrasmettere, crea un insieme C contenente i nodi già coperti, inserisce il messaggio nella coda di ritrasmissione con un certo ritardo, il cui valore massimo dipende dal rapporto tra il proprio numero di vicini e il massimo numero di vicini tra i vicini conosciuti, e se all'interno

del tempo di attesa riceve delle repliche del messaggio aggiunge a C i nuovi nodi coperti e verifica se la sua ritrasmissione è ancora utile; i messaggi non cancellati vengono trasmessi al termine del periodo di attesa.

2.3.11 Scalable Broadcast Algorithm adattivo (SBA adattivo)

Nell'interessante e ampia comparazione di Williams e Camp [49] viene fatto notare come le prestazioni del protocollo SBA siano fortemente dipendenti dal valore massimo scelto per il ritardo di ritrasmissione. Per questo motivo viene suggerita una semplice, ma efficace, estensione a SBA, in cui questo ritardo massimo viene scelto tra due valori a seconda del numero di messaggi ricevuti, in media, per secondo. Si noti però che questa estensione ha senso nel caso il traffico nella rete possa essere piuttosto elevato.

Un altro ricercatore propone di rendere adattivo SBA sfruttando tecniche di machine learning [4]. In particolare il valore del ritardo massimo viene scelto da ogni nodo grazie a un semplice modello di regressione basato su due ingressi, ovvero il numero di pacchetti ricevuti al secondo e il numero di vicini diretti. È importante sottolineare che questa tecnica, sicuramente più raffinata della decisione su due valori, richiede l'addestramento del modello usato, al fine di ottenere i parametri per la regressione.

2.3.12 Multipoint relaying

Multipoint relaying [33] è un altro metodo che sfrutta la conoscenza dei nodi vicini fino a due hop di distanza. L'idea di fondo è la stessa del dominant pruning, ovvero ogni nodo deve scegliere un sottoinsieme dei suoi vicini diretti che garantisca la copertura di tutti i nodi distanti due hop, sempre facendo in modo che il numero di ritrasmissioni sia più limitato possibile. Ogni nodo però comunica l'insieme calcolato di multipoint relays (MPR) ai suoi vicini, inserendolo nei messaggi di controllo e non in ognuno dei messaggi di broadcast; l'aggiornamento di tale insieme avviene in reazione a cambiamenti nella topologia locale del nodo. Come nel caso del dominant pruning, la

scelta di questo insieme è affidata a un'euristica, ora descritta: per prima cosa vengono inseriti nell'insieme quei nodi a un hop che sono gli unici a raggiungere almeno un nodo distante due hop; poi, fino a quando non sono rimasti nodi scoperti a tale distanza, viene scelto il nodo vicino diretto che raggiunge il maggior numero di nodi ancora scoperti a 2 hop di distanza. Si noti come il problema della scelta sarebbe risolto anche senza il primo passo, tuttavia questo è utile per ottimizzare l'insieme scelto, visto che inserisce immediatamente quei nodi che è sicuro debbano ritrasmettere.

2.3.13 Ad Hoc Broadcast Protocol (AHBP)

Il protocollo AHBP [30] utilizza un approccio molto simile al multipoint relaying: infatti è richiesta una conoscenza dei nodi vicini fino a due hop di distanza e ogni nodo, quando deve trasmettere un messaggio di broadcast, deve scegliere un insieme di suoi vicini diretti che ritrasmettano il messaggio. In questo caso, i nodi scelti per la ritrasmissione sono detti Broadcast Relay Gateway (BRG). AHBP sfrutta un maggior numero di informazioni rispetto al multipoint relaying per la scelta dei BRG: questa scelta infatti viene effettuata al momento della trasmissione di un messaggio di broadcast basandosi anche sulla lista dei nodi per cui il messaggio è passato; poi l'insieme dei BRG viene inserito direttamente nell'header del messaggio da trasmettere. Per il resto la tecnica di scelta dei BRG segue gli stessi passi di quella degli MPR.

Per il protocollo AHBP è stata anche proposta un'estensione, chiamata AHBP-EX, pensata per affrontare il problema della mobilità dei nodi, che può essere causa di inconsistenze nelle informazioni utilizzate per scegliere i BRG. L'estensione prevede che quando un nodo riceve un messaggio da un altro nodo che non è presente nella sua lista dei vicini si elegge automaticamente a BRG, ritrasmettendo quindi il messaggio. L'introduzione di questa semplice regola dovrebbe avere il seguente effetto: aumentare la robustezza in presenza di mobilità a discapito di un maggior numero di ritrasmissioni (alcune delle quali, eventualmente, non necessarie).

2.3.14 Ad Hoc Broadcast Protocol adattivo (AHBP adattivo)

AHBP-EX, come abbiamo visto, permette di gestire il caso in cui un nodo si sposta nel raggio di trasmissione di un altro nodo e riceve da quest'ultimo un messaggio di broadcast prima che entrambi vengano a conoscenza del fatto di essere vicini diretti. Invece non viene considerata la situazione in cui due nodi non sono più vicini diretti, ma uno sceglie l'altro come BRG in base a informazioni obsolete; ciò rende probabile che il messaggio di broadcast non raggiunga alcuni nodi che dovevano essere proprio coperti dal BRG non più raggiungibile. Questo problema appare evidente nei risultati delle simulazioni in [49], ove AHBP-EX, pur migliorando i risultati della versione base, garantisce una copertura della rete inferiore a quella di protocolli come SBA. Una soluzione per questo problema potrebbe essere variare l'intervallo di tempo di emissione dei messaggi di controllo, adattandolo alla mobilità del nodo, facendo però attenzione che un periodo troppo breve aumenta notevolmente l'overhead. In alternativa viene proposta in [4] un'estensione per AHBP che introduce una misura di confidenza sull'effettiva corrispondenza tra le informazioni a disposizione del nodo e la situazione reale, basata per esempio sulla velocità del nodo e il numero dei vicini diretti; questo potrebbe permettere al nodo di scegliere i BRG in maniera più consapevole e conservativa.

2.3.15 Metodo basato sui Connected Dominating Sets (CDS)

Questo metodo, proposto in [31] dagli stessi autori di SBA e AHBP, risulta essere molto simile ad AHBP, visto che ogni nodo calcola i BRG al momento di trasmettere un messaggio di broadcast. In questo caso la procedura si basa proprio sul calcolo distribuito di un CDS, ovvero di un insieme di nodi della rete che sia connesso, cioè in cui esiste un percorso che unisce tutte le coppie di nodi appartenenti all'insieme, e per cui tutti i nodi che non fanno parte dell'insieme siano distanti al massimo un hop da un nodo che ne fa parte. Rispetto ad AHBP, la scelta di un nodo x è basata non solo

sulle informazioni riguardanti i nodi fino a due hop di distanza e sui nodi attraversati dal messaggio fino a questo punto, ma anche sui BRG scelti dal mittente che hanno priorità maggiore rispetto a x, cioè che sono stati scelti prima di x nella procedura: questo significa che ogni nodo deve rimuovere dall'insieme di tutti i suoi vicini a due hop di distanza tutti quelli che sono vicini diretti di ognuno dei BRG con priorità maggiore alla sua.

2.3.16 Metodo basato sui nodi interni

In [40] viene proposta una tecnica di broadcast che si appoggia alla ricerca di un CDS. A ogni nodo deve essere assegnato un ID e questi devono essere confrontabili tra loro. Vengono date tre definizioni di nodi interni: un nodo viene detto *intermedio* se esistono due suoi vicini che non sono direttamente collegati tra loro; un nodo intermedio è considerato *inter-gateway* se non esiste un altro nodo intermedio a lui collegato che copre tutti i suoi vicini più uno e ha ID maggiore del suo; infine un inter-gateway diventa *gateway* se non esistono altri due inter-gateway a lui collegati che insieme coprono tutti i suoi vicini più uno e almeno uno dei loro ID è minore del suo. Si noti che è necessario che ogni nodo conosca i suoi vicini fino a due hop di distanza. Gli autori propongono quindi di far ritrasmettere solo i nodi interni e suggeriscono come miglior scelta i soli nodi gateway. Gli autori propongono anche un modo per sfruttare il GPS ed evitare che i nodi debbano scambiarsi informazioni sui loro vicinati, ma comunque è previsto che vengano scambiate alcune informazioni mediante messaggi di controllo.

2.3.17 Lightweight and Efficient Network-Wide Broadcast (LENWB)

Anche questo algoritmo [41] si fonda sulla conoscenza dei nodi distanti fino a due hop. La scelta della ritrasmissione, in questo caso, dipende dalla priorità di ogni singolo nodo e quest'ultima è direttamente proporzionale al numero di vicini diretti. Grazie alla conoscenza dei suoi vicini fino a 2 hop di distanza, ogni nodo è in grado di sapere quali tra i suoi vicini diretti hanno priorità

maggiore rispetto alla sua e quali inferiore. Alla prima ricezione di ogni messaggio un nodo può sapere, noto il mittente, quali dei suoi vicini diretti con priorità maggiore della sua possono ritrasmettere tale messaggio e quindi calcolare se, dopo tali ritrasmissioni, la rimanente parte dei suoi vicini diretti risulta coperta e quindi stabilire se è necessario ritrasmettere il messaggio.

2.3.18 Ring Algorithm

L'ultima tecnica di questa categoria, chiamata Ring Algorithm, viene proposta in [19]. L'idea di fondo è cercare di ridurre il numero di nodi che devono trasmettere, nonché la complessità dei calcoli necessari alla loro scelta, basandosi sul fatto che se esiste un ciclo formato da tutti i vicini diretti di un nodo allora è molto probabile che la ritrasmissione di questo nodo risulti ridondante. I passi compiuti dall'algoritmo sono i seguenti: i nodi devono poter essere ordinati (per esempio in ordine crescente in base al loro ID) e ogni nodo con più di un vicino verifica se altri due nodi consecutivi (più l'ultimo con il primo) sono connessi direttamente oppure attraverso un altro nodo che nella lista viene prima di loro; se ciò è sempre verificato allora esiste un ciclo e il nodo decide di non ritrasmettere i messaggi di broadcast, altrimenti decide di ritrasmetterli. Le informazioni richieste dall'algoritmo riguardano dunque, per ogni nodo, i vicini diretti e i loro vicini fino a un hop di distanza; perciò ogni nodo deve conoscere i suoi vicini fino a due hop di distanza. Gli autori non specificano alcunché sullo scambio delle informazioni tra i nodi, ma si può supporre avvenga tramite opportuni messaggi di controllo; in presenza di mobilità ogni nodo deve inoltre eseguire l'algoritmo per rivalutare la sua decisione a ogni cambiamento rilevato tra i suoi vicini.

2.4 Metodi basati sui cluster

I metodi che fanno parte di questa classe si basano sul concetto chiamato clustering per selezionare quali nodi debbano ritrasmettere i messaggi di broadcast. L'idea fondamentale è quindi quella di creare, all'interno della rete di nodi, una struttura formata da più cluster, cioè una struttura con due

livelli gerarchici. Ogni cluster è un insieme di nodi vicini: più precisamente, in ogni cluster è presente un solo nodo dominante, chiamato testa del cluster; tra tutti gli altri nodi membri del cluster quelli che hanno tra i loro vicini diretti almeno un nodo appartenente a un altro cluster sono i gateway. Si noti che le tecniche basate sui cluster possono essere divise in due categorie, a seconda se la struttura creata è totalmente indipendente dalla sorgente del broadcast (basate su Source Independent CDS, SI-CDS per brevità) oppure se dipende da essa per quanto riguarda la selezione dei gateway e quindi viene creata dinamicamente a ogni trasmissione (basate su SD-CDS). È importante evidenziare come svariati metodi basati sulla conoscenza dei vicini, come ad esempio il Dominant Pruning, il Multipoint Relay, l'AHBP, siano di fatto dei metodi SD-CDS, mentre altri (un'esempio è quello basato sui nodi interni) sono dei metodi SI-CDS. In questa categorie ci concentriamo invece sui metodi esplicitamente basati sui cluster.

2.4.1 Metodo 1 basato sui cluster (SI-CDS)

In questo primo metodo, presentato in [25] come possibile soluzione al problema del broadcast storm, ogni nodo è caratterizzato da un ID univoco e deve emettere periodicamente un messaggio di controllo per avvertire i suoi vicini della propria presenza. La formazione di un cluster segue queste regole: il nodo che ha l'ID più piccolo rispetto a tutti i suoi vicini si elegge testa del cluster e tutti i suoi vicini diventano membri del cluster; tra questi, i nodi che hanno tra i loro vicini un componente di un altro cluster diventano dei gateway. Se i nodi possono muoversi, quando due teste si incontrano quella con l'ID maggiore diventa un normale membro. Quando la struttura a cluster è formata, il broadcast di un messaggio viene eseguito in questo modo: alla ricezione di un messaggio questo può essere ritrasmesso solamente se il nodo è una testa o un gateway; tali nodi possono eventualmente ricorrere a un'ulteriore tecnica (per esempio Gossip o basata sulla distanza) per decidere se ritrasmettere o meno, visto che all'interno di un cluster potrebbero anche esserci numerosi nodi gateway e ciò potrebbe causare un numero eccessivo di ritrasmissioni.

2.4.2 Metodo 2 basato sui cluster (backbone SI-CDS)

Un altro metodo basato sui cluster è descritto in [22]. Ogni cluster si forma partendo con l'elezione della testa, il nodo con l'ID più piccolo; a questo punto però la testa v recupera le informazioni sul suo vicinato tramite messaggi di controllo e costruisce un insieme di copertura a 3 oppure 2.5 hop, ovvero l'insieme di tutte le altre teste distanti fino a tre hop oppure fino a due hop, ma includendo anche quelle che hanno alcuni membri del loro cluster compresi tra vicini di v distanti fino a due hop. La selezione dei gateway quindi procede seguendo un'euristica che predilige i nodi vicini che raggiungono il maggior numero di teste e si arresta quando tutti gli appartenenti all'insieme di copertura sono stati raggiunti e quindi eliminati. Effettuare il broadcast sulla struttura a cluster così ottenuta è piuttosto semplice: ogni nodo che riceve un messaggio di broadcast lo ritrasmette se è una testa oppure un gateway e il messaggio non è una replica, altrimenti non lo ritrasmette.

2.4.3 Metodo 3 basato sui cluster (backbone SD-CDS)

Gli stessi autori, in [22], propongono anche un approccio SD-CDS, che dovrebbe permettere di ridurre ulteriormente il numero di ritrasmissioni. Questo metodo condivide con il precedente la tecnica di scelta delle teste dei cluster. La selezione dei gateway invece avviene dinamicamente, ogni volta che una testa v di un cluster riceve un messaggio di broadcast: all'interno del messaggio deve essere presente l'insieme di copertura (a 3 o 2.5 hop) della testa da cui si riceve il messaggio e l'insieme di gateway scelti da quest'ultima e grazie a queste informazioni v è in grado di calcolare quali sono i gateway necessari per inoltrare il messaggio alle rimanenti teste del suo insieme di copertura. La procedura di broadcast dunque varia in parte rispetto al caso SI-CDS: quando una testa riceve un messaggio di broadcast per la prima volta calcola dinamicamente l'insieme di nodi gateway e ritrasmette il messaggio, aggiungendo il suo insieme di copertura e la lista dei suoi vicini che devono a loro volta ritrasmettere; ogni altro nodo ritrasmette il messaggio solo su richiesta, controllando la lista di gateway all'interno del messaggio.

2.5 Metodi basati su alberi

Questa categoria comprende quei metodi che creano e mantengono un albero di qualche genere al fine di gestire l'inoltro di messaggi all'interno di una rete. Questa tecnica è soprattutto utilizzata nelle reti cablate, mentre è decisamente meno diffusa nelle reti wireless, soprattutto quando entra in gioco la mobilità dei nodi. Tuttavia i ricercatori hanno fatto alcune proposte, soprattutto per affrontare il problema del multicast. Di seguito ne sono presentate alcune esplicitamente pensate per il broadcast all'interno di una MANET.

2.5.1 TreeCast

Questo metodo, proposto in [13], si basa sull'idea di costruire e mantenere uno spanning tree, ovvero un albero che permette di coprire l'intera rete, da usare per effettuare il broadcast dei messaggi. L'algoritmo utilizzato, a detta degli autori, è in grado di ridurre notevolmente l'overhead richiesto per costruire e mantenere l'albero e anche di tenere in considerazione la mobilità dei nodi. Ogni nodo deve avere un suo ID univoco (nID) e deve appartenere a un singolo albero di broadcast, anch'esso identificato da un ID (tID). La creazione e il mantenimento dell'albero avvengono tramite lo scambio di appositi messaggi di controllo, che permettono anche di gestire l'unione di due alberi, la rottura di un albero in due parti e di evitare che si formino dei cicli. Il broadcast dei messaggi, una volta costruito l'albero, si appoggia proprio ai suoi rami: ogni messaggio deve contenere nID e tID relativi al mittente e il nodo che lo riceve lo ritrasmette solo se non è una foglia dell'albero e il tID contenuto nel messaggio combacia con il suo. Si noti che gli autori propongono anche una serie di estensioni atte a ridurre l'overhead, migliorare la stabilità dell'albero e gestire correttamente la mobilità dei nodi.

2.5.2 Metodo collision-free basato su alberi

In [42] viene proposto un algoritmo per la costruzione di un albero di broadcast, con radice nel nodo sorgente del messaggio, che garantisca un broadcast

con ridondanza ridotta, elevata copertura della rete e assenza di collisioni; tale algoritmo è pensato anche per risolvere i problemi di raggiungibilità che affliggono una soluzione simile [6]. L'algoritmo in considerazione costruisce un albero assegnando a ogni nodo n della rete (sorgente del messaggio esclusa) un padre, responsabile della ritrasmissione del messaggio verso n (il risultato è un'approssimazione di un CDS minimo); le ritrasmissioni vengono inoltre programmate, seguendo una strategia greedy, per evitare possibili collisioni. Si noti che l'implementazione distribuita dell'algoritmo richiede la conoscenza dei nodi vicini fino a due hop di distanza, ottenibile tramite appositi messaggi di controllo, e inoltre due complete ricerche in ampiezza sull'intera rete.

2.6 Altri metodi

In questa categoria introduciamo brevemente due tecniche adattive che sfruttano dei meccanismi differenti rispetto a tutti i metodi visti finora.

2.6.1 Pure machine learning

Questo primo metodo, presentato in [4], considera la decisione di ritrasmissione di un messaggio di broadcast come se fosse un problema di classificazione binaria. L'idea generale è consentire ai nodi di imparare dall'esperienza e adattarsi alla topologia locale. Per fare ciò è necessaria una funzione che sia in grado di indicare quando la ritrasmissione di un nodo sia utile o meno. Questa funzione può prendere in ingresso informazioni quali il numero di vicini diretti, il numero di messaggi duplicati, la velocità del nodo e altre ancora e i suoi parametri vengono ricavati mediante un addestramento. Una ritrasmissione è considerata utile se in seguito viene ricevuto lo stesso messaggio da almeno uno dei propri vicini. Ogni nodo, di fronte a una trasmissione inutile e basandosi su alcune ipotesi, è in grado di adattarsi alla situazione.

2.6.2 Inter-protocol learning

In [4] viene anche proposta una tecnica, basata sempre sull'apprendimento automatico, che permette a un nodo di adattarsi alle condizioni della rete non modificando i parametri di un singolo protocollo, bensì permettendogli di scegliere il protocollo più adatto alla situazione. L'autore propone che un nodo utilizzi SBA e AHBP a seconda della situazione, ma si potrebbero scegliere anche altri protocolli. Per decidere quale protocollo utilizzare si ricorre a un semplice albero di decisione, in cui viene presa in considerazione una variabile alla volta, che è stato addestrato grazie ai risultati comparativi di numerose simulazioni eseguite con i protocolli presi in considerazione e in varie condizioni della rete.

Capitolo 3

I metodi scelti

Il gran numero di tecniche introdotte nel capitolo precedente dimostra, in maniera piuttosto evidente, la rilevanza del problema considerato e come questo possa essere affrontato sfruttando idee, informazioni e tecnologie differenti. La prima parte di questo capitolo riguarda la necessaria scelta che deve essere compiuta per ottenere i metodi da sottoporre alle prove comparative, tenendo conto che scegliamo Opportunistic Flooder a priori, in quanto è la tecnica da noi proposta e che si vuole confrontare con altre; a questa semplice motivazione possiamo anche aggiungere che Opportunistic Flooder sia obiettivamente interessante, in quanto risulta essere una tecnica particolarmente semplice e allo stesso tempo efficiente, nonché pensata per operare in contesti mobili. Al fine di condurre una comparazione chiara e non dispersiva si è deciso di fissare il numero dei metodi a cinque, Opportunistic Flooder compreso.

Per selezionare i metodi da utilizzare nella comparazione, prendiamo innanzitutto in considerazione una serie di fattori su cui basare le motivazioni da utilizzare per escludere un certo numero di tecniche. In particolare soffermiamoci su:

- necessità tecnologiche, in particolare quelle riguardanti il posizionamento dei nodi (GPS);
- comparabilità dei metodi;

- obiettivi principali delle tecniche (efficienza, robustezza, ecc);
- caratteristiche della rete (per esempio mobilità);
- presenza di più versioni della stessa tecnica.

Iniziamo quindi una prima scrematura, utile per poi concentrarsi su un numero inferiore di tecniche e infine giungere, attraverso un'ulteriore scelta, al numero desiderato di metodi.

Eliminazione dei metodi che richiedono informazioni sulla posizione dei nodi. Su nodi semplici come quelli utilizzati nelle WSN non è generalmente disponibile la conoscenza riguardo la posizione dei nodi, per la quale è di solito necessario ricorrere a dispositivi GPS aggiuntivi. Per questo motivo eliminiamo le seguenti tecniche:

- metodo basato sulla posizione;
- metodo basato sulla posizione (adattivo);
- Flooding based on One-hop Neighbor Information and Adaptive Holding (FONIAH);
- Efficient Flooding Scheme Based on 1-hop Information.

Eliminazione delle tecniche radicalmente differenti dalle altre. Poiché l'obiettivo è selezionare un numero ristretto di metodi da confrontare, scegliamo di rivolgere l'attenzione a quelli che sfruttano tecniche quantomeno comparabili. Escludiamo dunque le due tecniche, decisamente particolari, che nel precedente capitolo erano state inserite in una categoria a parte:

- pure machine learning;
- inter-protocol learning.

Eliminazione dei metodi il cui obiettivo primario è l'affidabilità. In alcuni ambiti è di primaria importanza che i messaggi inviati in broadcast raggiungano il maggior numero possibile di nodi, ma nel nostro caso ciò non è l'obiettivo primario poiché siamo maggiormente interessati all'efficienza. Dunque eliminiamo l'unica tecnica spiccatamente votata alla robustezza:

- Robust Broadcast Algorithm.

Eliminazione dei metodi che devono mantenere una struttura globale. Visto che l'ambito di nostro interesse comprende anche situazioni caratterizzate dalla mobilità dei nodi, riteniamo che il mantenimento di una struttura globale, che sia sotto forma di cluster o di albero, non sia particolarmente adatta. Da ciò deriva lo scarto di tutti i metodi basati su cluster o alberi e di un metodo facente parte della categoria di quelli basati sulla conoscenza dei vicini:

- metodo basato sui nodi interni;
- metodo 1 basato sui cluster (SI-CDS);
- metodo 2 basato sui cluster (backbone SI-CDS);
- metodo 3 basato sui cluster (backbone SD-CDS);
- TreeCast;
- metodo collision-free basato su alberi.

Eliminazione per preferenza tra versioni simili. La mobilità dei nodi influisce direttamente sulle caratteristiche di una rete, rendendola di fatto dinamica. Sono preferibili dunque i metodi che sono in grado di adattarsi alle condizioni locali della rete, come ad esempio la densità dei nodi, nonché alla mobilità stessa. Tuttavia non sempre questi metodi adattivi sono raccomandabili, in quanto richiedono uno scambio aggiuntivo di messaggi oppure complicano ulteriormente la tecnica. Per semplicità si elencano tutti i metodi scartati e quindi si giustifica ogni scelta:

- Gossip;
- Gossip2 (basato sul numero dei vicini);
- metodo basato sul contatore (adattivo);
- Scalable Broadcast Algorithm adattivo (SBA adattivo);

- Ad Hoc Broadcast Protocol adattivo (AHBP adattivo);
- metodo basato sulla copertuira dei vicini.

Si noti che sono stati scartati sia Gossip sia Gossip2 in quanto Gossip è un metodo non adattivo, chiaramente poco adatto a reti dinamiche, mentre Gossip2 richiede la conoscenza del numero dei vicini, ma poi sfrutta tale informazione in maniera limitata. A entrambi questi due metodi si preferisce Gossip3. Per quanto riguarda il metodo basato sul contatore, la versione adattiva offre prestazioni interessanti, ma si ritiene che l'aggiunta dei messaggi necessari a ottenere l'informazione sul numero dei vicini porti ad alcune complicazioni, come ad esempio scegliere il periodo con cui questi messaggi devono essere inviati, nonché a un possibile overhead elevato, e tutto ciò contrasta con l'implicita semplicità che contraddistingue il funzionamento di questa tecnica; per questo motivo si è scelto di scartare la versione adattiva e tenere quella base. È stato inoltre deciso di eliminare la versione adattiva sia di SBA sia di AHBP: nel primo caso poiché l'adattività riguarda situazioni in cui il traffico è estremamente elevato e i ritardi di ritrasmissione sono molto ristretti e queste condizioni della rete non ci interessano particolarmente; nel secondo in quanto la versione adattiva è colpita da un aumento eccessivo dell'overhead, come affermato anche in [49]. Per la stessa ragione per cui eliminiamo la versione adattiva di AHBP, infine scartiamo anche il metodo basato sulla copertura dei vicini, che è praticamente equivalente a SBA, a parte l'adattività del periodo di emissione dei messaggi di controllo.

A questo punto il nostro interesse si focalizza su un certo numero di tecniche appartenenti a sole tre categorie delle sei presentate nel capitolo 2, ovvero metodi basati sulle probabilità, sull'area aggiuntiva coperta e sulla conoscenza dei vicini. Di seguito è presente la seconda e ultima fase del processo di scelta, da cui infine emergono i cinque metodi che prenderemo in considerazione nel prosieguo del lavoro.

Scelta nella categoria dei metodi basati sulle probabilità. Dopo la scrematura sono rimasti i seguenti metodi:

- Gossip3 (basato sul numero di repliche ricevute);

- Smart Gossip.

Potrebbe essere interessante prendere in considerazione Smart Gossip, soprattutto poiché si distingue per alcune importanti caratteristiche, tra le quali spicca un setup piuttosto semplice, piuttosto che Gossip3. Tuttavia, visto che è nostro obiettivo scegliere alcuni metodi da confrontare con Opportunistic Flooder, preferiamo concentrarci esclusivamente sulla categoria a cui esso appartiene, ovvero quella dei metodi basati sull'area aggiuntiva coperta, e sulla categoria di cui fanno parte le tecniche basate sulla conoscenza dei vicini, che dovrebbero offrire buone prestazioni e interessanti spunti di confronto. Per questo motivo non scegliamo né Gossip3 né Smart Gossip.

Scelta nella categoria dei metodi basati sull'area aggiuntiva coperta. I metodi ancora presenti in questa categoria sono i seguenti:

- metodo basato sul contatore;
- metodo basato sulla distanza;
- metodo basato sulla distanza e sul numero di hop;
- Joint Distance-Counter Threshold Broadcast Algorithm (JDCT).

Oltre a Opportunistic Flooder, già selezionato a priori, scegliamo anche il metodo basato sulla distanza e JDCT: il primo sfrutta le stesse informazioni di Opportunistic Flooder, ma in maniera differente, e possiamo quindi considerarlo una base di riferimento, mentre il secondo condivide alcune similarità con Opportunistic Flooder e promette una buona efficienza, combinando un maggior numero di informazioni.

Scelta nella categoria dei metodi basati sulla conoscenza dei vicini. Questo è senza dubbio il gruppo di metodi più nutrito (dopo la prima scrematura ne contiene ancora undici). Per sfoltirlo ulteriormente prima della selezione finale ci basiamo su un ragionamento simile a quello svolto in [49]: eliminiamo quindi i metodi self pruning e LENWB poiché concettualmente simili a SBA, ma il primo è sicuramente meno efficiente, mentre il secondo sembra avere scarse prestazioni in presenza di mobilità; inoltre scartiamo

dominant pruning, multipoint relaying e il metodo basato sui CDS siccome AHBP sfrutta un algoritmo più efficiente rispetto al primo, è molto simile ma al contempo utilizza qualche informazione in più rispetto al multipoint relaying e infine include una semplice estensione per la mobilità, che invece non è presente nel metodo basato sui CDS. Rimangono:

- Irrigator protocol;
- Fireworks;
- Delayed Flooding with Cumulative Neighbourhood (DFCN);
- Scalable Broadcast Algorithm (SBA);
- Ad Hoc Broadcast Protocol (AHBP);
- Ring Algorithm.

Tra questi decidiamo di eliminare l'Irrigator protocol e Fireworks poiché tendono a prediligere un'alta copertura della rete a discapito del numero di messaggi inviati e affidano al caso la scelta dei vicini per la ritrasmissione dei messaggi. Tra i rimanenti quattro metodi decidiamo di far cadere la scelta su SBA e AHBP poiché, come ben documentato in letteratura, si dimostrano efficienti in assenza di mobilità e riescono a ottenere risultati soddisfacenti anche nei casi in cui la rete è dinamica. Inoltre entrambi questi metodi sfruttano la conoscenza dei vicini fino a 2 hop di distanza, ma lo fanno in maniera differente e si ritiene sia interessante confrontarli approfonditamente.

A questo punto possiamo elencare i cinque metodi selezionati, affiancando a ciascuno di essi, tra parentesi, il nome che per ragioni di brevità e conformità reciproca viene usato nella restante parte del lavoro:

- metodo basato sulla distanza (*DistFlooder*);
- Opportunistic Flooder (*OppFlooder*);
- Joint Distance-Counter Threshold Broadcast Algorithm (*JDCTFlooder*);

- Scalable Broadcast Algorithm (*SBAFlooder*);
- Ad Hoc Broadcast Protocol (*AHBPFlooder*).

La restante parte di questo capitolo contiene le descrizioni dettagliate dei cinque metodi scelti e una sezione dedicata a una serie di considerazioni basate sulle similarità che li accomunano e le differenze che li distinguono.

3.1 OppFlooder

OppFlooder è brevemente accennato in [5] e l'idea fornita è quella di una tecnica decisamente semplice e allo stesso tempo efficiente, che si basa sulle stesse informazioni necessarie per il funzionamento di DistFlooder, ovvero la distanza dai nodi da cui viene ricevuto il generico messaggio di broadcast. Rispetto alla breve descrizione del funzionamento presente nel riferimento in letteratura, aggiungiamo una soglia sulla distanza, che però assolve un compito leggermente differente di quella vista in DistFlooder: viene usata per impedire eventuali ritrasmissioni di messaggi giunti da nodi molto vicini, che dunque apporterebbero una copertura aggiuntiva molto limitata e con grande probabilità risulterebbero inutili.

Simboli utilizzati e loro significati

- x: un generico nodo della rete;
- m: un generico messaggio di broadcast;
- d: distanza calcolata dal nodo da cui viene ricevuto m;
- D: valore di soglia per la distanza, sotto cui m non deve essere ritrasmesso;
- T_{max}: il massimo ritardo previsto per una generica ritrasmissione.

Funzionamento dettagliato

Il nodo sorgente del messaggio di broadcast scarta semplicemente tutte le repliche ricevute. Il funzionamento dettagliato degli altri nodi invece è questo:

1. x, alla prima ricezione di m, calcola d; se $d < D$ allora salta direttamente al passo 4, in quanto la ritrasmissione non è necessaria;
2. x si mette in attesa per un periodo di tempo calcolato in base al valore di d (più grande per valori piccoli di d e viceversa) e compreso tra 0 e T_{max}; se durante quest'attesa viene ricevuto nuovamente m allora x salta al passo 4;
3. x ritrasmette m e salta al passo 5;
4. x cancella la ritrasmissione di m;
5. x scarta tutte le repliche di m che riceve da questo momento in poi.

Parametri necessari

OppFlooder necessiterebbe di un solo parametro, ossia T_{max}, da scegliere in base alle necessità rispetto alla latenza della rete, non dimenticando però che la sua scelta influisce sulla probabilità che avvengano collisioni. La soglia D introdotta è un parametro aggiuntivo, la cui scelta però è piuttosto semplice: deve essere un valore piccolo rispetto al raggio di trasmissione della radio utilizzata: a grandi linee si può scegliere un valore pari a circa 1/10 del raggio di trasmissione.

3.2 DistFlooder

Questo metodo è il più semplice tra quelli scelti ed è stato presentato in [25] proprio per cercare di arginare il problema del broadcast storm, facendo uso di una conoscenza molto limitata. Per essere più precisi, questo metodo basa il suo funzionamento solamente sulla distanza dai nodi da cui viene ricevuto un messaggio di broadcast, valore ottenibile misurando la potenza con cui viene ricevuto il messaggio e conoscendo la potenza con cui è stato inviato (che, in genere, è determinata a priori), oppure facendo più semplicemente riferimento al valore del RSSI.

Simboli utilizzati e loro significati

- x: un generico nodo della rete;

- m: un generico messaggio di broadcast;
- d: distanza calcolata dal nodo da cui viene ricevuto m;
- d_{min}: distanza dal nodo più vicino da cui è stato ricevuto m;
- D: valore di soglia per la distanza, sotto cui m non deve essere ritrasmesso;
- T_{max}: il massimo ritardo previsto per una generica ritrasmissione.

Funzionamento dettagliato

Il nodo sorgente di un generico messaggio di broadcast si comporta molto semplicemente: dopo l'invio di m scarta ogni sua replica ricevuta. Tutti gli altri nodi invece funzionano nel seguente modo:

1. x, alla prima ricezione di m, inizializza d_{min} con il valore d appena calcolato; se $d_{min} < D$ allora salta al passo 5, in quanto la ritrasmissione non è necessaria;
2. x si mette in attesa per un periodo di tempo casuale, compreso tra 0 e T_{max}; se durante quest'attesa viene ricevuto nuovamente m allora x interrompe temporaneamente l'attesa e salta al passo 4;
3. x ritrasmette m e salta al passo 6;
4. se il valore d appena calcolato è minore di d_{min} allora x aggiorna d_{min} assegnandogli il valore di d; se $d_{min} \geq D$ ritorna al passo 2, rimettendosi in attesa dal punto in cui questa era stata interrotta (ovvero facendo ripartire il timer dal punto in cui si era fermato);
5. x cancella la ritrasmissione di m;
6. x scarta tutte le repliche di m che riceve da questo momento in poi.

Parametri necessari

DistFlooder richiede due parametri per funzionare, ovvero D e T_{max}. Il primo deve essere calcolato sperimentalmente, in quanto nella sua scelta bisogna

tener conto almeno della radio utilizzata e quindi del massimo raggio di trasmissione; il secondo invece può essere stabilito esattamente come per OppFlooder.

3.3 JDCTFlooder

JDCTFlooder [17] è una tecnica che sfrutta un quantitativo di informazioni leggermente superiore ai primi due approcci presentati e si propone di offrire un'efficienza superiore rispetto a DistFlooder senza causare un degrado dell'efficacia. Oltre alla distanza dai nodi da cui viene ricevuto un messaggio di broadcast, JDCTFlooder sfrutta anche il numero di repliche ricevute per effettuare la scelta sulla ritrasmissione. Visto che la spiegazione disponibile è in parte poco chiara, la descrizione dettagliata del funzionamento di questo metodo comprende anche alcune aggiunte dettate dal ragionamento e dal buon senso.

Simboli utilizzati e loro significati

- x: un generico nodo della rete;
- m: un generico messaggio di broadcast;
- d: distanza calcolata dal nodo da cui viene ricevuto m;
- d_{min}: distanza dal nodo più vicino da cui è stato ricevuto m;
- D: valore di soglia per la distanza, sotto cui m non deve essere ritrasmesso;
- c: contatore del numero di repliche di m;
- C: valore di soglia per il numero di repliche, oltre cui m non deve essere ritrasmesso;
- T_{max}: il massimo ritardo previsto per una generica ritrasmissione.

Funzionamento dettagliato

Così come nei due metodi precedenti, anche in questo caso il nodo sorgente,

dopo l'invio di m, scarta tutte le sue repliche ricevute. Gli altri nodi seguono invece questi passi:

1. x, alla prima ricezione di m, pone $c = 1$, calcola d e inizializza d_{min} con questo valore; se $d_{min} \leq D$ e $c \geq C$ allora salta al passo 6, in quanto la ritrasmissione non è necessaria (di solito ciò non capita, in quanto si sceglie $C > 1$);

2. x si mette in attesa per un periodo di tempo in funzione di d_{min} (esattamente come in OppFlooder, più grande per valori piccoli di d_{min} e viceversa); se durante quest'attesa viene ricevuto nuovamente m allora interrompe temporaneamente l'attesa e salta al passo 4;

3. x ritrasmette m e salta al passo 7;

4. x incrementa c di 1, calcola la nuova distanza d e aggiorna d_{min} se $d < d_{min}$; se $d_{min} > D$ ritorna al passo 2, rimettendosi in attesa dal punto in cui questa era stata interrotta (ovvero facendo ripartire il timer dal punto in cui si era fermato), aggiornando il tempo di attesa complessivo basandosi sul nuovo valore di d_{min}, quando necessario;

5. x verifica se $c < C$ e in tal caso ritorna al passo 2, rimettendosi in attesa dal punto in cui questa era stata interrotta (ovvero facendo ripartire il timer dal punto in cui si era fermato), aggiornando il tempo di attesa complessivo basandosi sul nuovo valore di d_{min}, quando necessario;

6. x cancella la ritrasmissione di m;

7. x scarta tutte le repliche di m che riceve da questo momento in poi.

Parametri necessari

JDCTFlooder ha bisogno degli stessi parametri di DistFlooder (T_{max} e D) più uno aggiuntivo, ovvero C. In questo caso risulta necessario eseguire alcuni test sperimentali per ricavare la coppia (D,C) di valori più adatta alle proprie necessità.

3.4 SBAFlooder

SBAFlooder [29] è il primo dei due metodi considerati che sfruttano la conoscenza dei nodi vicini per decidere se ritrasmettere o meno un messaggio di broadcast. In particolare SBAFlooder si basa sulla semplice idea che se tutti i nodi vicini hanno già ricevuto il messaggio allora non è più necessario ritrasmetterlo. Il funzionamento di questo metodo può essere diviso in due parti distinte:

1. la creazione e aggiornamento della topologia locale, grazie a cui ogni nodo raccoglie le informazioni relative ai suoi vicini fino a 2 hop di distanza;

2. la ritrasmissione dei messaggi, che sfrutta le informazioni sulla topologia locale più la conoscenza del nodo che ha inviato il messaggio.

Si noti che il meccanismo per la creazione e il mantenimento della topologia locale non è fissato, ma è pratica comune affidarsi a opportuni messaggi di controllo, che i nodi trasmettono periodicamente a tutti gli altri nodi direttamente raggiungibili.

Simboli utilizzati e loro significati

- x, y, z: generici nodi della rete;
- m: un generico messaggio di broadcast;
- m_c: un generico messaggio di controllo;
- V: struttura dati usata da x per contenere i propri vicini diretti e la lista dei loro vicini;
- V_x: insieme del vicini diretti di x;
- d_x: numero dei vicini diretti di x
- D_x: valore massimo tra il numero di vicini dei vicini diretti di x;
- T_c: periodo di tempo con cui viene trasmesso m_c;

- T_t: periodo di tempo dopo cui x, se non riceve alcun m_c da un nodo vicino y, cancella y da V_x; ovviamente $T_t > T_c$;
- C_m: insieme di nodi già coperti per m;
- T: valore calcolato per il tempo di attesa prima di una generica ritrasmissione;
- T_{max}: il massimo ritardo previsto per una generica ritrasmissione.

Funzionamento dettagliato: creazione e aggiornamento della topologia locale

- x, con un periodo T_c, trasmette m_c; m_c contiene il suo identificativo univoco e gli identificativi univoci di tutti i vicini diretti di cui x è a conoscenza.
- x, a ogni ricezione di m_c, inserisce il mittente di m_c e i suoi vicini in V, creando un nuovo record oppure aggiornando quello già presente; in ogni caso x salva il timestamp relativo alla ricezione di m_c.
- x, in concomitanza con l'invio di m_c, verifica se in V sono presenti vicini da cui non è stato ricevuto m_c per un periodo maggiore o uguale a T_t e in tal caso li cancella dalla lista.
- x, ad ogni modifica della topologia locale, aggiorna i valori di d_x e D_x.

Funzionamento dettagliato: diffusione dei messaggi di broadcast
Il nodo sorgente di m si comporta sempre allo stesso modo, ovvero scarta tutte le repliche ricevute di m. I nodi che invece ricevono tale messaggio operano in questo modo:

1. x, alla prima ricezione di m, verifica se $V_x \subseteq V_y \cup y$, dove y è il mittente di m e in tal caso salta al passo 6; altrimenti inizializza $C_m = V_y \cup y$ (si noti che se $y \notin V_x$ allora $V_y = \emptyset$);
2. x calcola $T = \frac{1+D_x}{1+d_x} \times T_{max}/2$; se $T > T_{max}$ allora $T = T_{max}$;

3. x si mette in attesa per un periodo di tempo casuale nell'intervallo $[0, T]$; se durante quest'attesa viene ricevuto nuovamente m allora interrompe temporaneamente l'attesa e salta al passo 5;

4. x ritrasmette m e salta al passo 7;

5. x aggiorna C_m, ovvero $C_m = C_m \cup V_z \cup z$ dove z è il mittente della replica di m (se $z \notin V_x$ allora $V_z = \emptyset$); se $V_x \supset C_m$ ritorna al passo 3, rimettendosi in attesa dal punto in cui questa era stata interrotta (ovvero facendo ripartire il timer dal punto in cui si era fermato);

6. x cancella la ritrasmissione di m;

7. x scarta tutte le repliche di m che riceve da questo momento in poi.

Si noti che il calcolo del tempo di attesa T non è esattamente quello proposto in letteratura: in particolare si è scelto un valore di ritardo costante pari a $T_{max}/2$ (in [29] è un piccolo ma non definito Δ) e di imporre $T \leq T_{max}$.

Parametri necessari

SBAFlooder richiede di assegnare un valore a tre parametri: T_c, T_t e T_{max}. Il primo dipende principalmente dalla dinamicità della rete: una rete in cui i nodi possono muoversi richiede indubbiamente un aggiornamento più frequente delle topologie locali, al fine di non basare le decisioni sulle ritrasmissioni dei messaggi su informazioni obsolete. Il secondo parametro è strettamente legato al primo e i valori per entrambi vanno ricavati sperimentalmente, al fine di bilanciare affidabilità delle informazioni topologiche e traffico richiesto per ottenerle. Per quanto riguarda il terzo parametro anche in questo caso si possono ripetere le considerazioni valide per i primi tre metodi.

3.5 AHBPFlooder

AHBPFlooder [30] è il secondo metodo che sfrutta le informazioni sul vicinato in maniera diretta, ma al contrario di SBA non è il nodo che riceve un messaggio di broadcast a scegliere se ritrasmetterlo o meno, poiché questa decisione viene presa dal mittente del messaggio che, all'atto di trasmettere,

basandosi sulla sua conoscenza dei suoi vicini e dei vicini di questi ultimi, sceglie un insieme dei suoi vicini che permetta a tutti i nodi distanti 2 hop di ricevere il messaggio. Anche in questo caso il funzionamento può essere suddiviso in due parti: la prima è necessaria affinché tutti i nodi abbiano a disposizione informazioni aggiornate sulla topologia locale della rete, ovvero conoscano tutti i nodi fino a 2 hop di distanza; la seconda invece è quella relativa alla ritrasmissione vera e propria dei messaggi di broadcast, che si basa sulla scelta del nodo da cui viene ricevuto il messaggio, sulle informazioni relative alla topologia locale e sul percorso già attraversato da tale messaggio. Si noti che, essendo interessati anche a situazioni in cui è presente la mobilità dei nodi, AHBPFlooder corrisponde alla versione estesa di AHBP, ovvero AHBP-EX. Così come in SBAFlooder, per la creazione e l'aggiornamento delle informazioni topologiche si ricorre a messaggi di controllo appropriati, che ogni nodo deve trasmettere periodicamente. Si noti che questa operazione è assolutamente identica a quella usata in SBAFlooder, fatta eccezione per la mancanza dell'aggiornamento del valore di due variabili.

Simboli utilizzati e loro significati

- x, y: generici nodi della rete;
- m: un generico messaggio di broadcast;
- m_c: un generico messaggio di controllo;
- V: struttura dati usata da x per contenere i propri vicini diretti e la lista dei loro vicini;
- V_x: insieme del vicini diretti di x;
- T_c: periodo di tempo con cui viene trasmesso m_c;
- T_t: periodo di tempo dopo cui x, se non riceve alcun m_c da un nodo vicino y, cancella y da V_x; ovviamente $T_t > T_c$;
- BRG_x: insieme di nodi scelti da x per la ritrasmissione di m (Broadcast Relay Gateway);

- P: insieme dei nodi già attraversati da m;
- G: il grafo della topologia locale, contenente nodi e archi;
- T_{max}: il massimo ritardo previsto per una generica ritrasmissione.

Funzionamento dettagliato: creazione e aggiornamento della topologia locale

- x, con un periodo T_c, trasmette m_c; m_c contiene il suo identificativo univoco e gli identificativi univoci di tutti i vicini diretti di cui x è a conoscenza.
- x, a ogni ricezione di m_c, inserisce il mittente di m_c e i suoi vicini in V, creando un nuovo record oppure aggiornando quello già presente; in ogni caso x salva il timestamp relativo alla ricezione di m_c.
- x, in concomitanza con l'invio di m_c, verifica se in V sono presenti vicini da cui non è stato ricevuto m_c per un periodo maggiore o uguale a T_t e in tal caso li cancella dalla lista.

Funzionamento dettagliato: diffusione dei messaggi di broadcast
Tutti i nodi che trasmettono un messaggio di broadcast, o perché sono la sorgente stessa del messaggio oppure perché scelti da qualche altro nodo, devono scegliere a quali nodi, tra i loro vicini, affidare il compito di ritrasmettere e operano in questo modo (il nodo sorgente ovviamente parte dal punto 2 invece che dal punto 1):

1. x, alla prima ricezione di m, controlla BRG_y contenuto in m, dove y è mittente di m; se il suo identificativo non è presente e $y \in V_x$ allora salta al passo 6;
2. x inizializza $BRG_x = \emptyset$ e costruisce G partendo dalle informazioni contenute in V;
3. x riduce G applicando le seguenti regole:
    a. se $y \in P$ elimina y, tutti i suoi vicini e tutti gli archi associati;

   b. se entrambi gli estremi di un arco sono vicini di x, elimina l'arco;

   c. se y è rimasto senza vicini (è un nodo isolato), lo elimina.

4. siano G_1 i nodi di G che sono vicini diretti di x, G_2 quelli a 2 hop di distanza; x riempie BRG_x ripetendo le seguenti operazioni:

   a. se $G_2 = \emptyset$ salta al passo 5, altrimenti sceglie w = nodo di G_1 che è unico vicino di un nodo di G_2 (se presente) oppure w = nodo di G_2 con il maggior numero di vicini;

   b. aggiorna BRG_x, ovvero $BRG_x = BRG_x \cup w$ e da G rimuove w, i suoi vicini e tutti gli archi associati.

5. x inserisce BRG_x in m al posto di BRG_y, aggiunge il suo identificativo a P, ritrasmette m dopo un ritardo casuale compreso tra 0 e T_{max} e salta al passo 7;

6. x cancella la ritrasmissione di m;

7. x scarta tutte le repliche di m che riceve da questo momento in poi.

Si noti che, al punto 1, se x riceve un messaggio di broadcast da y e $y \notin V_x$ allora x ritrasmette sempre il messaggio. Questo è l'effetto dell'estensione usata per tener conto della mobilità.

Parametri necessari

AHBPFlooder richiede gli stessi identici parametri di SBAFlooder, ovvero T_c, T_t e T_{max}. La scelta di T_c e T_t, da eseguire sempre per via sperimentale, deve far riferimento alle stesse considerazioni fatte nel caso di SBAFlooder. Si noti invece che T_{max} in questo caso assume un ruolo ben diverso rispetto a quello ricoperto sia in SBAFlooder sia in tutti gli altri metodi scelti: il ritardo di ritrasmissione non è più un'attesa utile a ricevere repliche su cui basare la decisione di ritrasmissione, ma è semplicemente un modo per ridurre la probabilità che avvengano trasmissioni simultanee e quindi possibili collisioni. Per questo motivo è sensato scegliere T_{max} più piccolo rispetto a tutti gli altri metodi.

3.6 Considerazioni riguardo i metodi scelti

In quest'ultimo paragrafo sono raccolte alcune considerazioni riguardanti i cinque metodi scelti e le due categorie a cui appartengono. In particolare si ritiene utile soffermarsi, seppur brevemente, sui punti di forza e le debolezze che contraddistinguono le due classi di metodi considerate; inoltre è interessante prendere in esame similarità e differenze presenti tra alcune delle tecniche scelte. Si ritiene possa essere molto interessante verificare se e in che misura le considerazioni contenute in questa sezione saranno confermate in fase sperimentale.

3.6.1 Le due categorie a confronto

Le differenze che distinguono i tre metodi appartenenti alla categoria delle tecniche basate sull'area aggiuntiva coperta dai due inclusi nell'altro gruppo sono piuttosto evidenti. Enumeriamole, in modo da poter poi compiere alcune interessanti osservazioni:

1. i primi sono caratterizzati da un funzionamento semplice, mentre gli altri richiedono operazioni più complesse, anche dal punto di vista computazionale;

2. i primi necessitano il mantenimento di uno stato molto ridotto, invece i secondi operano su uno stato più esteso, la cui dimensione dipende direttamente da alcune caratteristiche della rete (per esempio la densità dei nodi);

3. i primi si basano su limitate informazioni (distanze, numero di repliche ricevute) facilmente disponibili e tendenzialmente slegate dalle caratteristiche della rete, al contrario gli altri sfruttano una conoscenza più dettagliata e strutturata, che richiede un apposito meccanismo di acquisizione e mantenimento.

Per quanto riguarda il primo punto si può osservare che la maggiore complessità, che contraddistingue le tecniche basate sulla conoscenza dei vicini, rende l'implementazione dei metodi più difficile e soggetta a errori; inoltre

l'esecuzione del codice è computazionalmente più impegnativa. Non bisogna trascurare soprattutto quest'ultimo fatto, poiché queste tecniche vengono anche utilizzate — è il caso di nostro interesse — in reti formate da apparecchi dotati di ridotte potenze di calcolo e limitate riserve energetiche. Ovviamente bisogna anche considerare il fatto che un metodo più complesso dovrebbe essere in grado di operare scelte più accurate e quindi fornire un'efficienza superiore.

Dall'esame della seconda differenza emergono due considerazioni. La prima riguarda i requisiti in termini di memoria: le informazioni relative alla topologia locale alla base del funzionamento di SBAFlooder e AHBPFlooder, nonché alcune altre strutture dati utilizzate da questi metodi, richiedono un quantitativo di memoria maggiore rispetto al limitato stato mantenuto dalle altre tre tecniche. Nonostante in molti ambiti dell'informatica il problema della memoria sia passato in secondo piano, nel caso delle WSN il quantitativo di memoria a disposizione può essere piuttosto limitato. Inoltre si noti che la memoria richiesta per lo stato di SBAFlooder e AHBPFlooder non è un valore fisso, ma dipende per esempio dalla densità dei nodi e più in particolare dal numero massimo di vicini che un nodo si presume possa avere. Ciò è un problema in quanto, sempre nell'ambito di nostro interesse, spesso queste tecniche vengono implementate con linguaggi semplici, quali il *nesC*, che non prevedono la gestione dinamica della memoria: diventa dunque necessario individuare il cosiddetto caso pessimo e fissare un valore che lo soddisfi, facendo sempre attenzione alla scarsa disponibilità di memoria.

L'ultima differenza porta a osservare che informazioni più numerose, se usate oculatamente, dovrebbero consentire il raggiungimento di un'efficienza più elevata, permettendo di superare alcuni limiti che affliggono le tecniche appartenenti alla prima categoria: un esempio è l'incapacità di queste tecniche di evitare alcune ritrasmissioni inutili, come quelle effettuate dai nodi posti sul perimetro della rete. Tuttavia bisogna anche considerare che ottenere queste informazioni comporta un costo aggiuntivo in termini di traffico, ovvero implica un overhead maggiore. Infine può essere rilevante notare che la validità delle informazioni topologiche ottenute con un metodo semplice, ma inaffidabile, come quello usato da SBAFlooder e AHBPFlooder, può di-

pendere da svariati fattori, come ad esempio la mobilità dei nodi o possibili collisioni: l'eventuale scorrettezza o incompletezza di tali informazioni può impattare notevolmente sull'efficienza o sull'efficacia dei due metodi.

3.6.2 DistFlooder, OppFlooder e JDCTFlooder a confronto

Queste tre tecniche, come già accennato nel corso della loro descrizione, sono simili sotto molti punti di vista. Di seguito sono evidenziale le similarità e le differenze più rilevanti e i loro effetti.

DistFlooder e OppFlooder

DistFlooder e OppFlooder basano il loro funzionamento sulla stessa informazione, che viene però utilizzata diversamente: DistFlooder sfrutta la conoscenza della distanza solo per valutare quanto sia utile ritrasmettere il messaggio, mentre OppFlooder ingloba la stessa valutazione nell'idea di favorire la ritrasmissione dei nodi più lontani, in base a un tempo di ritrasmissione non più casuale ma opportunamente calcolato. Questo, unito al fatto che in OppFlooder la ricezione di una singola replica decreta la cancellazione della ritrasmissione, mentre in DistFlooder la ritrasmissione può avvenire nonostante siano state ricevute più repliche, dovrebbe concretizzarsi in una maggiore efficienza di OppFlooder.

DistFlooder e JDCTFlooder

JDCTFlooder può essere considerato un ibrido tra DistFlooder e il metodo basato sul contatore, con in aggiunta il calcolo del ritardo di ritrasmissione basato sulla distanza invece che casuale. Escludendo la componente derivata dal metodo basato sul contatore (per fare ciò è necessario imporre il parametro $C = 1$), si può notare che JDCTFlooder si comporta come DistFlooder, calcolo del ritardo di ritrasmissione a parte. Aumentando il valore del parametro C si fa in modo che JDCTFlooder tenda a ritrasmettere più messaggi rispetto a DistFlooder, favorendo così l'efficacia piuttosto che l'efficienza. Quest'ultima dovrebbe invece migliorare proprio grazie al calcolo più accorto del ritardo di ritrasmissione.

OppFlooder e JDCTFlooder

OppFlooder condivide con JDCTFlooder l'idea di favorire la ritrasmissione dei nodi più distanti. Le due tecniche però differiscono per il fatto che OppFlooder cancella la ritrasmissione alla ricezione della prima replica di un messaggio, mentre non solo JDCTFlooder permette (come DistFlooder) la ricezione di più repliche, purché da una distanza superiore al parametro soglia D, ma consente anche la ritrasmissione qualora il numero di repliche ricevute da una distanza inferiore a D sia minore del parametro soglia C. Si può quindi concludere che JDCTFlooder, a seconda della scelta dei suoi parametri, dovrebbe poter garantire, in generale, un'efficacia superiore a OppFlooder a discapito dell'efficienza.

3.6.3 SBAFlooder e AHBPFlooder a confronto

Sia SBAFlooder sia AHBPFlooder fanno affidamento sulla conoscenza che ogni nodo possiede in merito agli altri nodi distanti fino a due hop, ma sfruttano due strategie opposte per prendere le decisioni di ritrasmissione: nel primo metodo è il nodo che riceve il messaggio a decidere se la sua ritrasmissione è utile, mentre nel secondo è il nodo che invia il messaggio a scegliere un insieme di suoi vicini che devono ritrasmettere. AHBPFlooder dovrebbe quindi garantire un'efficienza maggiore, in quanto la scelta dei nodi che devono ritrasmettere è un processo in cui non compare alcuna componente probabilistica, come ad esempio la ricezione di repliche in un tempo di attesa casuale, e che assicura, almeno localmente, l'assenza di ritrasmissioni inutili. La strategia di AHBPFlooder presenta però un inconveniente: è particolarmente sensibile alla validità delle informazioni che sfrutta. Supponiamo, ad esempio, che un nodo debba ritrasmettere un messaggio e scelga i suoi BRG in base alla topologia locale a sua disposizione, ma uno dei BRG in realtà non si trovi più nel suo raggio di trasmissione: il risultato è che, in generale, una parte dei nodi a due hop di distanza non viene raggiunta dal messaggio. Invece in SBAFlooder eventuali mancate ritrasmissioni dovute all'obsolescenza delle informazioni possono essere generalmente sostituite dalla decisione di ritrasmettere presa da qualche altro nodo, ma può anche

capitare che l'inesattezza delle informazioni provochi ritrasmissioni inutili. Inoltre evidenziamo il fatto che AHBPFlooder, al contrario di SBAFlooder (e anche di tutti gli altri metodi scelti) inserisce in tutti i messaggi di broadcast un certo quantitativo di dati, di dimensione variabile, e quindi l'overhead richiesto per il suo funzionamento è più elevato. Infine osserviamo anche che AHBPFlooder è sicuramente più complesso rispetto a SBAFlooder: la scelta dei BRG richiede infatti una sequenza di operazioni piuttosto rilevante.

Capitolo 4

Ricerche di minimo teorico

La necessità di eseguire delle ricerche di minimo teorico nasce dall'idea di non comparare semplicemente tra loro alcuni metodi per il broadcasting, ma di valutarne l'effettiva efficienza mettendoli a confronto con i risultati ottenuti attraverso l'uso di un algoritmo centralizzato, in grado di trovare il miglior risultato raggiungibile in teoria.

Un semplice e valido indicatore di efficienza per il broadcast è rappresentato dal numero di ritrasmissioni necessarie a coprire l'intera rete, ovvero, in termini leggermente differenti, dal numero di nodi della rete che, a partire dalla sorgente del messaggio, devono ritrasmettere tale messaggio affinché sia ricevuto da tutti i nodi della rete. La ricerca di questo numero è nota, nella letteratura scientifica, come ricerca del Minimum Connected Dominating Set (MCDS).

Nel seguito di questo capitolo per prima cosa è presentato il problema della ricerca di un MCDS e quindi viene proposto un algoritmo capace di risolverlo, almeno nei casi di nostro interesse.

4.1 La ricerca del MCDS

Prima di affrontare il problema della ricerca di un MCDS, è utile fornire una definizione precisa di MCDS. Un MCDS è il minimo insieme S di vertici di un grafo G tale che:

1. ogni vertice in S è in grado di raggiungere un qualsiasi altro vertice in S tramite un percorso che attraversa solamente altri vertici di S;

2. qualsiasi vertice di G che non appartiene a S è adiacente a un vertice di S.

Risulta piuttosto naturale rappresentare una rete wireless sotto forma di un grafo orientato, i cui vertici sono i nodi stessi della rete e in cui è presente un arco che parte da un generico vertice u e arriva in un altro vertice v se e solo se v si trova nel raggio di trasmissione di u. Al fine di ridurre la complessità della situazione, introduciamo alcune opportune semplificazioni rispetto al caso reale:

- modelliamo il raggio di trasmissione di un nodo con un cerchio perfetto, centrato sul nodo stesso; questa è una scelta sensata se, come nel nostro caso, si intende usare radio omnidirezionali;

- stabiliamo che il raggio di trasmissione sia identico per tutti i nodi della rete, introducendo di fatto la bidirezionalità dei collegamenti.

A questo punto possiamo modellare la rete wireless con un grafo semplice (non orientato), in cui esiste un arco tra u e v se e solo se i due nodi si trovano a una distanza inferiore rispetto al loro raggio di trasmissione; in figura 4.1 è presente un esempio. Affiancando quest'ultima affermazione alla definzione di MCDS, risulta piuttosto palese che la ricerca di un MCDS è proprio ciò a cui siamo interessati, ma prima di affrontare questo problema si ritiene opportuno evidenziare un paio di particolarità a riguardo:

- in una generica rete possono essere presenti uno o più MCDS; nell'esempio di figura 4.1 ce ne sono due;

- se $n = |MCDS|$ allora n corrisponde al numero di trasmissioni necessarie per coprire l'intera rete se il nodo sorgente del messaggio appartiene a un MCDS, mentre se tale nodo non appartiene a un MCDS il numero di trasmissioni necessarie è pari a $n + 1$.

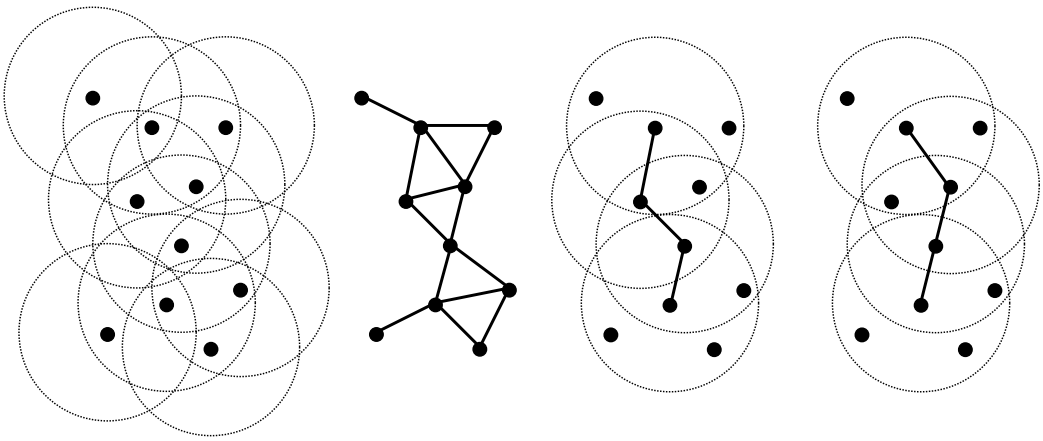

Figura 4.1: Una rete d'esempio, il grafo e i due MCDS

La ricerca di un MCDS è un problema ampiamente studiato nella letteratura ed è stato dimostrato che, nel caso sia applicato a un generico grafo, appartiene alla categoria dei problemi NP-difficili [8]. In realtà, come detto sopra, noi non siamo interessati a ricercare un MCDS su un generico grafo, ma su un tipo più particolare, chiamato *unit-disk graph* (UDG). Purtroppo la ricerca di un MCDS rimane un problema NP-difficile anche se applicata su UDG, come dimostrato in [3].

Visto che la ricerca di un MCDS risulta essere un problema di non facile soluzione, in letteratura sono stati proposti diversi algoritmi centralizzati che cercano una soluzione approssimata, ovvero un CDS; alcuni di questi sono raccolti in [1]. Purtroppo il grado di approssimazione garantito da questi algoritmi è troppo grande per i nostri scopi: per esempio, l'algoritmo di Ruan [35] restituisce un CDS tale che $|CDS| \leq (2 + ln(\Delta)) \cdot |MCDS|$, dove Δ è il massimo numero di vicini di un nodo all'interno della rete. Visto che anche in una rete a media densità di nodi è probabile che $\Delta \geq 8$, ciò che si ottiene nel migliore dei casi è un CDS la cui cardinalità è garantita essere inferiore a $4 \cdot |MCDS|$ e ciò sicuramente non è adatto ai nostri scopi. Per questo motivo è stato deciso di proporre un algoritmo che restituisce un risultato esatto e riesce ad eseguire le ricerche in un tempo accettabile, almeno per le reti di nostro interesse.

4.2 L'algoritmo di ricerca proposto: MinBroadcast

Come è stato già accennato a inizio capitolo, ciò che ci interessa è ottenere il limite inferiore sul numero di nodi che devono trasmettere un messaggio affinché questo raggiunga l'intera rete. È quindi chiaro che lo scopo dell'algoritmo non è quello di trovare tutti gli MCDS di una rete, ma quello di trovarne uno qualsiasi e, grazie a esso, sapere il minimo numero di nodi che è necessario ritrasmettano all'interno di tale rete, almeno per quanto riguarda il modello teorico presentato al paragrafo precedente. In particolare, l'algoritmo qui presentato restituisce il primo MCDS trovato, essendo noti la topologia della rete, il raggio di trasmissione dei nodi e la sorgente del messaggio; nel caso in cui il nodo sorgente non faccia parte di un MCDS restituisce il primo MCDS trovato unito alla sorgente del messaggio.

L'algoritmo proposto, chiamato *MinBroadcast*, può essere assimilato a una ricerca in profondità su un albero. Un albero, al pari di un grafo, è composto da nodi — in seguito chiamati vertici, per non confonderli con i nodi della rete — e da archi; al contrario di un grafo però, un albero è caratterizzato da una struttura gerarchica, in cui i vertici appartengono a un preciso livello e gli archi possono collegare vertici di livelli differenti. L'idea su cui si basa MinBroadcast è utilizzare un albero per rappresentare gli stati e l'evoluzione di un broadcast all'interno di una rete:

- ogni vertice rappresenta un particolare stato della rete, che è univocamente identificato dalla lista dei nodi che hanno trasmesso, ordinati secondo la sequenza di trasmissione; tutti i vertici che rappresentano uno stato in cui hanno trasmesso k nodi si trovano al livello k dell'albero;

- ogni arco corrisponde alla trasmissione da parte di un nodo e infatti permette di scendere di un livello lungo l'albero.

Il funzionamento generale dell'algoritmo proposto è il seguente: all'inizio della ricerca ci si trova nella radice dell'albero, che rappresenta lo stato in cui solo il nodo sorgente del messaggio ha trasmesso; l'algoritmo sceglie uno dei

nodi adiacenti alla sorgente affinché trasmetta a sua volta e quindi si sposta in un nuovo vertice, che si trova al secondo livello; l'algoritmo ora sceglie uno tra i nodi adiacenti ai due che hanno già trasmesso, ossia sceglie uno tra i nodi che sono stati raggiunti dal messaggio, e lo fa ritrasmettere e così via. La discesa in profondità lungo l'albero si arresta quando tutti i nodi della rete sono stati raggiunti dal messaggio, oppure se la soluzione finora trovata non è migliorabile a partire da tale punto nell'albero. Si noti che, al fine di eseguire una ricerca completa, è ovvio che l'algoritmo, in ogni vertice dell'albero, debba scegliere per la trasmissione ognuno dei nodi che hanno già ricevuto il messaggio e non l'hanno ancora ritrasmesso.

Questa essenziale presentazione del funzionamento di MinBroadcast mette in luce come l'albero su cui effettuare la ricerca venga generato sul momento, o meglio venga aggiornato a partire dallo stato precedente e in base al nodo che ha trasmesso. Inoltre è piuttosto facile notare che la ricerca, così come è stata descritta finora, rischia di generare e dover operare su alberi di dimensioni notevoli e perciò richiedere molto tempo. Per questo motivo introduciamo due importanti ottimizzazioni, che permettono di limitare considerevolmente i cammini che la ricerca deve compiere, ovviamente senza dover rinunciare all'esattezza del risultato. L'introduzione di queste ottimizzazioni ha un effetto davvero importante: dagli esperimenti eseguiti è stato rilevato che le ottimizzazioni permettono di ridurre notevolmente il tempo richiesto per una ricerca e nei casi più eclatanti di passare da centinaia di ore di calcolo a pochi secondi o al massimo a qualche minuto. Prima di presentare le ottimizzazioni però definiamo lo stato della rete, poiché sarà utile farvi riferimento nel seguito.

Lo stato della rete

Come è stato detto in precedenza, ogni vertice dell'albero rappresenta un particolare stato della rete, che è identificato in modo univoco dalla lista ordinata dei nodi che hanno trasmesso. Di seguito specifichiamo tutte le informazioni che compongono lo stato della rete, seguite da un nome da usare come riferimento:

- la lista dei nodi scoperti, ovvero di quei nodi che, non essendo adiacenti

ad alcun nodo che ha trasmesso, non sono ancora stati raggiunti dal messaggio ($nodiS$);

- la lista dei nodi utili, cioè di quei nodi che sono stati già raggiunti dal messaggio, non lo hanno ancora ritrasmesso e hanno dei vicini scoperti ($nodiU$);
- la lista dei nodi che hanno trasmesso il messaggio ($nodiT$).

Si noti che per ognuno dei nodi scoperti o utili è mantenuta la lista dei suoi vicini che sono ancora scoperti; per fare riferimento, per esempio, alla lista dei vicini di un nodo n si usa la scrittura *vicini(n)*.

4.2.1 Ottimizzazione locale: eliminazione dei nodi dominati

Questa prima ottimizzazione è detta locale, in quanto opera internamente a ogni singolo stato della rete ed è indipendente dalla rimanente parte della ricerca. Siano n e m due nodi della rete che hanno ricevuto il messaggio di broadcast e possono quindi ritrasmetterlo: n domina m se la sua ritrasmissione permette di raggiungere almeno gli stessi nodi scoperti che riceverebbero il messaggio da m. Si noti che se $nodiS(n) = nodiS(m)$ allora n nomina m e viceversa (in questo caso si parla di dominanza debole).

L'idea alla base di questa ottimizzazione è la seguente: non c'è alcuna utilità nel far trasmettere un nodo dominato, in quanto esiste un altro nodo che permette una copertura uguale o maggiore. Più precisamente, la trasmissione di un nodo dominato m è inutile poiché:

- in caso di copertura uguale, lo stato della rete in cui si giunge facendo trasmettere m è esattamente equivalente a quello ottenuto con la trasmissione di n per quanto riguarda l'insieme dei nodi ancora scoperti e quello dei nodi utili;
- in caso di copertura minore, il nuovo stato della rete generato dalla trasmissione di m non può portare a una soluzione migliore rispetto a quello raggiungibile scegliendo n per ritrasmettere.

Si noti che nel caso in cui n e m hanno gli stessi vicini scoperti è indifferente eliminare uno o l'altro ai fini della ricerca di un MCDS.

In conclusione, questa ottimizzazione viene eseguita in ogni vertice dell'albero di ricerca ed elimina tutti i nodi dominati dall'insieme dei nodi utili.

4.2.2 Ottimizzazione globale: eliminazione dei sottoalberi ridondanti

Al contrario della precedente, questa ottimizzazione è definita globale, dato che richiede la conoscenza degli stati della rete già esplorati dalla ricerca per essere applicata. Precedentemente è stato affermato che ogni stato può essere identificato attraverso la lista ordinata dei nodi che hanno trasmesso. Tenendo presente che lo scopo dell'algoritmo è quello di trovare un qualsiasi MCDS, possiamo introdurre la definizione di stato equivalente: due stati s_1, s_2 sono equivalenti se la lista dei nodi che hanno trasmesso contiene gli stessi elementi, non importa in che ordine. La spiegazione è piuttosto semplice: se in s_1 e s_2 le liste $nodiT$ contengono gli stessi nodi allora anche le liste $nodiS$ e $nodiU$ sono uguali nei due stati, rendendo di fatto s_1 e s_2 equivalenti. Partendo dalla definizione di stato equivalente possiamo facilmente giungere a quella di sottoalbero equivalente: due sottoalberi sono equivalenti se gli stati contenuti nelle loro radici sono equivalenti.

L'ottimizzazione si basa quindi sul fatto che è inutile esplorare un sottoalbero quando ne è già stato visitato uno equivalente. L'effetto di questa ottimizzazione, che deve essere applicata in ogni vertice dell'albero, è l'eliminazione dei sottoalberi ridondanti.

Si ritiene importante evidenziare che il funzionamento di questa ottimizzazione richiede il salvataggio e il mantenimento di tutti gli stati della rete generati ed esplorati dalla ricerca. Ciò può diventare un serio problema in presenza di reti particolarmente grandi, in quanto il numero di stati da salvare può crescere considerevolmente, tanto da limitare le prestazioni dell'algoritmo.

4.2.3 Descrizione dell'algoritmo

Scendiamo ora nei dettagli riguardanti l'algoritmo. MinBroadcast basa il suo funzionamento su una funzione ricorsiva e questo è piuttosto naturale in quanto si tratta di una ricerca su un albero: ciò può essere facilmente spiegato affermando che un generico sottoalbero è del tutto equivalente all'albero principale dal punto di vista strutturale e quindi può essere trattato esattamente allo stesso modo.

Al fine di presentare una descrizione compatta e sufficientemente chiara di MinBroadcast, si è deciso di ricorrere allo pseudocodice, ovvero a una rappresentazione di alto livello dell'algoritmo, che pur usando le convenzioni tipiche di un linguaggio di programmazione è pensata per la lettura e la comprensione da parte dell'uomo. Prima dello pseudocodice sono elencate tutte le funzioni in esso contenute e per ognuna di esse è fornita la lista dei parametri e una spiegazione di ciò che fa:

- *iniziaBroadcast(nodoScelto, nodiS, nodiU, nodiT)*; è la funzione ricorsiva principale: ogni volta che questa funzione viene chiamata si scende di un livello nell'albero facendo trasmettere *nodoScelto* e quindi viene aggiornato lo stato della rete e viene deciso se continuare oppure bloccare il broadcast; si noti che questa funzione deve copiare lo stato prima di aggiornarlo, per non modificare lo stato della rete nel vertice al livello superiore; si noti inoltre che ogni volta che questa funzione termina si risale di un livello, ritornando allo stato precedente la trasmissione dell'ultimo nodo.

- *aggiornaStato(nodoScelto, nodiS, nodiU, nodiT)*; questa funzione aggiorna lo stato della rete e in particolare:

  1. elimina *nodoScelto* dai nodi scoperti (ciò avviene solo nella radice dell'albero) o dai nodi utili;
  2. aggiunge *nodoScelto* ai nodi che hanno trasmesso;
  3. per ogni vicino v_1 scoperto del nodo scelto:
     a. per ogni vicino v_2 di v_1: rimuove v_1 dai vicini di v_2 se $v_1 \in nodiU$;

b. rimuove il nodo scelto dai vicini di v_1;

c. rimuove v_1 dai nodi scoperti;

d. aggiunge v_1 ai nodi utili.

- *verificaFineBroadcast(nodiS, nodiT)*; questa funzione impedisce alla ricerca di scendere ulteriormente lungo l'albero se tutti i nodi sono stati raggiunti dal messaggio e salva la soluzione se è migliore di quella attuale oppure se non è possibile migliorare quest'ultima.

- *organizzaNodiUtili(nodiU)*; questa funzione esegue le seguenti operazioni sull'insieme dei nodi utili:

  1. rimuove dai vicini di ogni nodo utile eventuali nodi che non sono più scoperti poiché sono diventati utili;
  2. rimuove da nodiU eventuali nodi che non hanno più vicini scoperti;
  3. ordina i nodi utili sulla base del numero di vicini scoperti (in ordine decrescente); questa è un'euristica che mira a far esplorare per primi i rami dell'albero che dovrebbero portare alla soluzione migliore.

- *eliminaNodiUtiliDominati(nodiS, nodiU)*; questa funzione verifica se tra i nodi utili ce n'è qualcuno dominato debolmente da un altro e lo elimina, ovvero applica l'ottimizzazione locale; gli stessi nodi devono essere eliminati anche dai vicini dei nodi scoperti.

- *verificaStatoInesplorato(nodiT, nuovoNodoScelto)*; questa funzione realizza l'ottimizzazione globale verificando se la trasmissione del nuovo nodo scelto porta la rete in uno stato che non è mai stato esplorato e in tal caso restituisce il valore *true*; la ricerca sul sottoalbero viene inibita in caso contrario (*false*).

Pseudocodice 1
Funzione iniziaBroadcast(nodoScelto, nodiS, nodiU, nodiT)

```
1: aggiornaStato(nodoScelto, nodiS, nodiU, nodiT)
2: verificaFineBroadcast(nodiS, nodiT)
3: organizzaNodiUtili(nodiU)
4: eliminaNodiUtiliDominati(nodiS, nodiU)
5: for all u ∈ nodiU do
6:    if verificaStatoInesplorato(nodiT, u) = true then
7:       iniziaBroadcast(u, nodiS, nodiU, nodiT)
8:    end if
9: end for
10: return
```

Lo pseudocodice appena presentato riguarda solamente l'algoritmo di ricerca e in particolare la funzione ricorsiva principale. Altre operazioni, come quella necessaria a creare lo stato iniziale della rete a partire dalla topologia e dal raggio di trasmissione, sono facilmente ottenibili in più di un modo e non si ritiene particolarmente utile presentarle.

Scendiamo ora a un livello di dettaglio maggiore, fornendo lo pseudocodice relativo alle funzioni descritte in precedenza e usate per rendere immediata la comprensione del funzionamento generale di MinBroadcast. Si inizia con la funzione utilizzata per aggiornare lo stato della rete.

Pseudocodice 2
Funzione aggiornaStato(nodoScelto, nodiS, nodiU, nodiT)

```
1:  vicini_nS ← vicini(nodoScelto)
2:  if nodo sorgente then
3:     nodiS ← nodiS − {nodoScelto}
4:  else
5:     nodiU ← nodiU − {nodoScelto}
6:  end if
7:  nodiT ← nodiT + {nodoScelto}
8:  for all v1 ∈ vicini_nS do
9:     vicini_v1 ← vicini(v1)
10:    for all v2 ∈ vicini_v1 do
11:       if v2 ∈ nodiU then
12:          vicini_v2 ← vicini(v2)
13:          vicini_v2 ← vicini_v2− {v1}
14:       end if
15:    end for
16:    vicini_v1 ← vicini_v1− {nodoScelto}
17:    nodiS ← nodiS − {v1}
18:    nodiU ← nodiU + {v1}
19: end for
20: return
```

Le righe 1–7 servono semplicemente a spostare il nodo che ha trasmesso dall'insieme di quelli scoperti, nel caso sia il nodo sorgente, oppure di quelli utili nell'insieme contenente i nodi che hanno trasmesso. Le righe 8–19 invece completano l'aggiornamento dello stato della rete.

Proseguiamo con lo pseudocodice della funzione che stabilisce quando fermare il broadcast. Si noti che, all'interno del codice, viene usata una variabile *soluzione* per far riferimento alla soluzione migliore trovata finora; inoltre, con la dicitura *ferma broadcast* si intende che l'algoritmo di ricerca non scende ulteriormente nell'albero.

Pseudocodice 3
Funzione verificaFineBroadcast(nodiS, nodiT)

```
1:  if nodiS = ∅ then
2:     if |NodiT| ≤ |soluzione| or soluzione = ∅ then
3:        soluzione ← nodiT
4:        ferma broadcast
5:     end if
6:  end if
7:  if |NodiT| = |soluzione|−1 then
8:     ferma broadcast
9:  end if
10: return
```

Nelle righe 1–6 il broadcast viene fermato se tutti i nodi sono stati coperti ed eventualmente viene salvata la soluzione. Le righe 7–9 bloccano invece la discesa nell'albero qualora si sia giunti a un livello pari alla cardinalità della soluzione meno uno, poiché in tal caso non è certamente più possibile migliorare la soluzione, ma al massimo si può eguagliarla.

Ecco lo pseudocodice che compie alcune operazioni sui nodi utili e li ordina.

Pseudocodice 4
Funzione organizzaNodiUtili(nodiU)

```
1:  for all u ∈ nodiU do
2:      vicini ← vicini(u)
3:      for all v ∈ vicini do
4:          if v ∈ nodiU then
5:              vicini ← vicini - {v}
6:          end if
7:      end for
8:  end for
9:  for all u ∈ nodiU do
10:     vicini ← vicini(u)
11:     if vicini = ∅ then
12:         nodiU ← nodiU − {u}
13:     end if
14: end for
15: ordina(nodiU)
16: return
```

Le righe 1–8 servono a cancellare dai vicini dei nodi utili quei nodi che non sono più scoperti, ma che ancora non sono stati cancellati. Nelle righe 9–14 vengono invece rimossi i nodi che, non avendo più alcun vicino scoperto, non sono più utili. Infine la riga 15 ordina la lista dei nodi utili.

Siamo giunti allo pseudocodice che descrive le operazioni relative alla prima ottimizzazione, cioè quella che elimina i nodi utili dominati. Si noti che con la scrittura *NodiU(i)* si intende l'i-esimo elemento della lista dei nodi utili.

Pseudocodice 5
eliminaNodiUtiliDominati(nodiS, nodiU)

```
 1: nodiRimossi ← ∅
 2: for i = 1 to |nodiU| do
 3:    vicini_1 ← vicini(NodiU(i))
 4:    for j = i+1 to |nodiU| do
 5:       vicini_2 ← vicini(nodiU(j))
 6:       if vicini_1 ⊇ vicini_2 then
 7:          nodiRimossi ← nodiRimossi + {NodiU(j)}
 8:          nodiU ← nodiU − {NodiU(j)}
 9:       end if
10:    end for
11: end for
12: for all s ∈ nodiS do
13:    vicini ← vicini(s)
14:    for all v ∈ vicini do
15:       if v ∈ nodiRimossi then
16:          vicini ← vicini - {v}
17:       end if
18:    end for
19: end for
20: return
```

Le righe 2–11 sono quelle che cancellano i nodi dominati, mentre quelle da 12 a 19 completano gli effetti della cancellazione eliminando tali nodi dai vicini dei nodi scoperti. Si noti che nel secondo ciclo, quello che inizia a riga 4, l'indice di iterazione j non parte da 1 bensì dal valore di i, in quanto la lista dei nodi utili è stata opportunamente ordinata (cfr. pseudocodice 4).

Infine presentiamo lo pseudocodice relativo alla seconda ottimizzazione. In questo caso, con *statiAttraversati* si intende l'insieme degli stati che sono già stati esplorati dalla ricerca.

Pseudocodice 6
verificaStatoInesplorato(nodiT, nuovoNodoScelto)

```
1: nuovoStato ← nodiT + {nuovoNodoScelto}
2: ordina(nuovoStato)
3: if nuovoStato ∈ statiAttraversati then
4:     return false
5: else
6:     statiAttraversati ← statiAttraversati + {nuovoStato}
7:     return true
8: end if
```

Si noti che l'ordinamento effettuato a riga 2 è arbitrario (per esempio può essere un ordinamento crescente) e serve per poter riconoscere facilmente due stati equivalenti.

4.2.4 Implementazione dell'algoritmo

In questa sottosezione accenniamo brevemente all'implementazione dell'algoritmo. MinBroadcast è stato scritto in Java, linguaggio scelto poiché offre un'ampia libreria di strutture dati e operazioni pronte all'uso e inoltre in quanto è ben conosciuto dall'autore del lavoro. L'algoritmo è stato implementato in due forme:

- interattiva, con interfaccia grafica; l'utente può creare la topologia di rete casualmente, sotto forma di griglia oppure caricarla da un file opportuno e avviare la ricerca; inoltre è possibile visualizzare graficamente la topologia e il risultato della ricerca.
- non interattiva; all'utente è solo richiesto di mettere nella cartella del programma i file relativi alle ricerche da eseguire e avviare l'applica-

zione, che esegue consecutivamente tutte le ricerche desiderate e salva i risultati, in forma testuale, nella stessa cartella.

Si noti che nell'implementazione dell'algoritmo sono state utilizzate variabili aggiuntive e strutture dati appropriate, in modo da aumentare l'efficienza e ridurre i tempi di esecuzione: per esempio, l'insieme *statiAttraversati* presente nello pseudocodice 6 non è un comune insieme o una lista, ma un hashset, ovvero un insieme su cui le ricerche vengono effettuate in modo particolarmente efficiente. Inoltre, al fine di sfruttare pienamente i microprocessori moderni, che spesso sono in grado di eseguire più operazioni contemporaneamente, la ricerca viene spezzata ed eseguita da più thread, quando possibile.

Le prestazioni ottenuti sono state decisamente soddisfacenti: la ricerca di un MCDS su 20 topologie differenti formate da 60 nodi, per ognuna delle quali sono stati scelti 20 nodi sorgente per il messaggio e 4 diversi raggi di trasmissione (per un totale di 1600 ricerche) ha richiesto circa 45 minuti su un singolo PC.

Capitolo 5

Esperimenti con il simulatore

I simulatori sono strumenti particolarmente utili per compiere valutazioni e comparazioni. Per questo motivo le simulazioni ricoprono un ruolo importante in questo lavoro, poiché hanno permesso di confrontare i cinque metodi selezionati in svariate condizioni della rete.

In questo capitolo innanzitutto introduciamo brevemente OMNeT++, ovvero il simulatore utilizzato. Quindi sono presentati gli obiettivi principali delle simulazioni, gli scenari considerati per coprire tutti i casi di interesse e i parametri generali delle simulazioni. Infine viene presentato il processo che ha portato alla scelta dei parametri delle singole tecniche da comparare.

5.1 Il simulatore utilizzato: OMNeT++

OMNeT++ [44] è un simulatore a eventi discreti basato sul linguaggio C++, che può essere gratuitamente utilizzato per scopi accademici. OMNeT++ non è specializzato nella modellazione e simulazione della comunicazione all'interno di una particolare famiglia di reti, ma è piuttosto un framework estendibile e quindi adattabile a un gran numero di situazioni.

Per eseguire la comparazione è stata usata la versione 4.0 del simulatore, abbinata al *Mobility Framework*, un'estensione per OMNeT++ appositamente pensata per supportare simulazioni di reti wireless mobili, come per esempio le WSN. In particolare sono stati usati un modello dettagliato della

Distanza (m)	RSSI	Distanza (m)	RSSI
5	-4.9945	55	-36.2363
10	-14.0254	60	-37.3699
15	-19.3081	65	-38.4128
20	-23.0563	70	-39.3783
25	-25.9636	75	-40.2772
30	-28.3390	80	-41.1181
35	-30.3474	85	-41.9080
40	-32.0872	90	-42.6527
45	-33.6218	95	-43.3571
50	-34.9945	100	-44.0254

Tabella 5.1: Distanze e RSSI nel simulatore

radio CC2420, il livello MAC 802.15.4 e il modello di canale, chiamato *Multiple Access Interference*, realizzati nell'ambito del progetto europeo WASP[1] [46]. È importante evidenziare che il modello di canale utilizzato è fondamentalmente il *log-distance path loss model*: infatti è previsto che all'aumentare della distanza diminuisca la potenza del segnale di trasmissione e aumenti la probabilità che siano presenti errori nel messaggio ricevuto; ovviamente è contemplata la possibilità che avvengano collisioni e, in aggiunta, per il calcolo del rumore e quindi per decidere se un messaggio sia valido o meno vengono considerati e sommati i contributi di tutte le comunicazioni.

Si noti che la radio CC2420 e il livello MAC 802.15.4 sono spesso impiegati nelle WSN reali e sono usati nei test sperimentali eseguiti per questo lavoro.

Visto che, in alcuni metodi scelti per la comparazione, il valore del RSSI viene usato per ottenere una stima della distanza dal nodo da cui è appena stato ricevuto il messaggio, si ritiene utile specificare il range di valori che esso può assumere. Tramite apposite simulazioni è stato possibile rilevare che l'RSSI è un numero compreso nell'intervallo che va da 50 a -45, dove però tutti i valori tra 50 e -10 fanno riferimento a distanze molto piccole. In tabella 5.1 è riportato il valore del RSSI rilevato alle distanze multiple di 5, fino al raggio di trasmissione massimo.

[1] Wirelessly Accessible Sensor Populations

Si noti che l'RSSI non varia linearmente con la distanza e ciò è dovuto al fatto che, nel modello di canale utilizzato, il segnale si attenua seguendo il log-distance path loss model, che riproduce il decadimento dei segnali radio all'interno di edifici in base a una funzione logaritmica. Si noti inoltre che, in realtà, il raggio di trasmissione massimo non è esattamente pari a 100 metri, ma esiste la probabilità, seppur piccola, di ricevere messaggi anche da distanze leggermente superiori. La scelta di riportare i valori fino a 100 metri è dovuta a semplice praticità.

5.2 Obiettivi e scenari delle simulazioni

L'obiettivo principale delle simulazioni, come già affermato in precedenza, è quello di confrontare OppFlooder con gli altri quattro metodi selezionati in un ampia varietà di condizioni di rete, in modo da poter verificare se alcune tecniche sono preferibili ad altre in particolari condizioni o addirittura dimostrare che una di esse, magari proprio OppFlooder, si dimostra la più adatta a operare nelle varie situazioni proposte. In particolare siamo interessati a valutare le prestazioni dei metodi scelti rispetto a:

- la densità della rete, poiché essa potrebbe influire sull'efficienza delle tecniche proposte, dato che alcuni nodi potrebbero ritrasmettere senza che ce ne sia necessità in presenza di alte densità;
- le dimensioni della rete, in quanto aumentando l'area è necessario un maggior numero di ritramissioni e di hop per raggiungere l'intera rete e ciò rende l'evoluzione del broadcast tendenzialmente meno prevedibile;
- la congestione della rete, in modo da verificare come ogni tecnica subisce i suoi effetti; la congestione della rete può essere ottenuta sia aumentando la frequenza di invio dei messaggi da parte dei nodi sorgente sia ingrandendo la dimensione dei pacchetti; è stato scelto il primo modo siccome i messaggi di broadcast sono generalmente di piccole dimensioni;

- la mobilità della rete, così da appurare quanto i metodi selezionati siano adatti a contesti in cui i nodi possono muoversi a differenti velocità.

Al fine di coprire tutti i casi di nostro interesse ed eseguire le simulazioni in modo che i risultati ottenuti siano facilmente riconducibili alle condizioni che li hanno prodotti, si è scelto di strutturare le simulazioni compiendo i seguenti passi:

- identificare uno scenario base, che rappresenti le condizioni della rete che ci interessano maggiormente, per esempio perché sono quelle che spesso si incontrano nell'ambito delle WSN; in realtà gli scenari base saranno due, il primo con i nodi fissi e il secondo con i nodi in movimento, visto che riteniamo particolarmente interessante studiare approfonditamente il comportamento delle tecniche di broadcast in presenza di mobilità;

- scegliere un limitato numero di valori, uno dei quali relativo allo scenario base, su cui ogni parametro della rete può variare; tale scelta è strettamente legata agli scopi delle simulazioni;

- creare i rimanenti scenari di simulazione a partire dai due scenari base, facendo variare un solo parametro della rete per volta.

Per quanto riguarda gli scenari base, si è deciso di scegliere una rete con densità dei nodi media (300 nodi per chilometro quadrato), un'area non eccessivamente grande (0.2 chilometri quadrati), in modo da poter eseguire le ricerche di minimo teorico più rapidamente, e una frequenza di invio dei messaggi bassa (un messaggio ogni 20 secondi), poiché in molti casi è ciò che si trova nelle WSN. In uno dei due scenari i nodi sono completamente fermi, mentre nell'altro si possono muovere a una velocità ridotta, compresa tra 0 e 2 metri al secondo. Questo particolare intervallo di velocità è stato scelto poiché è rappresentativo delle condizioni di mobilità tipiche di uno scenario in cui i soggetti monitorati sono persone o animali, circostanza abbastanza tipica per una WSN mobile.

La scelta degli altri valori, per ognuno dei parametri, è stata compiuta in modo da prendere in considerazione più situazioni possibili. Tutti questi

Densità (nodi/km^2)	100	200	**300**	400	500	
Area (km^2)	0.1	**0.2**	0.5	1.0	1.5	
Frequenza invio (msg/s)	0.01	**0.05**	0.1	0.3	0.5	1.0
Velocità nodi (m/s)	**[0,0]**		**[0,2]**	[3,5]	[5,20]	

Tabella 5.2: I parametri su cui sono basati gli scenari di simulazione

valori sono riportati nella tabella 5.2, evidenziando quelli relativi agli scenari base tramite l'uso del grossetto. Si noti che la velocità dei nodi è descritta tramite l'intervallo in cui può variare.

Per quanto riguarda il movimento dei nodi, si è scelto di adottare un modello piuttosto semplice, conosciuto in letteratura con il nome *random waypoint* [2], senza però che siano previsti periodi di tempo in cui il nodo si ferma. Questo tipo di mobilità è difficilmente riscontrabile in una rete reale, ma si ritiene che sia particolarmente idoneo per valutare quanto un metodo di broadcast sia adatto a generici contesti mobili, dato che rappresenta il caso peggiore che possa capitare.

In conclusione, in totale saranno eseguiti nove studi: uno per ciascuno scenario base e poi quelli, con e senza mobilità, in cui sono variabili rispettivamente la densità dei nodi, l'area della rete e la frequenza di invio dei messaggi; infine quello in cui varia la velocità di movimento dei nodi.

5.3 Parametri generali delle simulazioni

In questo paragrafo presentiamo brevemente (e riportiamo in tabella 5.3) quei parametri che sono uguali per tutte le simulazioni e per ognuno di essi motiviamo la scelta, se necessario:

- tempo di simulazione; la durata di ogni simulazione è pari a 18 minuti, dove i primi 3 minuti un tempo sufficiente a permettere a SBAFlooder e AHBPFlooder di raccogliere le informazioni sul vicinato, mentre i successivi 15 sono quelli in cui avviene effettivamente il broadcast; si è scelta una durata sufficientemente lunga per limitare l'effetto di spora-

dici comportamenti anomali e per permettere alle reti in movimento di variare considerevolmente;

- raggio di trasmissione; in questo caso il valore non è stato scelto, ma rilevato sperimentalmente, ed è pari a circa 100 metri;
- numero nodi sorgente; in ogni esperimento sono 20 i nodi che fungono da sorgente dei messaggi di broadcast, così da generare un traffico più consistente e vario rispetto a quello indotto da un singolo nodo;
- dimensione dei dati contenuti in ogni messaggio di broadcast; è stato deciso di impostare una dimensione piuttosto piccola, pari a 16 byte, in quanto i messaggi di broadcast, in genere, non trasportano una gran mole di informazioni;
- numero di ripetizioni per ogni situazione; al fine di evitare che i risultati possano essere inficiati da alcune simulazioni "sfortunate", si è deciso di far girare ogni simulazione 20 volte nel caso di nodi fissi e 10 volte nel caso di nodi in movimento; ovviamente ognuna di queste ripetizioni è differente dalle altre sia per la topologia della rete, sia per i momenti in cui vengono inviati i messaggi e anche per la mobilità dei nodi, quando presente.

Non è un vero e proprio parametro, ma si vuole anche far notare che il perimetro della rete, in tutte le simulazioni, ha forma quadrata.

Parametro	**Valore**
Tempo di simulazione	18 minuti
Raggio di trasmissione	$\approx$ 100 metri
# nodi sorgente messaggi	20
Dimensione payload dati	16 byte
# ripetizioni (nodi fissi)	20
# ripetizioni (nodi mobili)	10

Tabella 5.3: I parametri comuni a tutte le simulazioni

5.4 Scelta dei parametri dei metodi

Prima di eseguire gli esperimenti richiesti per realizzare gli studi desiderati, è stato necessario far girare una prima serie di simulazioni, al fine di decidere il valore da assegnare ai parametri dei singoli metodi. Le simulazioni sono state effettuate sui due scenari base, solitamente con densità variabile oppure, in alcuni casi, con mobilità variabile. Per la descrizione degli indici di prestazione utilizzati nei risultati, si rimanda all'inizio del capitolo 7.

Nel processo di scelta, ogni parametro è stato valutato singolarmente, a meno che due parametri non siano fortemente legati tra loro. Nel caso in cui il valore di un parametro non sia stato ancora scelto ne viene preso uno in base al buon senso oppure ai parametri già scelti per un altro metodo o ancora a dei valori consigliati nella letteratura.

Per motivare la scelta di ogni parametro si è deciso di inserire solo i risultati più significativi, in modo da non appesantire inutilmente la presentazione. In particolare sono quasi sempre stati presentati i risultati ottenuti

Metodo	**Parametro**	**Valore**
OppFlooder	T_{max}	1 s
	$RSSI_{max}$	-10.0
DistFlooder	T_{max}	1 s
	$RSSI_{max}$	-35.0
JDCTFlooder	T_{max}	1 s
	$RSSI_{max}$	-37.5
	C	2
SBAFlooder	T_{max}	1 s
	T_c	60 s (r. fissa) - 10 s (r. mobile)
	T_t	120 s (r. fissa) - 15 s (r. mobile)
AHBPFlooder	T_{max}	0.1 s
	T_c	60 s (r. fissa) - 10 s (r. mobile)
	T_t	120 s (r. fissa) - 15 s (r. mobile)
	$RSSI_c$	-43.3

Tabella 5.4: I parametri dei metodi nelle simulazioni

dalle simulazioni con rete mobile (alla velocità più bassa), in quanto sono generalmente equivalenti ai risultati delle simulazioni su rete fissa ai fini della scelta dei parametri.

In tabella 5.4 sono raccolti, per ogni metodo, i valori dei parametri ottenuti mediante il processo di scelta, così da facilitarne la consultazione.

5.4.1 OppFlooder

La descrizione di OppFlooder (cfr. paragrafo 3.1) mostra come esso sia un metodo semplice, sia dal punto di vista del funzionamento sia per quanto riguarda il setup. Infatti è necessario scegliere un solo parametro per OppFlooder, ovvero il massimo ritardo previsto per una generica ritrasmissione (T_{max}). In realtà abbiamo inserito un parametro aggiuntivo, cioè una soglia sulla distanza sotto cui scartare direttamente i messaggi ricevuti. Si è deciso di assegnare a tale soglia un valore piccolo, compreso tra 5 e 10 metri; volendo trasformare la soglia sulla distanza in una soglia sul RSSI si ottiene $RSSI_{max} = -10.0$.

Scelta di $\mathbf{T}_{max}$

Noto che il ritardo di ritrasmissione è utile non solo per ridurre la probabilità che avvengano collisioni, ma anche per favorire la ritramissione dei nodi più lontani e permettere un corretto funzionamento del delay and cancel, si ritiene che il valore massimo di tale ritardo non debba essere troppo piccolo. Sono quindi stati selezionati quattro differenti valori tra cui optare: 1.5, 1.0, 0.5 e 0.1 secondi.

Come si può vedere in figura 5.1, la variazione di T_{max} produce effetti tutto sommato limitati per quanto riguarda la copertura della rete, il costo di broadcast e il numero di ritrasmissioni. Ovviamente influenza molto più marcatamente il tempo richiesto, in media, affinché un messaggio raggiunga ogni nodo della rete.

Si è deciso di porre $T_{max} = 1s$, poiché questa scelta permette di ottenere prestazioni praticamente ottimali sia per quanto riguarda la copertura della rete (a parte il caso a bassa densità) sia per ciò che concerne l'efficienza. Qualora fosse desiderabile avere una latenza più bassa si può optare senza

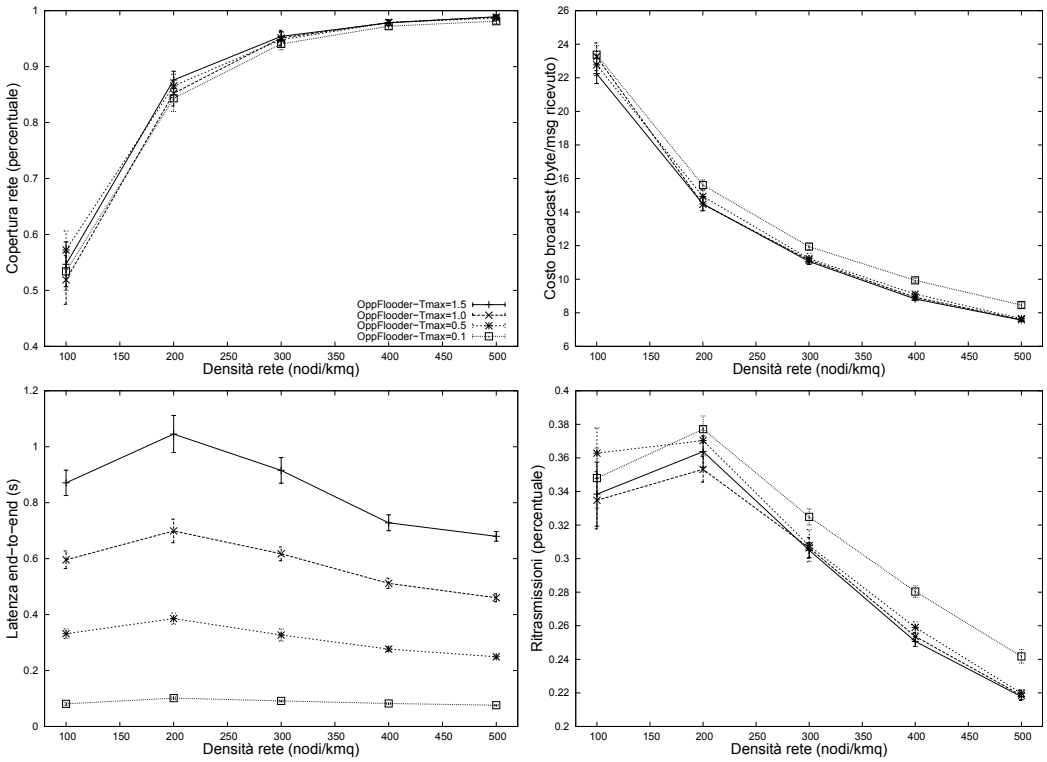

Figura 5.1: L'impatto di T_{max} in OppFlooder

problemi per 0.5 secondi, mentre il valore 0.1 secondi è quello che, latenza a parte, fa registrare le prestazioni peggiori ed è abbastanza distaccato rispetto agli altri.

5.4.2 DistFlooder

Nel paragrafo 3.2 abbiamo visto che il funzionamento di DistFlooder può essere influenzato da due parametri, ovvero il massimo ritardo previsto per una generica ritrasmissione (T_{max}) e la soglia sulla distanza, sotto cui un qualsiasi messaggio non deve essere ritrasmesso (D). In realtà quest'ultimo, nell'implementazione sul simulatore, è stato sostituito da una soglia sul valore del RSSI relativo al messaggio ricevuto, sopra cui esso non deve essere ritrasmesso ($RSSI_{max}$); in tabella 5.1 sono raccolte le corrispondenze tra distanza e RSSI. Vediamo ora quali valori sono stati scelti per questi due parametri.

Scelta di T_{max}

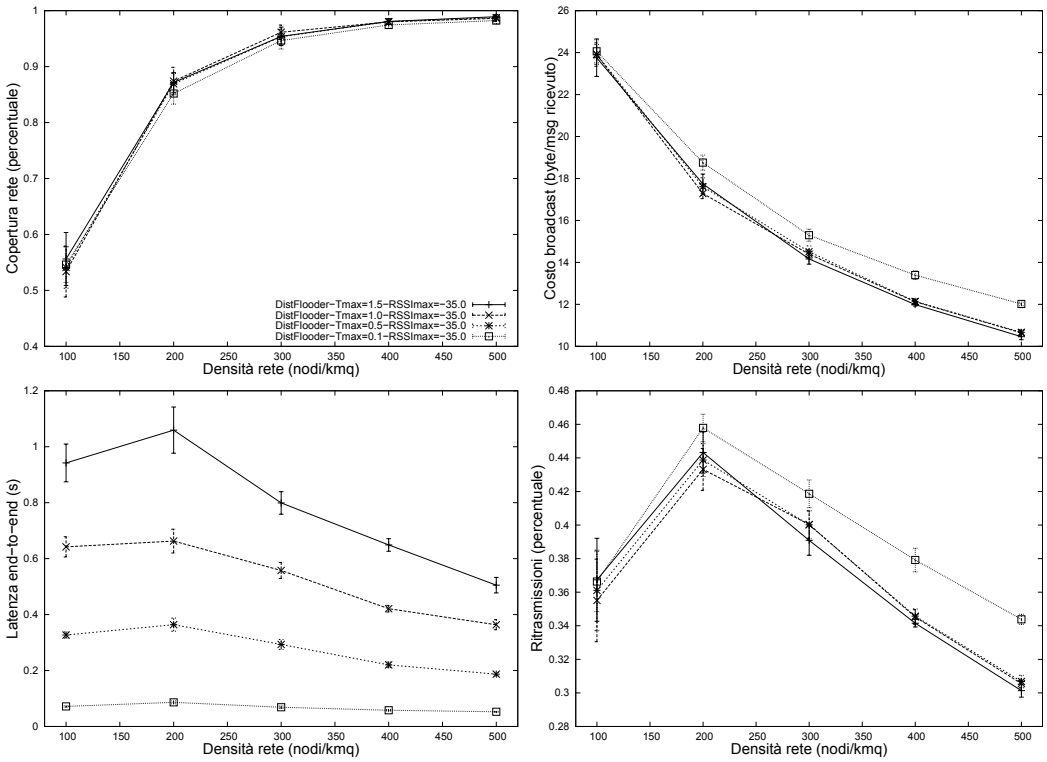

Figura 5.2: L'impatto di T_{max} in DistFlooder

Per scegliere il ritardo massimo di ritrasmissione è stato fissato $RSSI_{max} = -35.0$, un valore equivalente a una distanza di circa 50 metri. Il parametro T_{max} invece è stato fatto variare, esattamente come per OppFlooder, su quattro differenti valori: 1.5, 1.0, 0.5 e 0.1 secondi.

La figura 5.2 mostra che, per la scelta di T_{max} in DistFlooder si può fare lo stesso identico ragionamento compiuto per OppFlooder, con l'unica differenza che in questo caso il ritardo di ritramissione è utile per ricevere una o più repliche e quindi decidere se ritrasmettere o meno. Scegliamo quindi $T_{max} = 1s$. Questa decisione sarebbe comunque stata quella più logica, in quanto in seguito si vogliono confrontare questi metodi.

Scelta di RSSI$_{max}$

Rimane ora da scegliere il valore da assegnare al parametro che influenza maggiormente il funzionamento di DistFlooder, ovvero $RSSI_{max}$. In questo caso, oltre a valutare i risultati ottenuti con una serie di differenti valori,

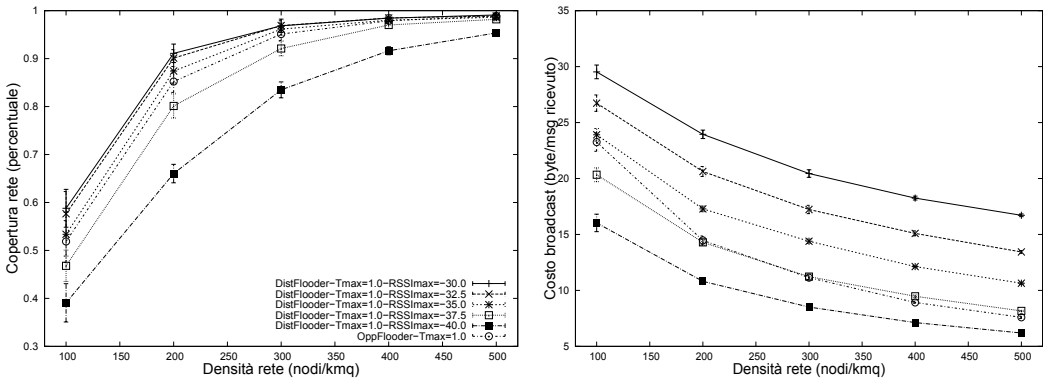

Figura 5.3: L'impatto di RSSI$_{max}$ in DistFlooder

basiamo la scelta anche sul fatto, come già accennato in precedenza, che l'obiettivo è quello di comparare i vari metodi. L'idea adottata è quindi optare per il valore di $RSSI_{max}$ che permette di raggiungere una copertura di rete più simile possibile a quella ottenuta con OppFlooder. I cinque valori provati per questo parametro sono i seguenti: -30.0, -32.5, -35.0, -37.5 e -40.0; ovviamente $T_{max} = 1s$.

Risulta evidente (figura 5.3) come il parametro $RSSI_{max}$ possa influire sulle prestazioni di DistFlooder, favorendo la copertura della rete oppure l'efficienza, vista come limitato numero di ritrasmissioni e quindi basso costo del broadcast. La nostra scelta ricade sul valore -35.0, poiché è quello che maggiormente avvicina la copertura della rete di DistFlooder a quella di OppFlooder. Si noti che, nei grafici qui mostrati, la copertura è leggermente superiore a quella di OppFlooder, ma negli esperimenti con nodi fissi risulta leggermente inferiore.

5.4.3 JDCTFlooder

JDCTFlooder, descritto nel paragrafo 3.3, richiede un parametro in più rispetto a DistFlooder: oltre a T_{max} e $RSSI_{max}$ bisogna scegliere anche C, ovvero la soglia sul numero di repliche, oltre cui un generico messaggio non deve essere ritrasmesso. Visto che sia $RSSI_{max}$ sia C influiscono sul numero di ritrasmissioni e anche sulla copertura della rete, la loro scelta è stata parzialmente collegata.

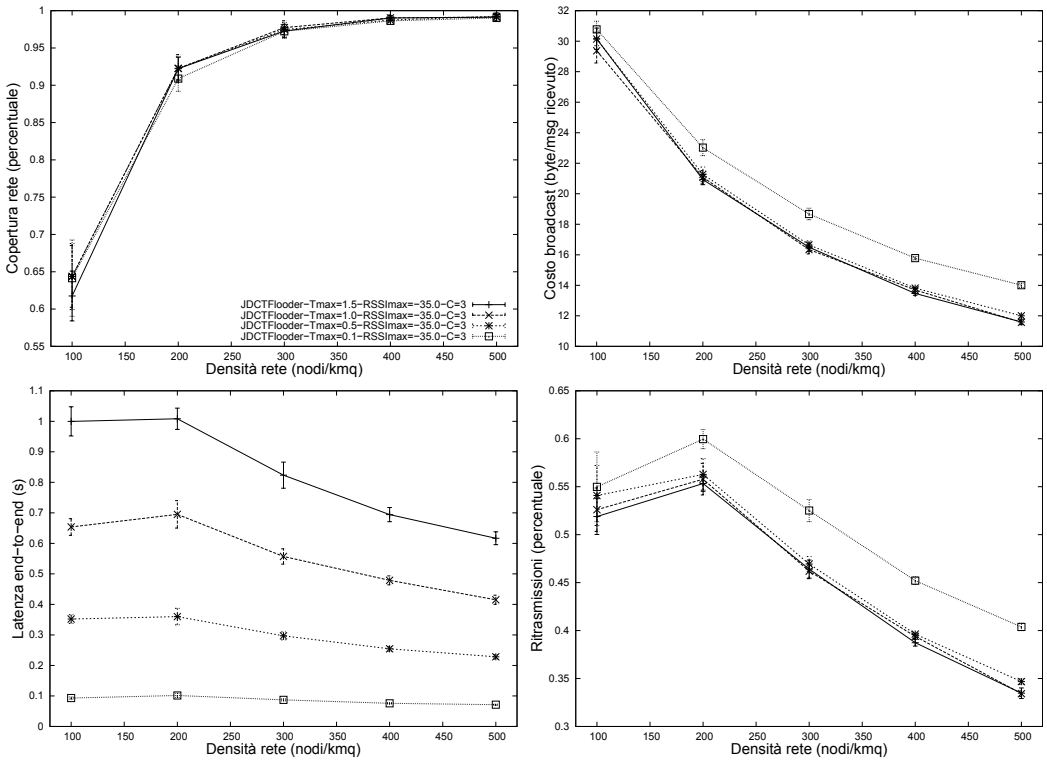

Figura 5.4: L'impatto di T_{max} in JDCTFlooder

Scelta di T_{max}

Anche in questo caso, per scegliere il ritardo massimo di ritrasmissione è stato fissato $RSSI_{max} = -35.0$, mentre si è posto $C = 3$. I quattro valori su cui è stato fatto variare T_{max} sono sempre gli stessi: 1.5, 1.0, 0.5 e 0.1 secondi.

Ancora una volta, come per i due metodi precedenti, scegliamo $T_{max} = 1s$, sulla base delle medesime motivazioni e poiché l'andamento dei risultati di figura 5.4 è piuttosto simile ai casi precedenti.

Scelta di $RSSI_{max}$ e C

In JDCTFlooder, sia $RSSI_{max}$ sia C possono singolarmente influenzare le prestazioni; l'idea è quindi di trovare una combinazione dei due (potrebbe essercene più di una valida) che soddisfi le nostre necessità. Per fare ciò sono state innanzitutto eseguite alcune simulazioni tenendo fisso $C = 3$ e variando solo $RSSI_{max}$. I valori scelti per $RSSI_{max}$ sono gli stessi usati per DistFlooder, ovvero: -30.0, -32.5, -35.0, -37.5, -40.0.

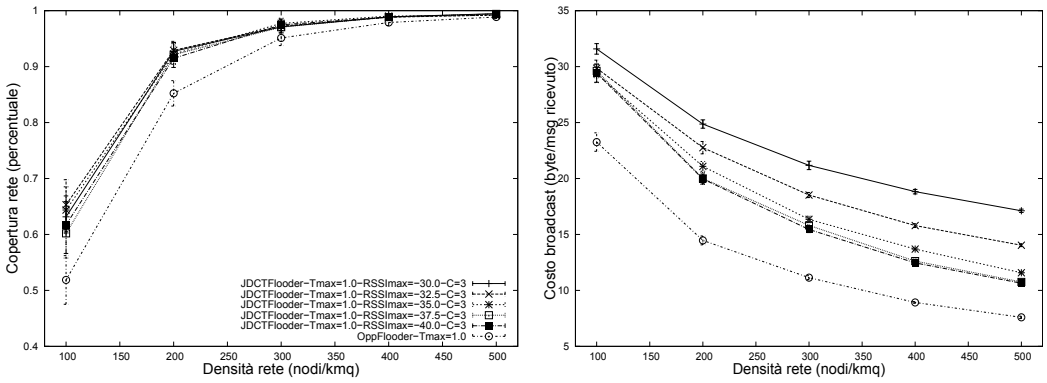

Figura 5.5: L'impatto di RSSI_{max} in JDCTFlooder

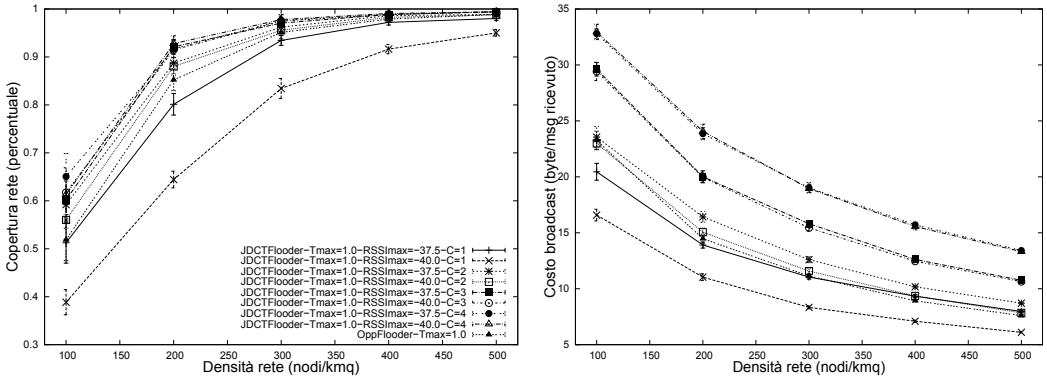

Figura 5.6: L'impatto di RSSI_{max} e C in JDCTFlooder

Osservando figura 5.5, appare subito evidente come JDCTFlooder permette di ottenere una copertura della rete maggiore a OppFlooder, in tutti i casi, a un costo di broadcast superiore. A questo punto è stato deciso di eseguire le simulazioni per scegliere C, ma invece di fissare RSSI_{max} si è preferito prendere due possibili valori (-37.5 e -40.0); per quanto riguarda C invece si è ritenuto sufficiente indagare sui valori 1, 2, 3 e 4.

Figura 5.6 mostra come la scelta $C = 2$ sia quella che rende più simili JDCTFlooder e OppFlooder, soprattutto alla densità dello scenario base e a quelle superiori. Per quanto riguarda $RSSI_{max}$ sì è deciso di optare per il valore -37.5, in quanto -40.0 comincia a rendere JDCTFlooder particolarmente simile a OppFlooder, non solo dal punto di vista dei risultati, ma anche rispetto al funzionamento dei due metodi (cfr. paragrafo 3.6.2).

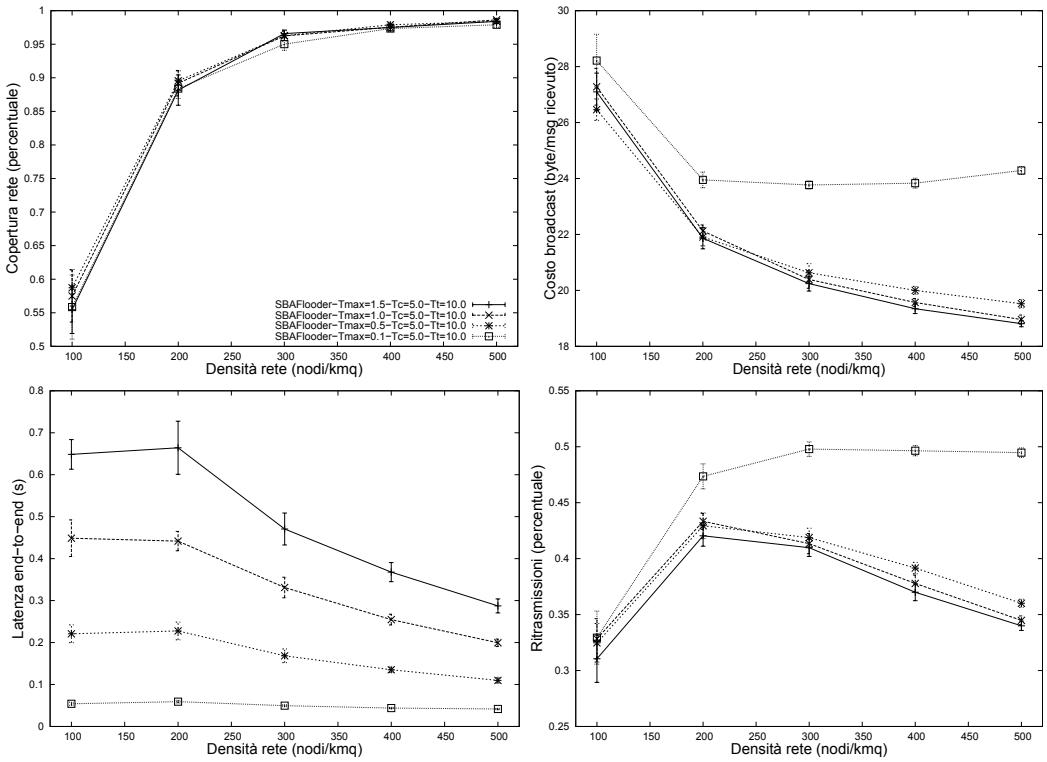

Figura 5.7: L'impatto di T_{max} in SBAFlooder

5.4.4 SBAFlooder

SBAFlooder (paragrafo 3.4) necessita che vengano scelti tre parametri: il sempre presente T_{max}, il periodo di tempo con cui devono essere trasmessi i messaggi di controllo (T_c) e il timeout dopo cui cancellare un vicino se non si è ricevuto alcun messaggio di controllo da parte sua (T_t). È facile intuire che gli ultimi due parametri sono strettamente legati e quindi il loro processo di scelta ha seguito le stesse fasi viste in JDCTFlooder per le due soglie.

Scelta di T_{max}

La selezione T_{max} ha richiesto gli stessi passaggi visti nei casi precedenti ed è avvenuta sempre tra gli stessi valori: 1.5, 1.0, 0.5 e 0.1 secondi. Per quanto riguarda gli altri due parametri si è stabilito di imporre $T_c = 5$ e $T_t = 10$ secondi.

L'andamento di SBAFlooder, in figura 5.7, non risulta particolarmente differente rispetto ai metodi precedenti e quindi scegliamo sempre $T_{max} = 1s$.

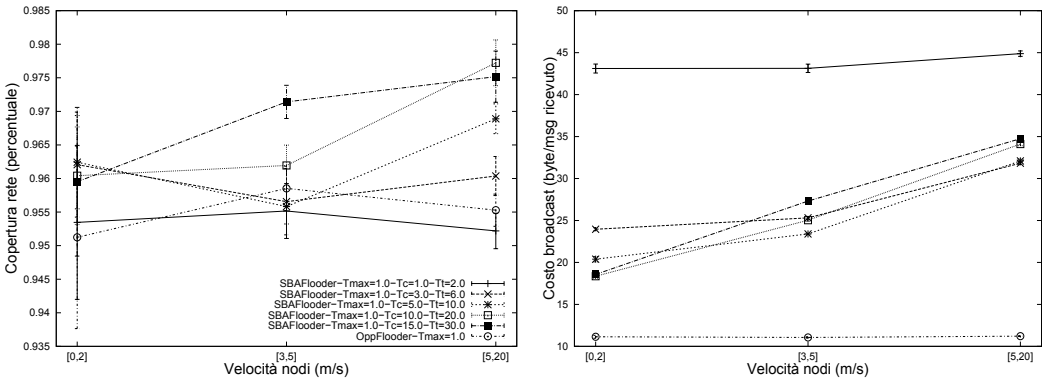

Figura 5.8: L'impatto di T_c in SBAFlooder

Scelta di T_c e T_t

Per quanto riguarda questi due parametri, è innanzitutto interessante notare che nel caso di reti fisse è logico preferire due valori grandi, poiché gli unici cambiamenti nella rete dipendono dal possibile malfunzionamento dei nodi e ciò solitamente accade di rado. Per questo motivo è stato deciso che $T_c = 60s$ e $T_t = 120s$ in assenza di mobilità; alcune simulazioni, qui non riportate, hanno dimostrato che tali valori non compromettono le prestazioni di SBAFlooder. Il discorso cambia in presenza di mobilità e dunque è stata eseguita una serie di simulazioni, a differenti velocità di movimento dei nodi, per verificare come il periodo di emissione dei messaggi di controllo influenza le prestazioni della rete. A T_c sono stati fatti assumere i valori 1, 3, 5, 10 e 15 (secondi), mentre $T_t = 2 \cdot T_c$.

I risultati di figura 5.8 mostrano come, in generale, la copertura della rete vari solo marginalmente. A basse velocità i due valori da preferire sono 10 e 15 secondi, siccome risultano essere maggiormente efficienti. Quindi, per concludere, sono stati scelti proprio questi due ultimi valori per T_c, da usare nelle simulazioni necessarie per selezionare anche il valore di T_t tra $1.5 \cdot T_c$, $2.0 \cdot T_t$ e $2.5 \cdot T_t$.

Dai risultati di figura 5.9 emerge che la scelta migliore sia $T_c = 10s$ e $T_t = 1.5 \cdot T_c = 15s$: infatti, rispetto alle altre opzioni, le differenze per quanto riguarda la copertura della rete sono marginali, ma il costo del broadcast è praticamente uguale in presenza di basse velocità, mentre è inferiore nel caso di medie velocità.

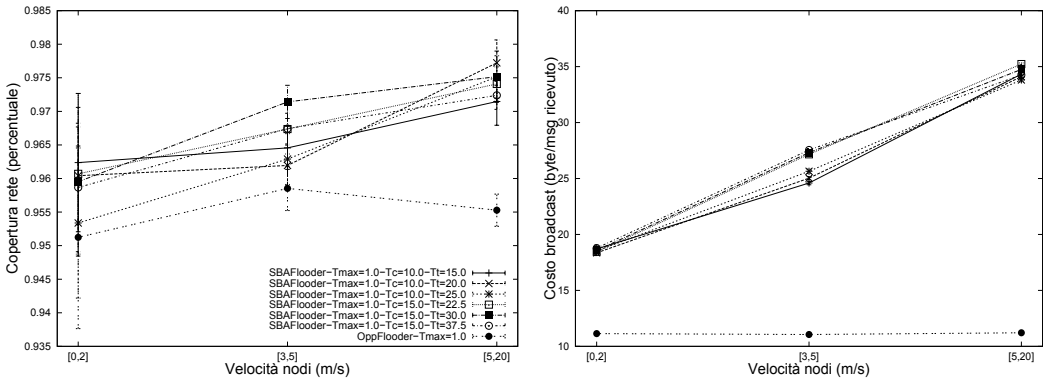

Figura 5.9: L'impatto di T_c e T_t in SBAFlooder

5.4.5 AHBPFlooder

AHBPFlooder, descritto dettagliatamente al paragrafo 3.5, ha gli stessi parametri di SBAFlooder: T_{max}, T_c e T_t. Il ritardo di ritrasmissione però, in questo caso, ha il solo scopo di ridurre la probabilità che avvengano collisioni e quindi il suo valore può essere scelto sensibilmente più piccolo rispetto a tutti gli altri metodi.

Le prime simulazioni con AHBPFlooder hanno prodotto dei risultati mediocri rispetto a quelli degli altri metodi e ciò è stato solo in parte inaspettato (cfr. 3.6.3). Si è pensato che la possibile causa potesse essere l'inaffidabilità dei nodi adiacenti più lontani. Per verificare questa teoria ed escludere possibili errori di implementazione, AHBPFlooder è stato testato su alcune semplici topologie a griglia, con nodi opportunamente distanziati, e i risultati ottenuti sono stati ottimi, confermando l'ipotesi fatta. Per questo motivo è stato necessario inserire un ulteriore parametro, ovvero una soglia sul valore del RSSI dei messaggi di controllo ricevuti ($RSSI_c$), sotto cui il messaggio viene semplicemente scartato. Affrontiamo subito la scelta di questo parametro e quindi proseguiamo con gli altri.

Scelta di $\mathbf{RSSI}_c$

Per selezionare il valore di questo nuovo parametro sono stati fissati gli altri tre: $T_{max} = 0.1s$, T_c e T_t uguali ai valori proposti per SBAFlooder, il primo in base alla considerazione fatta poco sopra e gli altri due supponendo che la scelta fatta per SBAFlooder potesse essere valida anche per AHBPFlooder.

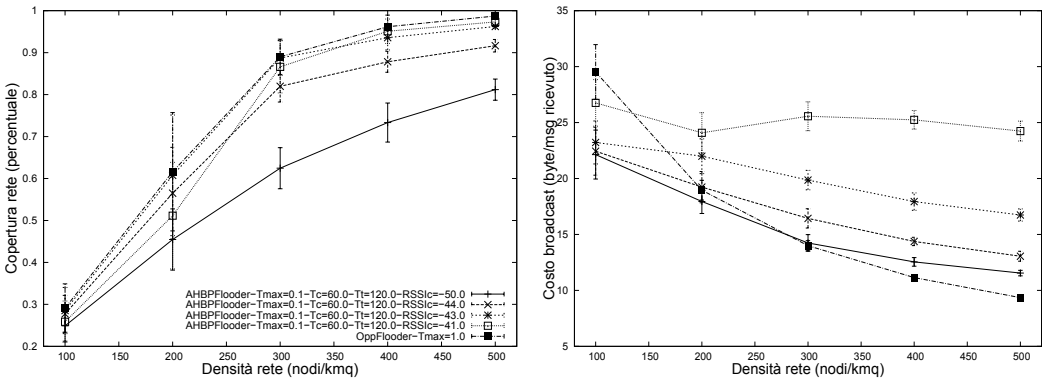

Figura 5.10: L'impatto del parametro aggiuntivo RSSI_c in AHBPFlooder (nodi fissi)

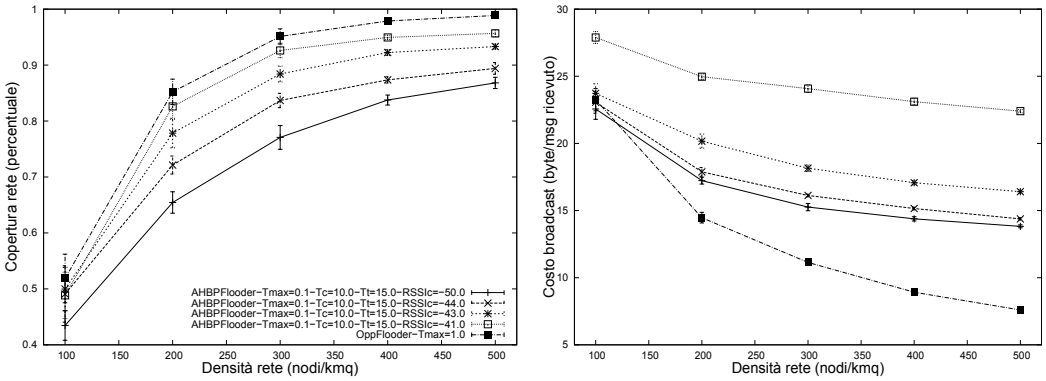

Figura 5.11: L'impatto del parametro aggiuntivo RSSI_c in AHBPFlooder (nodi mobili)

Si è optato di far variare $RSSI_c$ tra i seguenti valori: -50.0 (in pratica, nessuna soglia), -44.0, -43.0 e -41.0. È stato deciso di mostrare sia i risultati ottenuti con rete mobile sia quelli con rete fissa, in quanto tra i due casi è presente una certa differenza.

Le figure 5.10 e 5.11 mostrano come sia evidente l'effetto di questo nuovo parametro. Il valore -41.0, soprattutto con una rete mobile, è quello che permette di ottenere la copertura di rete maggiore, ma il costo del broadcast è nettamente superiore rispetto a quello delle altre opzioni. Si è quindi scelto di optare per $RSSI_c = -43.3$, un valore equivalente a circa 95 metri, che essendo di poco superiore a -43.0 offre un buon compromesso tra copertura della rete e costo di broadcast.

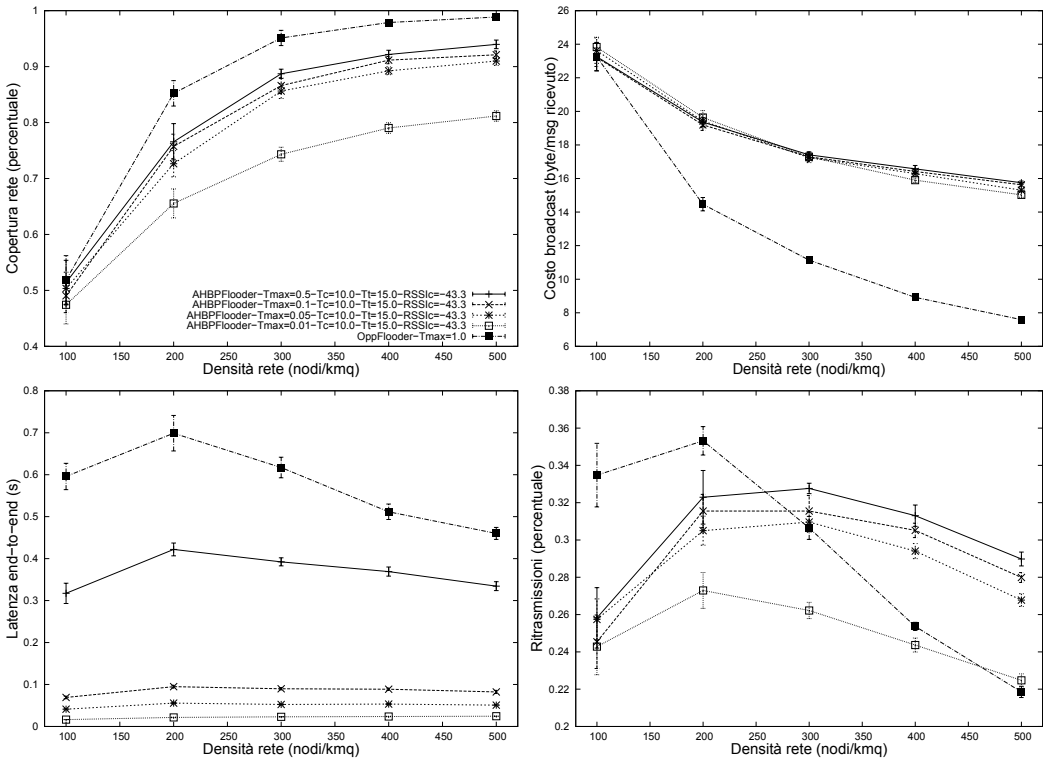

Figura 5.12: L'impatto di T_{max} in AHBPFlooder

Scelta di T_{max}

Per AHBPFlooder, i valori su cui si è deciso di far variare T_{max} sono i seguenti: 0.5, 0.1, 0.05 e 0.01 secondi. $RSSI_c$ è già stato scelto, mentre T_c e T_t assumono rispettivamente i valori 10 e 15 secondi (rete in movimento).

Dai risultati di figura 5.12 emerge che il valore 0.5 offre i risultati che maggiormente si avvicinano (e addirittura superano, in rete fissa e a bassa densità) a quelli di OppFlooder, per quanto riguarda la copertura. Nonostante ciò si è preferito scegliere $T_{max} = 0.1s$, in modo da distinguerlo nettamente dal massimo ritardo di ritrasmissione selezionato per gli altri quattro metodi.

Scelta di T_c e T_t

Per la scelta di questi due parametri si possono fare le stesse considerazioni viste per SBAFlooder, in quanto il meccanismo di creazione e mantenimento della topologia locale funziona esattamente allo stesso modo in questi due metodi. Ancora una volta sono state prima eseguite le simulazioni con T_c

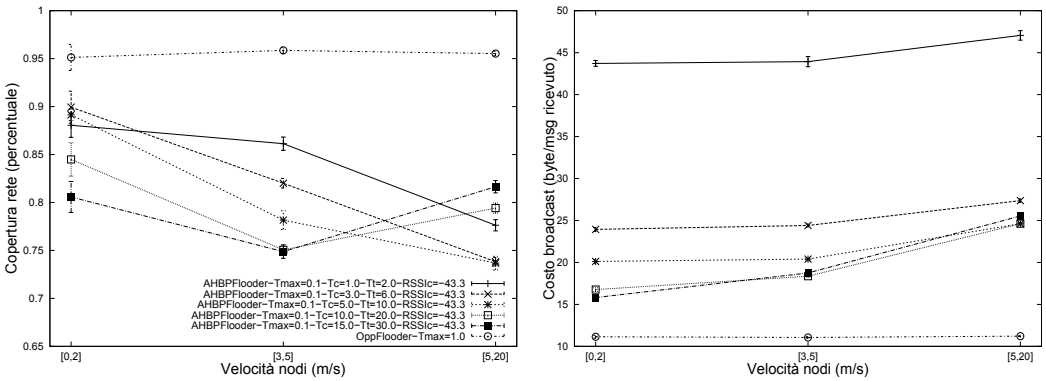

Figura 5.13: L'impatto di T_c in AHBPFlooder

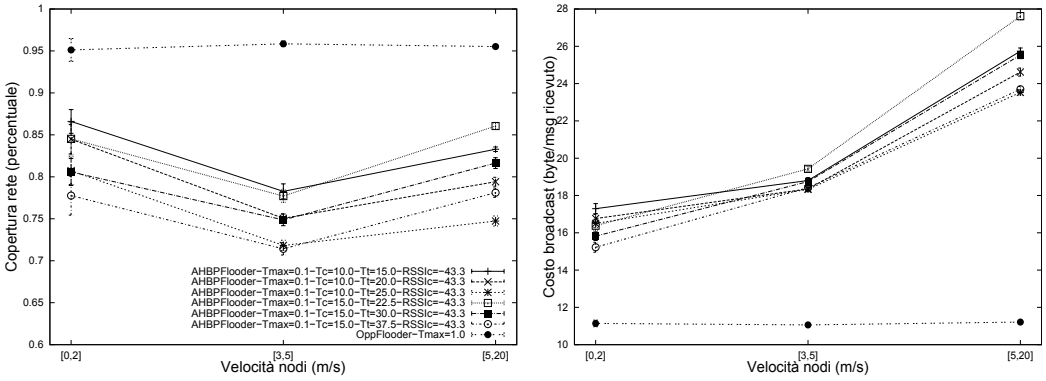

Figura 5.14: L'impatto di T_c e T_t in AHBPFlooder

variabile tra 1, 3, 5, 10 e 15 secondi e $T_t = 2 \cdot T_c$.

Anche in questo caso (figura 5.13) la decisione ricade sui valori 10 e 15 secondi e il primo sembra essere il migliore, poiché garantisce una copertura di rete superiore di alcuni punti percentuali nello scenario a velocità ridotta. Il processo di scelta si conclude con le simulazioni in cui vengono considerati questi due valori e T_t viene preso pari a 1.5, 2 e 2.5 volte T_c.

Figura 5.14 mostra dei risultati che potrebbero portare a optare per più di una combinazione di T_c e T_t. Tenendo però conto della scelta già compiuta per SBAFlooder e delle prestazioni non eccellenti di AHBPFlooder riguardo la copertura della rete, si è deciso di confermare $T_c = 10s$ e $T_t = 15s$.

Capitolo 6

Esperimenti in scenari reali

Gli esperimenti compiuti in scenari reali rappresentano il valore aggiunto di questo lavoro, in quanto permettono di verificare in che misura i risultati ottenuti mediante le simulazioni siano attendibili, nell'ottica di volerli utilizzare per studi e valutazioni nell'ambito delle WSN reali.

Effettuare esperimenti su reti reali presenta, purtroppo, alcune limitazioni di rilievo. Innanzitutto, almeno al giorno d'oggi, non esistono reti, sufficientemente grandi e liberamente utilizzabili, che permettono di variare la topologia dei nodi e di studiare in maniera ripetibile la mobilità; ciò potrebbe essere risolto dal progetto Senslab [37], attualmente in fase di realizzazione. Inoltre effettuare questi esperimenti richiede un tempo sicuramente maggiore rispetto all'esecuzione delle simulazioni, soprattutto perché lo scorrere del tempo non può essere accelerato nel mondo reale.

Al fine di semplificare e velocizzare più possibile questa fase del lavoro, si è deciso di utilizzare due reti sperimentali, sufficientemente grandi, che permettono di eseguire gli esperimenti e ottenere i risultati mediante una comoda interfaccia web ed evitano di dover costruire e programmare manualmente la rete. Inoltre, per lo stesso motivo, è stato scelto di ridurre il numero di metodi da comparare da cinque a tre, eliminando DistFlooder e AHBPFlooder, nel primo caso poiché preferiamo tenere JDCTFlooder per i confronti con OppFlooder, mentre nel secondo siccome AHBPFlooder ha mostrato evidenti difficoltà già in simulazione e quindi si preferisce tenere

SBAFlooder come rappresentante della categoria che sfrutta le informazioni sui vicini.

Seguendo un approccio simile a quello utilizzato al capitolo precedente, per prima cosa introduciamo brevemente le due reti sperimentali utilizzate, quindi riportiamo i risultati delle prove di connettività che sono state eseguite su entrambe le reti e sul simulatore per permettere l'acquisizione di risultati comparabili. Poi presentiamo gli obiettivi e gli scenari degli esperimenti e infine raccogliamo i parametri generali degli esperimenti eseguiti e i parametri dei metodi confrontati.

6.1 Le reti sperimentali utilizzate: Motelab e Indriya

Le reti sperimentali, utilizzate per comparare i metodi anche in scenari reali, sono state scelte sulla base di alcune necessità:

- le dimensioni della rete, sia dal punto di vista del numero dei nodi disponibili sia per quanto riguarda l'area in cui essi sono dislocati, devono essere sufficientemente grandi da permettere uno studio comparabile a quelli eseguiti in simulazione, soprattutto per quanto riguarda le complessità in gioco;
- la possibilità di eseguire liberamente un gran numero di esperimenti;
- la facilità di utilizzo della rete;
- il tipo di rete, che deve essere una WSN formata da nodi con le stesse caratteristiche di quelli simulati.

Una ricerca su internet ha permesso di trovare due WSN che soddisfano i requisiti elencati, ovvero Motelab e Indriya. Entrambe sono sufficientemente grandi e hanno dimensioni simili, sono utilizzabili mediante la medesima interfaccia web e sono composte da nodi TelosB. Le specifiche tecniche di tali nodi sono raccolte in tabella 6.1. Si è scelto di utilizzare entrambe le reti in modo da non basare la validazione dei risultati delle simulazioni sugli

Specifiche tecniche dei nodi TelosB	
Microcontrollore	Texas Instruments MSP430
Clock	8 MHz
Memoria RAM	10 Kbyte
Radio	Chipcon CC2420
Livello MAC	IEEE 802.15.4
Frequenza	2.4 GHz
Velocità	fino a 250 Kbit/s
Memoria aggiuntiva (flash)	1 Mbyte
Connettività	USB
Alimentazione	2 batterie AA
Sistema operativo	TinyOS

Tabella 6.1: Le specifiche tecniche dei nodi TelosB

esperimenti eseguiti su una sola rete reale, poiché esiste il rischio che i risultati ottenuti su tale rete possano corrispondere a un caso strano e isolato. La probabilità che ciò accada su due reti differenti è decisamente inferiore.

Per concludere, evidenziamo il fatto che è stato ovviamente necessario implementare nuovamente i tre metodi scelti per la comparazione in scenari reali. In questo caso ci si è appoggiati a TinyOS (versione 2.1.0), un sistema operativo open source basato sugli eventi, progettato per offrire una piattaforma di sviluppo per le reti di sensori, e al linguaggio nesC, che al tempo stesso è una versione semplificata ed estesa del comune C.

Motelab

Motelab [48] è una WSN sperimentale, che si trova nell'edificio di ingegneria elettrica e informatica dell'università di Harvard, appositamente pensata per supportare la ricerca nel campo delle WSN. La rete dovrebbe essere composta da 190 nodi, ma per gli esperimenti è stato possibile utilizzarne solo 97, poiché gli altri sono stati rimossi o disabilitati in seguito a problemi.

La topologia di Motelab risulta essere piuttosto particolare. I nodi, che sono disposti su tre piani dell'edificio, spesso risultano accoppiati: dei 97 nodi presenti, ben 86 fanno parte di una coppia, ovvero sono vicinissimi a

un altro nodo, mentre solo 11 sono singoli. Ci si aspetta che questa insolita topologia della rete possa produrre risultati almeno in parte differenti rispetto a quelli ottenuti con le simulazioni, che essendo frutto di una media sono rappresentativi di una rete molto uniforme.

Indriya

Così come Motelab, anche Indriya [24] è una WSN sperimentale realizzata per facilitare gli studi sulle reti di sensori in un ambiente reale. Indriya è situata su tre piani di un edificio della scuola di informatica dell'università nazionale di Singapore ed è composta da 127 nodi.

Visto che il numero di nodi disponibili in Motelab è pari a 97, si è deciso di utilizzare solo 97 dei 127 nodi di Indriya, così da eseguire gli esperimenti su due reti con lo stesso numero di nodi. La scelta dei nodi da eliminare è stata compiuta con l'obiettivo di ottenere una topologia con una distribuzione dei nodi per quanto possibile uniforme, eliminando quindi quelli particolarmente isolati e sfoltendo alcune aree molto dense. L'intento infatti è eseguire gli esperimenti su una rete reale che, al contrario di Motelab, rappresenti un caso più possibile generico e quindi simile a ciò che si ottiene facendo una media su numerose simulazioni eseguite su topologie casuali.

6.2 Le prove di connettività

Il primo passo per rendere comparabili i risultati ottenuti da Motelab e Indriya è stato, come abbiamo visto al paragrafo precedente, fare in modo che gli esperimenti vengano eseguiti con lo stesso numero di nodi. Ciò non toglie il fatto che le topologie delle due reti sperimentali considerate siano notevolmente diverse e su questo, ovviamente, non abbiamo alcun potere. Tuttavia è possibile eseguire gli esperimenti sulle due reti in condizioni simili di densità: infatti i nodi TelosB permettono di impostare la potenza della radio scegliendo un numero intero nell'intervallo da 1 a 31, dove 31 è l'impostazione relativa alla massima potenza e 1 a quella minima e ciò influisce direttamente sul raggio di trasmissione dei nodi e implicitamente sulla densità della rete così ottenuta.

Sono stati dunque eseguiti numerosi esperimenti, sia su Motelab sia su Indriya, per conoscere il numero di vicini di ogni singolo nodo, e quindi quello medio della rete, a tutte le potenze radio corrispondenti a un numero dispari. In ognuno di questi esperimenti, ogni singolo nodo invia periodicamente un semplice messaggio a tutti i suoi vicini per un tempo di 5 minuti. È stato scelto un periodo di emissione pari a 5 secondi, così da eseguire le prove di connettività in una situazione di traffico abbastanza rilevante, simile a quella presente negli esperimenti da effettuare in seguito. Per semplicità, e verificato che ciò accade piuttosto raramente, si è deciso di contare come vicino anche un nodo da cui si riceve un solo messaggio.

Ottenuti i risultati, sono state scelte quattro potenze radio per Motelab, in modo da avere quattro differenti densità medie della rete; quindi si è optato per le quattro potenze radio in Indriya che hanno permesso di avvicinarsi maggiormente al numero medio di vicini ottenuto su Motelab, conducendo esperimenti anche per i valori pari dell'intervallo adiacenti ai valori dispari migliori. Infine, su entrambe le reti, per ognuna delle potenze radio selezionata è stata eseguita una serie di otto esperimenti nell'arco dell'intera giornata, a distanza di tre ore uno dall'altro, così da poter ottenere un risultato medio affidabile. Tali prove hanno anche permesso di verificare se i risultati forniti dalle due reti sono influenzati dal momento in cui vengono eseguiti gli esperimenti, per esempio poiché durante il giorno sono presenti delle persone ed è più probabile che ci siano altre apparecchiature elettriche ed elettroniche

Rete	**Potenza radio**	**# medio vicini**
Motelab	31	17.14
	21	15.46
	15	12.77
	9	9.31
Indriya	25	17.05
	19	15.04
	13	12.60
	7	8.53

Tabella 6.2: Le potenze radio scelte con le prove di connettività

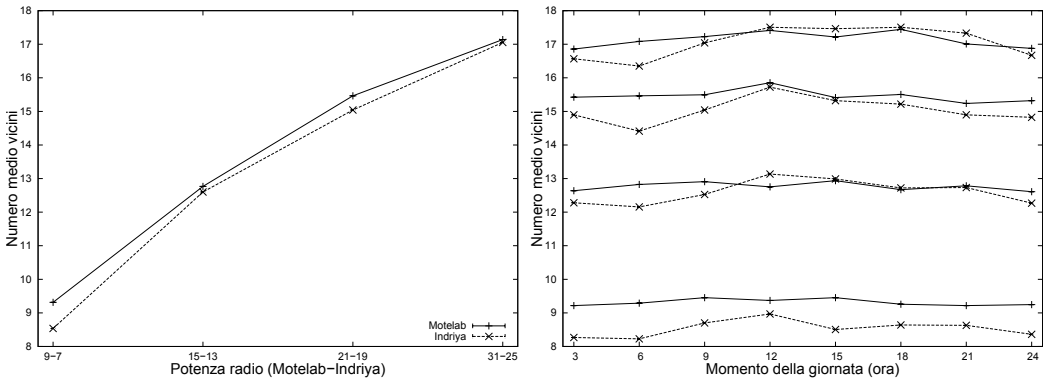

Figura 6.1: Risultati di connettività su Motelab e Indriya

in funzione.

I risultati numerici di questi test, per quanto riguarda le potenze radio selezionate, sono stati raccolti nella tabella 6.2 e presentati graficamente in figura 6.1. Il grafico di sinistra è autoesplicativo, mentre in quello di destra le quattro serie di valori corrispondono, ovviamente, alle quattro diverse potenze radio, sia per Motelab sia per Indriya. Si noti come, soprattutto per Indriya, le ore centrali della giornata sono quelle in cui si riscontra una maggiore connettività e ciò è esattamente l'opposto di ciò che ci si poteva aspettare. Analizzando più approfonditamente i risultati, è stato possibile notare che questo aumento di connettività è dovuto proprio all'aggiunta di vicini posti ai bordi del raggio di trasmissione, che però sono caratterizzati da una raggiungibilità intermittente e inaffidabile. Per sicurezza sono stati effettuati alcuni esperimenti per verificare se questi limitati divari di connettività influiscono sui risultati dell'esecuzione del broadcast: in merito non sono state riscontrate differenze significative e ciò ha permesso di eseguire gli esperimenti indipendentemente dal momento della giornata.

6.3 Parametri delle simulazioni per il confronto con i casi reali

Per poter eseguire una comparazione sensata tra le simulazioni e gli esperimenti svolti su Motelab e Indriya è necessario che le condizioni della rete

siano più simili possibile. Innanzitutto si è pensato di eseguire le simulazioni su topologie con perimetro sia quadrato sia rettangolare, visto che Motelab e Indriya hanno una forma simil rettangolare. Il rapporto tra larghezza e altezza, per il rettangolo, è stato scelto pari a 2.4, basandosi sulle planimetrie di Motelab. Inoltre, cosa più importante, gli stessi test di connettività svolti sulle reti sperimentali sono stati effettuati anche sul simulatore, in modo da ottenere le combinazioni di densità e area della rete più opportune (si noti che in questo caso, al contrario degli scenari delle simulazioni, il numero di nodi è fissato). Anche per questi test sono state eseguite venti ripetizioni su topologie differenti.

Grazie a numerosi tentativi, mirati ad approssimare sempre più il numero di nodi vicini sul simulatore a quelli in tabella 6.2, sono stati ottenuti i valori presentati nella tabella 6.3. In figura 6.2 sono invece presenti due grafici. Quello di sinistra mostra il numero medio di vicini su simulatore e sulle due reti sperimentali; si noti come la forma della rete simulata influisce, seppur in maniera limitata, sui risultati di connettività. Nel grafico di destra, invece, sull'asse delle x è presente il numero di messaggi ricevuti durante i 5 minuti di un test di connettività (nella fattispecie quello con densità pari a 340 e area 0.287, ma gli altri sono molto simili) e sull'asse delle y è presente il numero di nodi che hanno ricevuto una certa quantità di messaggi. È interessante notare come le curve relative al simulatore siano molto più ripide rispetto a quelle di Motelab e Indriya. Una curva molto ripida indica che tutti i nodi ricevono più o meno lo stesso numero di messaggi, e ciò è proprio dovuto al fatto che i risultati del simulatore sono il prodotto di una media tra molte

Densità (nodi\kmq)	**Area (kmq)**	**# medio vicini**	
		Rete quad.	Rete rett.
480	0.203	17.57	17.14
425	0.228	15.72	15.30
340	0.287	13.00	12.68
240	0.405	9.53	9.29

Tabella 6.3: Densità e area (per reti simulate) corrispondenti alle potenze radio scelte

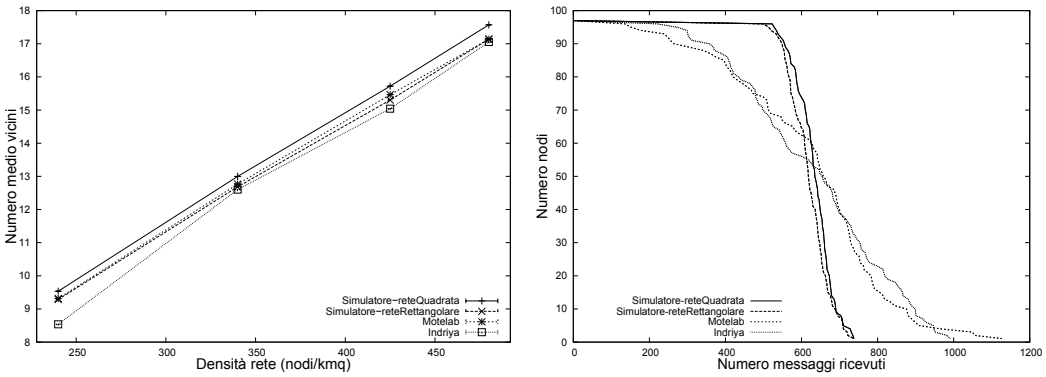

Figura 6.2: Risultati di connettività su simulatore, Motelab e Indriya

topologie differenti. La curva delle due reti sperimentali invece è molto più dolce, nonostante il tentativo compiuto su Indriya per ottenere una rete più uniforme possibile.

6.4 Obiettivi e scenari degli esperimenti

L'obiettivo degli studi sulle reti sperimentali è verificare se i risultati ottenuti con le simulazioni sono sufficientemente aderenti alla realtà. In tal caso possiamo affermare che l'ambiente di simulazione utilizzato è uno strumento su cui è possibile fare affidamento. Purtroppo, a causa delle limitazioni presentate all'inizio di questo capitolo, non è possibile adottare tutti gli scenari e testare tutti i metodi coinvolti nella campagna simulativa. Possiamo usare le reti sperimentali per valutare OppFlooder, JDCTFlooder e SBAFlooder in situazioni in cui varia:

- la densità della rete, grazie all'utilizzo di differenti livelli di potenza della radio;
- la congestione della rete, che possiamo controllare con la stessa facilità con cui l'abbiamo fatto in simulazione.

Anche in questo caso si è deciso di scegliere un certo numero di valori sia per la densità della rete (qui sotto forma di potenze radio) sia per la frequenza di invio dei messaggi; tali valori sono raccolti in tabella 6.4 e quelli evidenziati in grossetto corrispondono al valore assunto da un parametro quando

Potenza radio	Motelab	9	15	**21**	31
	Indriya	7	13	**19**	25
Frequenza invio (msg/s)			**0.05**	0.1	0.5

Tabella 6.4: I parametri su cui sono basati gli scenari degli esperimenti reali

l'altro varia. Dato che eseguire gli esperimenti sulle reti sperimentali richiede più tempo rispetto che al simulatore, è stato scelto un numero inferiore di possibilità per i due parametri.

Si noti che gli esperimenti equivalenti eseguiti sul simulatore considerano le stesse frequenze d'invio dei messaggi presenti nella tabella 6.4, mentre al posto delle potenze radio ci sono le quattro combinazioni densità-area viste in tabella 6.3.

6.5 Parametri generali degli esperimenti e parametri dei metodi

In quest'ultimo paragrafo presentiamo brevemente, così come è stato fatto per le simulazioni, i parametri comuni a tutti gli test eseguiti sulle reti sperimentali. Parametri e rispettivi valori sono anche riportati in tabella 6.5, mentre nell'elenco seguente è data una breve spiegazione per ciascuno di essi:

- tempo di simulazione; questo parametro è stato accorciato di 5 minuti rispetto ai test al simulatore, per ridurre i tempi necessari a eseguire tutti gli esperimenti; i primi 3 minuti sono usati, solo da SBAFlooder,

Parametro	**Valore**
Tempo di simulazione	13 minuti
# nodi sorgente messaggi	20
Dimensione payload dati	16 byte
# ripetizioni	10

Tabella 6.5: I parametri comuni a tutti gli esperimenti

per costruire le informazioni topologiche, mentre i successivi 10 sono quelli in cui avviene il broadcast;

- numero nodi sorgente; come nel caso delle simulazioni i nodi sorgente sono 20;
- dimensione dei dati contenuti in ogni messaggio di broadcast; anche questo parametro è rimasto invariato, ovvero 16 byte;
- numero di ripetizioni per ogni situazione; tale parametro è stato fissato a 10 e visto che non è possibile variare la topologia si è scelto di cambiare l'insieme dei 20 nodi che fungono da sorgente per i messaggi di broadcast.

Si noti che, per le reti sperimentali, non abbiamo a disposizione un valore unico e certo del raggio di trasmissione. Inoltre si evidenzia che anche per le simulazioni effettuate per il confronto con gli esperimenti reali si è scelto di adottare una durata pari a 13 minuti, mentre il numero di ripetizioni è rimasto lo stesso, ovvero 20 (ovviamente le simulazioni sono eseguite in condizioni di rete fissa).

Per quanto riguarda i parametri di OppFlooder, JDCTFlooder e SBAFlooder (tabella 6.6) si è deciso di utilizzare i valori scelti al paragrafo 5.4, tranne in un caso. L'intervallo dei valori del RSSI per i nodi TelosB infatti

Metodo	**Parametro**	**Valore**
OppFlooder	T_{max}	1 s
	$RSSI_{max}$	-10.0
JDCTFlooder	T_{max}	1 s
	$RSSI_{max}$	-40.0
	C	2
SBAFlooder	T_{max}	1 s
	T_c	60 s
	T_t	120 s

Tabella 6.6: I parametri dei metodi negli esperimenti reali

non è esattamente uguale a quello del simulatore: i valori ottenuti non appartengono più all'intervallo da 50 a -45, ma sono compresi tra 20 e -50 e sono dei numeri interi. Si è deciso di lasciare invariata la soglia di OppFlooder, poiché le differenze tra i valori più grandi sono comunque piuttosto limitate, ma di modificare leggermente quella sul RSSI di JDCTFlooder, portandola da -37.5 a -40.0, concordemente all'ampliamento dei valori del RSSI verso il basso.

Capitolo 7

Risultati

In questo capitolo sono raccolti tutti i risultati ottenuti grazie allo svolgimento delle ricerche, simulazioni ed esperimenti reali presentati nei capitoli precedenti. Tali risultati sono ovviamente accompagnati dalle considerazioni che emergono dalla loro analisi. In definiva, tutto il materiale inserito nel presente capitolo dovrebbe fornire le evidenze necessarie a raggiungere il duplice obiettivo di questa tesi di laurea.

Presentiamo innanzitutto gli indici di prestazioni che sono stati considerati per valutare il comportamento dei metodi comparati, dato che saranno utilizzati largamente nel seguito. Per ognuno degli indici indichiamo il nome, cosa rappresenta e come viene calcolato:

- copertura rete: rappresenta la percentuale media di nodi coperti dal broadcast ed è definita come $MSG_{ric}/\left[MSG_{inv} \cdot (NUM_{nodi} - 1)\right]$, dove MSG_{ric} è la somma dei messaggi ricevuti (repliche escluse) da tutti i nodi, MSG_{inv} è la somma dei messaggi introdotti nella rete dai nodi sorgente e NUM_{nodi} è il numero complessivo di nodi; si noti infatti che, in caso di copertura totale ogni messaggio introdotto viene ricevuto da tutti i nodi tranne uno, ovvero la sorgente che l'ha generato, e in questo caso l'indice assume il valore 1.

- ritrasmissioni: è la percentuale media di nodi che trasmettono un messaggio di broadcast (sorgente compresa) e si ottiene con la formula $MSG_{tras}/\left(NUM_{nodi} \cdot MSG_{inv}\right)$, dove MSG_{tras} è il numero totale di

messaggi trasmessi; ovviamente $NUM_{nodi} \cdot MSG_{inv}$ è il numero di trasmissioni che vengono effettuate con il metodo flooding.

- ritrasmissioni inutili: è la percentuale di trasmissioni che non sono utili, in quanto non coprono alcun nodo ancora non raggiunto dal broadcast; questo indice può essere facilmente calcolato se sono univocamente identificabili e noti tutti i messaggi trasmessi e tutti quelli ricevuti da almeno un nodo non ancora coperto dal broadcast.

- costo broadcast: è un indice che rappresenta l'unione dell'efficacia e dell'efficienza del broadcast attraverso il costo medio, in byte, necessario a consegnare ogni singolo messaggio di broadcast; è definito come $(B_b + B_c) / MSG_{ric}$, dove B_b e B_c sono rispettivamente la somma dei byte trasmessi, misurati in uscita dalla radio, per i messaggi di broadcast e per gli eventuali messaggi di controllo.

- traffico: è il traffico medio, in Kbyte/s, presente sulla rete e si calcola come $(B_b + B_c) / (1024 \cdot T_{broadcast})$, dove 1024 serve per passare da byte a Kbyte e $T_{broadcast}$ è la durata, in secondi, della parte di simulazione/esperimento in cui avviene l'esecuzione del broadcast.

- latenza: rappresenta il tempo medio necessario affinché un messaggio di broadcast venga ricevuto; questo valore viene direttamente fornito dal simulatore, mentre non è stato possibile calcolarlo negli esperimenti reali.

Gli indici ritenuti maggiormente significativi, per i nostri scopi, sono tre: la copertura rete, perché rappresenta le prestazioni, il traffico, come indicatore del "prezzo" pagato per eseguire il broadcast, e infine il costo broadcast, siccome cerca di sintetizzare proprio questo rapporto prezzo/prestazioni. Gli altri sono stati introdotti perché, in alcune situazioni, rivelano interessanti indicazioni prestazionali e possono fornire lo spunto per importanti considerazioni.

Prima di iniziare la presentazione dei numerosi risultati raccolti, è anche utile sottolineare che la maggior parte di essi è accompagnata da una stima

della loro attendibilità: in particolare viene usato l'intervallo di confidenza al 95%, ovvero si indica l'intervallo, intorno al valore medio rilevato dal campione scelto (ovvero gli esperimenti/simulazioni effettuate), che contiene il vero valore medio con probabilità del 95%. Si noti che, in alcuni casi, l'intervallo può essere così piccolo da non essere visibile, poiché coperto dal punto del grafico a cui si riferisce.

La rimanente parte di questo capitolo è stata convenientemente divisa in tre sezioni: nella prima i risultati della ricerche di minimo teorico sono raccolti e utilizzati per valutare l'efficienza dei metodi di broadcast considerati, rispetto al caso migliore possibile; nella seconda sono presentati i risultati dell'estesa campagna simulativa, attraverso cui sono stati comparati i cinque metodi scelti; infine nella terza vengono confrontati i risultati delle simulazioni con quelli degli esperimenti sulle reti reali.

7.1 I risultati delle ricerche di minimo teorico

Per valutare l'efficienza dei cinque metodi scelti rispetto a quella di un ipotetico metodo perfetto, capace di raggiungere la massima copertura della rete con il minor numero di trasmissioni, nel capitolo 4 è stato proposto un algoritmo, chiamato MinBroadcast, capace di trovare un Minimum Connected Dominating Set e quindi il numero minimo di trasmissioni necessarie. MinBroadcast è stato quindi utilizzato per ricercare tale valore per ognuna delle venti topologie casuali generate da OMNeT++ e utilizzate per eseguire le simulazioni nello scenario base con nodi fissi (cfr. paragrafo 5.2). Si noti che ognuna di queste topologie è formata da 60 nodi. Visto che la ricerca di questo valore è influenzata non solo dalla topologia della rete, ma anche dal nodo sorgente del messaggio, si è deciso di effettuare le ricerche, su ogni topologia, prendendo ognuno dei venti nodi a cui, nelle simulazioni, è stato affidato il compito di generare i messaggi di broadcast. Inoltre, visto che il raggio di trasmissione rilevato nelle simulazioni non è un valore preciso e affidabile, per le ragioni presentate nel paragrafo 5.1, si è ritenuto opportuno ripetere le ricerche per quattro raggi di trasmissione differenti, ovvero 105, 100, 95 e 90 metri.

Topologia	# minimo trasmissioni Raggio t. (m)				Copertura rete Raggio t. (m)			
	105	100	95	90	105	100	95	90
1	13.35	14.6	17.3	14.85	1	1	1	0.792
2	13.25	13.55	14.8	16.75	1	1	0.935	0.935
3	12.35	13.65	15.65	17.35	1	1	1	1
4	12.65	13.7	16.35	17.55	1	1	1	1
5	15.3	15.75	14.2	15.15	1	0.935	0.843	0.843
6	15.65	16.35	12.55	15.65	1	0.983	0.801	0.801
7	13.45	15.45	16.5	16.6	1	1	1	1
8	13.2	14.55	15.6	19.4	1	1	1	1
9	13.45	15.15	16.25	18.65	1	1	1	1
10	10.7	13.2	13.55	14.5	1	1	1	1
11	14.55	14.6	16.4	18.55	1	1	1	1
12	12.6	12.85	17.25	15.3	1	1	1	0.95
13	10.2	12.65	13.6	15.65	0.905	0.905	0.889	0.889
14	14.45	15.6	18.3	13.1	1	1	0.983	0.775
15	13.3	14.3	14.5	16.4	1	1	1	0.983
16	13.65	16.35	18.45	18.5	1	1	1	1
17	14.5	15.5	14.55	17.5	1	1	1	1
18	15.35	17.25	18.45	21.25	1	1	1	1
19	12.3	12.5	14.25	15.3	1	1	1	1
20	11.2	12.15	14.2	14.55	1	1	1	0.983
Media	13.27	14.49	15.69	16.63	0.995	0.991	0.973	0.948

Tabella 7.1: Risultati delle ricerche di minimo teorico

In tabella 7.1 sono raccolti i risultati ottenuti dalle ricerche effettuate. Tali risultati non riguardano solo il numero minimo di trasmissioni ma anche i valori di copertura della rete. Si è deciso di presentare anche questi ultimi, siccome in svariate situazioni la rete risulta essere partizionata (con uno o pochi nodi separati dagli altri) e ciò generalmente influisce sui risultati relativi al numero minimo di trasmissioni. Al fine di non appesantire inutilmente la lettura della tabella, sono stati riportati solamente i risultati medi delle venti topologie con ciascuno dei quattro raggi di trasmissione ed è stata quindi calcolata la media su tali risultati. I quattro numeri ottenuti sono usati per valutare l'efficienza dei metodi di broadcast.

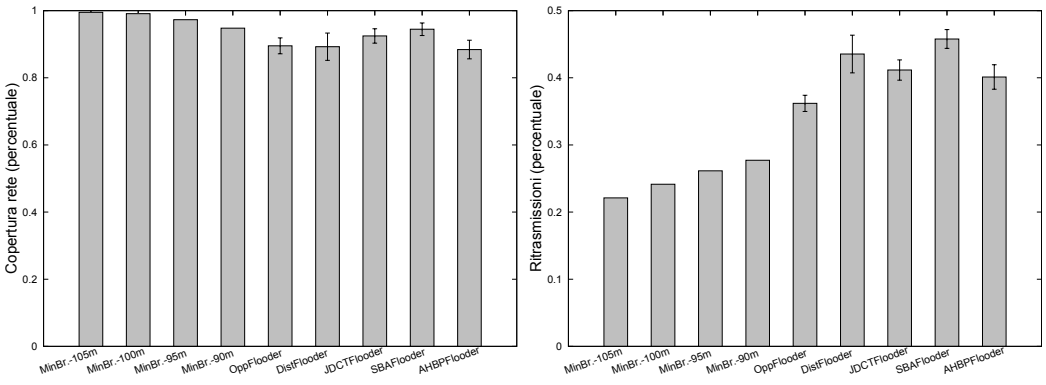

Figura 7.1: Confronto tra metodi e ricerche di minimo teorico

Osservando i risultati delle ricerche di minimo teorico, risulta evidente, quanto ovvio, che il numero minimo di trasmissioni cresce al diminuire del raggio di trasmissione, a meno di eventuali partizionamenti della rete. Inoltre, scorrendo i valori nella tabella, è piuttosto facile notare come, a parità di raggio di trasmissione, il numero minimo di trasmissioni può variare notevolmente a seconda della topologia: rispetto al caso con il valore più basso, quello all'altro estremo richiede un numero minimo di ritrasmissioni addirittura superiore del 62% quando il raggio di trasmissione è pari a 90 metri, mentre con gli altri, in ordine crescente, l'aumento percentuale è del 47%, 42% e 53%. Tali differenze sono rilevate, anche se in misura più limitata, pure per quanto riguarda la copertura della rete: con il raggio di trasmissione minimo si passa da un caso migliore con copertura completa a un caso peggiore con quasi un quarto di rete, in media, non raggiunta dai messaggi. Questo è uno dei motivi principali per cui, per valutare e comparare i metodi scelti, le simulazioni sono state ripetute su molte topologie differenti.

Utilizziamo ora i risultati medi di tabella 7.1 per valutare l'efficienza dei cinque metodi considerati. Per ottenere l'indice percentuale di ritrasmissioni a partire dai numeri in tabella è sufficiente dividere i numeri minimi di ritrasmissioni ottenuti per 60, che è il numero complessivo di nodi che costituiscono le topologie su cui sono state effettuate le ricerche di minimo teorico.

I due grafici di figura 7.1 mostrano come i risultati ottenuti con MinBroadcast siano migliori rispetto a quelli di qualsiasi metodo in simulazione. Ciò

non deve sorprendere, anzi è un risultato assolutamente atteso. Per quanto riguarda la copertura della rete, i cinque metodi offrono prestazioni buone, abbastanza simili tra loro e che non si discostano molto dal peggiore dei risultati teorici. Le differenze sono più evidenti nel caso delle ritrasmissioni: OppFlooder, che offre i risultati migliori, richiede che in media trasmettano il 36.2% dei nodi della rete nello scenario base, mentre per SBAFlooder questa percentuale sale addirittura al 45.8%; il minimo teorico, con il raggio di trasmissione più piccolo, è fissato al 27.7%, mentre con raggio di trasmissione pari a 100 metri, cioè equivalente a quello rilevato nelle simulazioni, scende al 24.2%.

OppFlooder quindi, almeno nella condizione di rete considerata, è quello che permette di avvicinarsi maggiormente all'efficienza massima ottenibile, da cui si discosta per un surplus di ritrasmissioni del 31% circa. È interessante notare che, nelle simulazioni sullo scenario base, le ritrasmissioni inutili rilevate per OppFlooder sono pari al 27.8% del totale: eliminando tale porzione, che in OppFlooder si ritiene dipenda soprattutto dalle ritrasmissioni effettuate dai nodi sul perimetro della rete, si scenderebbe dal 36.2% al 26.2%, che è circa il valore rilevato con MinBroadcast quando il raggio di trasmissione è pari a 95 metri.

In conclusione, questi risultati dimostrano che i cinque metodi scelti, e in particolare OppFlooder, permettono di ottenere una buona efficienza, in termini di numero di messaggi trasmessi, soprattutto tenendo conto che, a differenza di MinBroadcast, tali metodi devono operare in un sistema che è distribuito e che non gode delle semplificazioni introdotte per effettuare le ricerche di minimo teorico. Gli stessi risultati mettono anche in evidenza come ci sia ancora un po' di spazio per possibili miglioramenti.

7.2 I risultati delle simulazioni: comparazione dei metodi scelti

In questo paragrafo sono raccolti i numerosi risultati ottenuti compiendo le simulazioni in tutte le condizioni della rete previste dagli scenari considerati

(si veda la tabella 5.2). Per ragioni di praticità e chiarezza espositiva, si è deciso, nella presentazione, di separare i risultati relativi agli studi sugli scenari con nodi fissi da quelli con nodi in movimento e infine dall'indagine in cui viene valutato l'effetto della mobilità sul comportamento dei metodi scelti.

7.2.1 I risultati negli scenari con nodi fissi

In assenza di mobilità, SBAFlooder e AHBPFlooder dovrebbero permettere di ottenere migliori prestazioni rispetto agli altri tre metodi considerati, poiché si ritiene che la conoscenza dei vicini dovrebbe essere tendenzialmente corretta e quindi consentire di raggiungere una buona copertura della rete, limitando quanto possibile, soprattutto nel caso di AHBPFlooder, il numero di ritrasmissioni. Inoltre, la frequenza di emissione dei messaggi di controllo è piuttosto bassa e dunque il traffico aggiuntivo necessario a creare e mantenere le topologie locali dovrebbe risultare trascurabile.

La presentazione dei risultati delle simulazioni inizia dallo scenario base, per poi proseguire con i casi in cui varia la densità dei nodi, l'area della rete e la frequenza di invio dei messaggi di broadcast.

Lo scenario base

Come scenario base è stata scelta una rete con densità media (300 nodi/kmq), area abbastanza limitata (0.2 kmq) e una frequenza di invio dei messaggi bassa (un messaggio ogni 20 secondi). Vediamo come si comportano i cinque metodi in queste condizioni.

I risultati (figura 7.2) mostrano che, per quanto riguarda la copertura della rete, tutti i metodi considerati esibiscono buone prestazioni: SBAFlooder e JDCTFlooder risultano essere i migliori, ma ciò è controbilanciato, soprattutto nel caso di SBAFlooder, da un più elevato numero di ritrasmissioni. Osservando il grafico relativo al costo del broadcast, risulta abbastanza evidente che OppFlooder è il metodo che esibisce la maggior efficienza: esso infatti, pur non fornendo la copertura di rete massima, richiede un numero di ritrasmissioni inferiore rispetto a tutti gli altri e conseguentemente genera un traffico più limitato. Nel grafico del traffico sono distinti i vari contri-

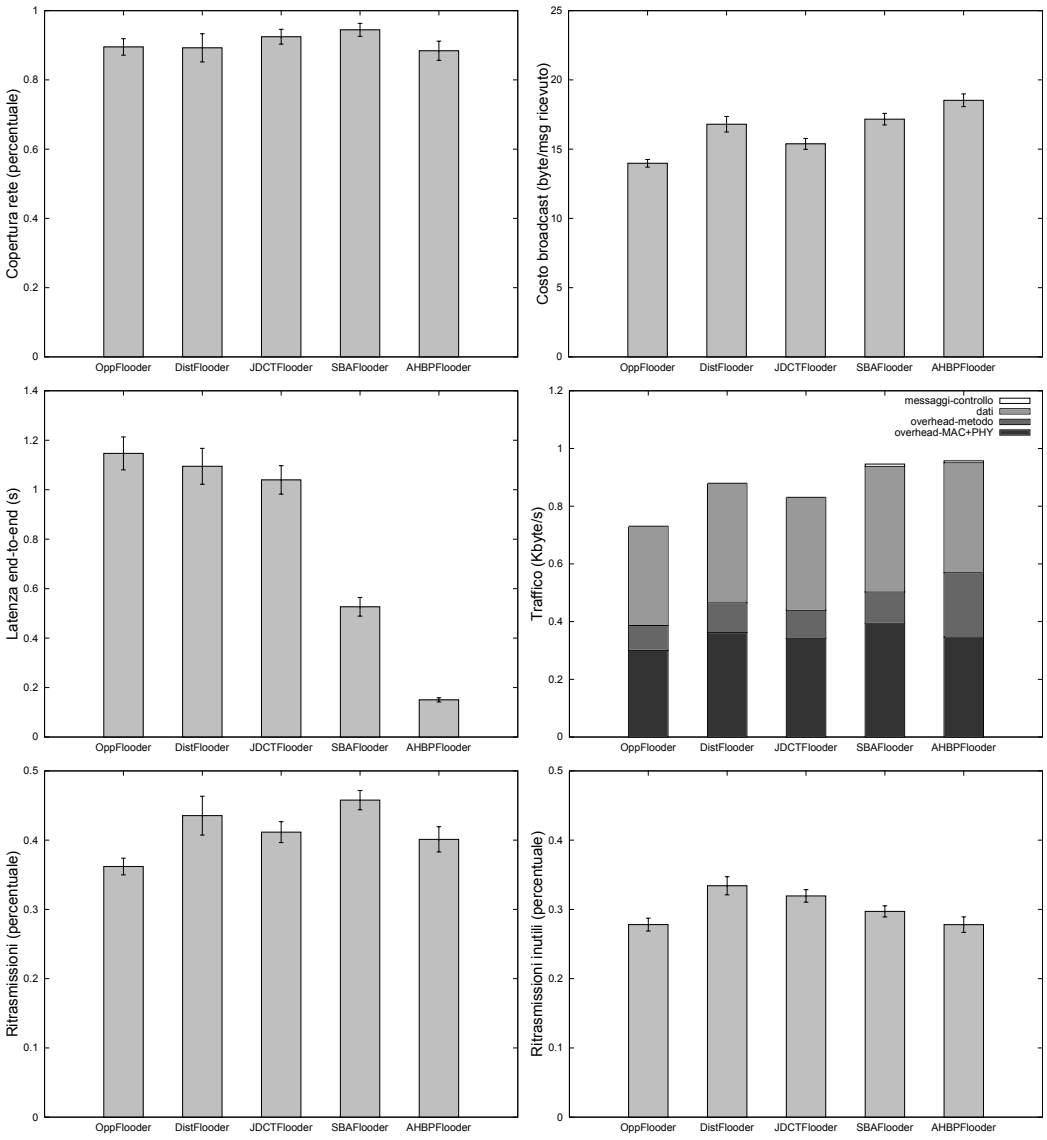

Figura 7.2: Risultati simulazioni - scenario base (nodi fissi)

buti: partendo dal basso, con *overhead-MAC+PHY* si intende l'overhead generato dai livelli MAC e fisico, che è indipendente dai metodi, sia per i messaggi di broadcast sia per quelli di controllo; con *overhead-metodo* invece si fa riferimento alle informazioni aggiunte nei messaggi di broadcast dai metodi; infine *dati* e *messaggi-controllo* sono relativi al traffico da imputare rispettivamente al contenuto dei messaggi di broadcast e di controllo. A proposito del traffico, è interessante evidenziare che tutti i metodi, escluso

AHBPFlooder, includono in ogni messaggio di broadcast le stesse identiche informazioni; l'overhead di tali metodi, in percentuale rispetto al traffico totale, risulta quindi identico. AHBPFlooder invece inserisce un maggior quantitativo di informazioni nei messaggi di broadcast e questo overhead lo penalizza in maniera abbastanza rilevante. Infine l'influenza dei messaggi di controllo sul traffico totale è decisamente trascurabile, come ci si aspettava. Per quanto riguarda la latenza, è palese che AHBPFlooder sia il metodo che garantisce, in media, il minor tempo di consegna dei messaggi; SBAFlooder si comporta meglio dei rimanenti tre medodi, poiché il calcolo del tempo di attesa favorisce la ritrasmissione da parte dei nodi con il maggior numero di vicini, con un ritardo generalmente inferiore al mezzo secondo (cfr. paragrafo 3.4); OppFlooder, DistFlooder e JDCTFlooder sono caratterizzati da latenze medie piuttosto simili, che in caso di necessità potrebbero essere portate al livello di SBAFlooder senza problemi, come dimostrato nei test effettuati per la scelta dei parametri dei metodi, nel paragrafo 5.4.

Nonostante l'assenza di mobilità, in questo scenario è OppFlooder il metodo da preferire (latenza a parte), in quanto risulta più efficiente rispetto a tutti gli altri: ciò è indicato innanzitutto dal costo del broadcast ed è supportato anche dal traffico generato e dalle ritrasmissioni complessive e inutili.

Lo scenario con densità variabile

Presentiamo ora i risultati relativi alle simulazioni in cui cambia la densità della rete e vediamo quale impatto ha tale variazione sulle prestazioni dei metodi condiderati.

I grafici di figura 7.3 mostrano come gli andamenti dei risultati dei vari metodi, ottenuti in differenti situazioni di densità della rete, sono in generale piuttosto simili. La copertura della rete, com'è logico aspettarsi, tende ad aumentare con il crescere della densità, poiché diventa sempre più improbabile che nella rete ci siano uno o più nodi isolati dagli altri. I risultati di copertura sono davvero scarsi alla minima densità e ciò è sicuramente dovuto alla presenza di uno o più partizionamenti nelle topologie su cui sono stati eseguiti i test; queste particolari condizioni di rete influiscono anche sugli altri indici di prestazione. La latenza di rete, per esempio, rimane piuttosto stabile, escluso il caso a bassa densità, con valori che decrescono leggermente all'aumentare

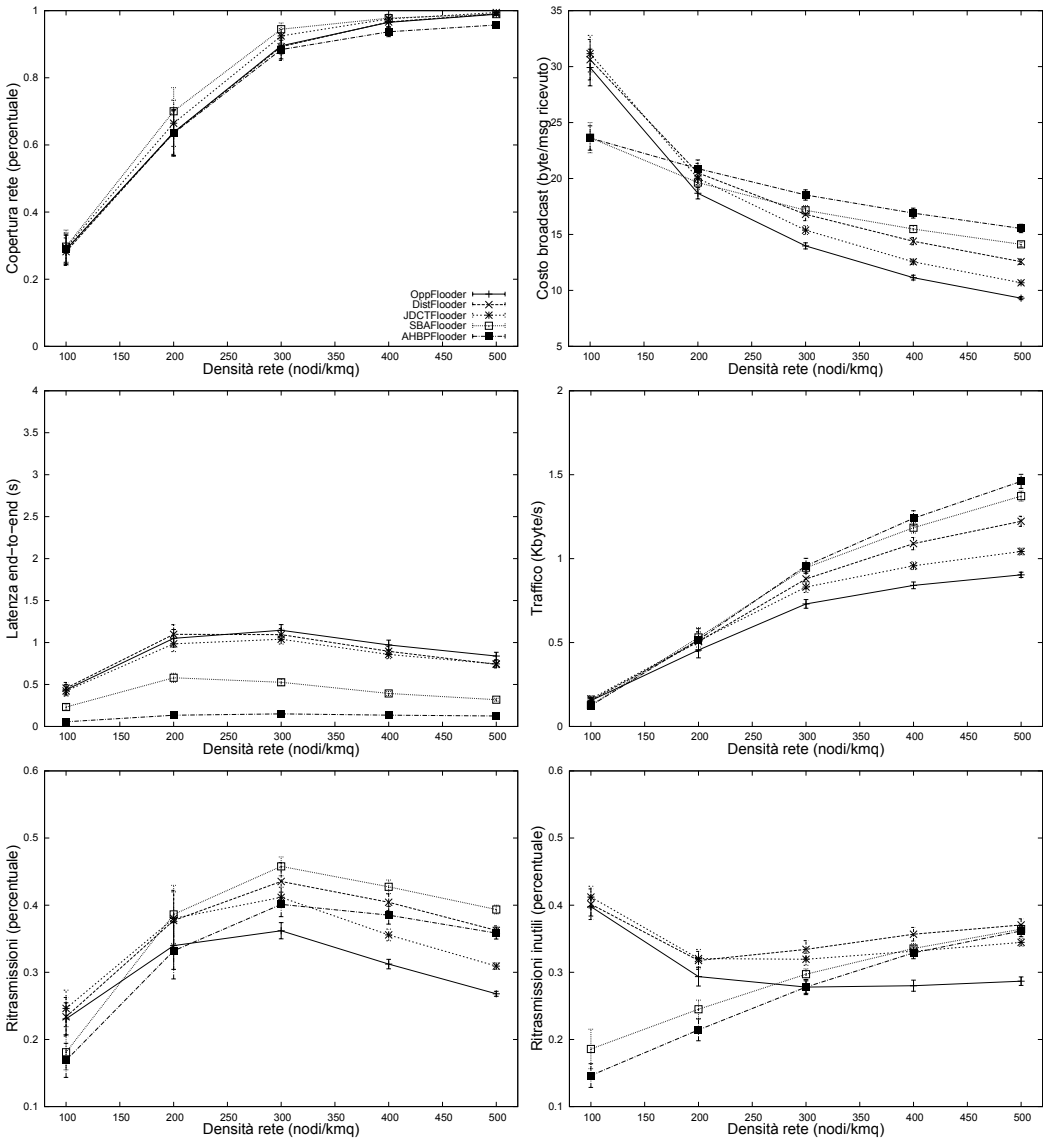

Figura 7.3: Risultati simulazioni - densità variabile (nodi fissi)

della densità; ciò vale per tutti i metodi tranne AHBPFlooder. È interessante notare che SBAFlooder e AHBPFlooder sono decisamente più efficienti, a parità di (scarsa) copertura, nel caso a densità minore: essi infatti usano le informazioni topologiche per evitare molte ritrasmissioni inutili, mentre gli altri tre metodi non possono farlo e il problema delle ritrasmissioni sui bordi della rete è accentuato dal partizionamento della stessa. Tuttavia, al crescere della densità, la loro efficienza prima si allinea e poi diventa addirit-

tura peggiore rispetto a quella di OppFlooder, DistFlooder e JDCTFlooder. Il numero di ritrasmissioni, limitandoci ai tre casi con copertura della rete elevata, tende a diminuire leggermente e ciò e dovuto al fatto che, a densità maggiori, è più probabile che siano presenti nodi nei punti più favorevoli per un broadcast efficiente. Il costo del broadcast ovviamente diminuisce al salire della densità, mentre il traffico tende ad aumentare (a basse densità la rete e partizionata), seppur seguendo un andamento sublineare.

A densità medio basse la scelta ottimale sembra essere SBAFlooder, poiché permette di ottenere una copertura maggiore rispetto agli altri metodi con un costo decisamente inferiore (pari a quello di AHBPFlooder). Nei contesti in cui la densità è medio alta invece è OppFlooder a fornire i migliori risultati, sopratutto per quanto riguarda l'efficienza del broadcast.

Lo scenario con area variabile

L'area della rete, a parità di raggio di trasmissione, è un parametro che influenza direttamente il numero medio di hop necessari a diffondere un messaggio. Risulta quindi piuttosto sensato ritenere che questo parametro incida maggiormente su alcuni indici di prestazione, come per esempio la latenza, mentre sia praticamente ininfluente rispetto ad altri. Si noti che, mantenendo fissa la densità e aumentando l'area si incrementa ovviamente anche il numero dei nodi.

I risultati ottenuti dalle simulazioni in questo scenario, presentati in figura 7.4, confermano quanto appena detto. La latenza della rete infatti, al crescere dell'area, aumenta con un andamento sublineare per tutti i metodi: ciò dipende dal fatto che il numero medio di hop non cresce linearmente con l'area. Anche il traffico generato cresce in maniera molto evidente (si noti il cambio di scala per questo grafico), poiché ingrandendo l'area aumenta anche il numero di nodi che devono ritrasmettere il messaggio. La copertura della rete rimane invece tendenzialmente elevata e stabile, con un leggero calo sono per quanto riguarda l'area più grande; ciò vale anche per il costo del broadcast, che cala in maniera molto limitata per OppFlooder, DistFlooder e JDCTFlooder, mentre sale in maniera altrettanto modesta per SBAFlooder e AHBPFlooder, il cui funzionamento evidentemente subisce, seppur lievemente, gli effetti della maggiore complessità introdotta all'aumentare dell'area.

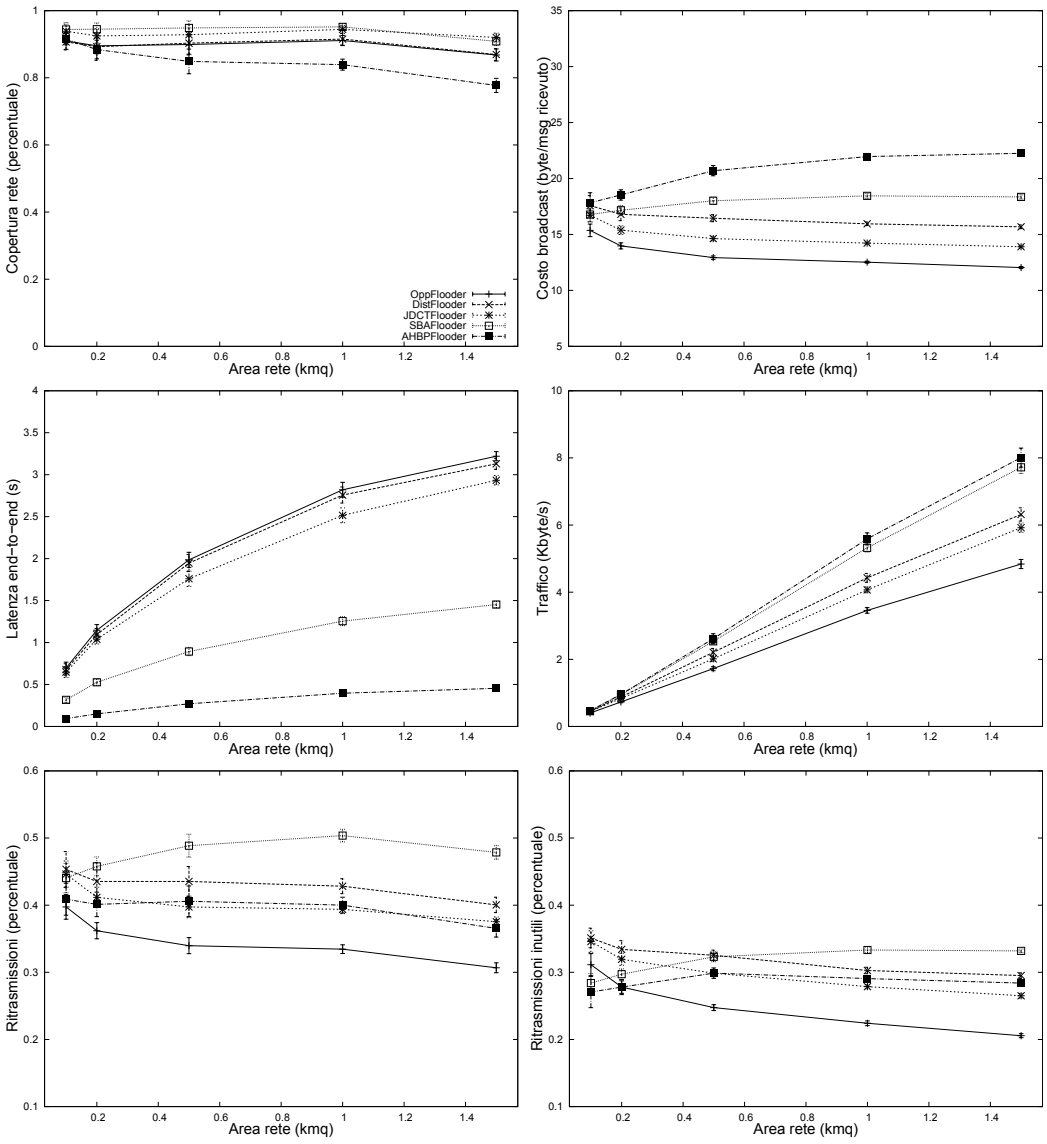

Figura 7.4: Risultati simulazioni - area variabile (nodi fissi)

Tale andamento è caratteristico anche delle ritrasmissioni inutili, mentre la percentuale totale di ritrasmissioni si assesta tra lo stabile e il decrescente per tutti i metodi tranne SBAFlooder, in cui tende prima ad aumentare e a diminuire solo alla densità maggiore.

Anche in questo scenario, escludendo la latenza, OppFlooder dimostra di essere il metodo da preferire: pur essendo superato da SBAFlooder e JDCT-Flooder per quanto riguarda la copertura della rete, è il metodo che garantisce

il minore costo di broadcast, pari a circa l'88% rispetto a JDCTFlooder e il 71% rispetto a SBAFlooder. OppFlooder inoltre è anche il metodo che richiede meno trasmissioni e quindi genera complessivamente meno traffico.

Lo scenario con frequenza d'invio dei messaggi variabile

Analizziamo infine l'impatto della congestione della rete, ottenuta attraverso il cambiamento della frequenza di invio dei messaggi di broadcast. È chiaro che questo parametro influisce direttamente sul traffico generato. Inoltre, in una rete congestionata, dovrebbe aumentare la probabilità che avvengano collisioni e ciò può indubbiamente aver effetto sia sulla copertura media della rete sia sul costo del broadcast, poiché le collisioni provocano lo "spreco" di messaggi.

Anche in questo caso risulta, dai grafici di figura 7.5, che alcuni indici di prestazione sono influenzati più di altri dalla modifica di un singolo parametro della rete. Ovviamente è il traffico generato a subire l'impatto più diretto ed evidente (ancora una volta è stato necessario un cambio di scala del grafico); si noti che, in questo caso, l'ampliarsi della forbice nei risultati, al crescere della frequenza d'invio, dipende dalla differente percentuale di ritrasmissioni di ogni singolo metodo: infatti, se in una rete di 60 nodi (di cui 20 sorgente dei messaggi) e frequenza d'invio di un messaggio ogni 20 secondi, un metodo con il 35% di ritrasmissioni trasmette, in media, 21 messaggi al secondo e uno con il 45% di ritrasmissioni 27 messaggi al secondo (cioè circa il 28% in più), decuplicando la frequenza il numero di messaggi al secondo, in media, diventa 210 per il primo e 270 per il secondo, che è ancora circa il 28% in più, ma in assoluto si passa da una differenza di 6 messaggi a una di 60. I risultati mostrano anche come la copertura della rete e il costo del broadcast subiscono l'effetto della congestione della rete e delle collisioni. Per quanto riguarda il costo del broadcast, è interessante notare come per SBAFlooder e AHBPFlooder, alle frequenze di invio inferiori, esso sia influenzato dall'emissione dei messaggi di controllo, poichè essi assumono un certo rilievo, in percentuale, rispetto al traffico dati. Infine, anche il numero di ritrasmissioni tende a diminuire, seppur lievemente, a causa delle collisioni, mentre le ritrasmissioni inutili rimangono stabili o crescono in maniera limitata, così come anche la latenza della rete, anch'essa influenzata

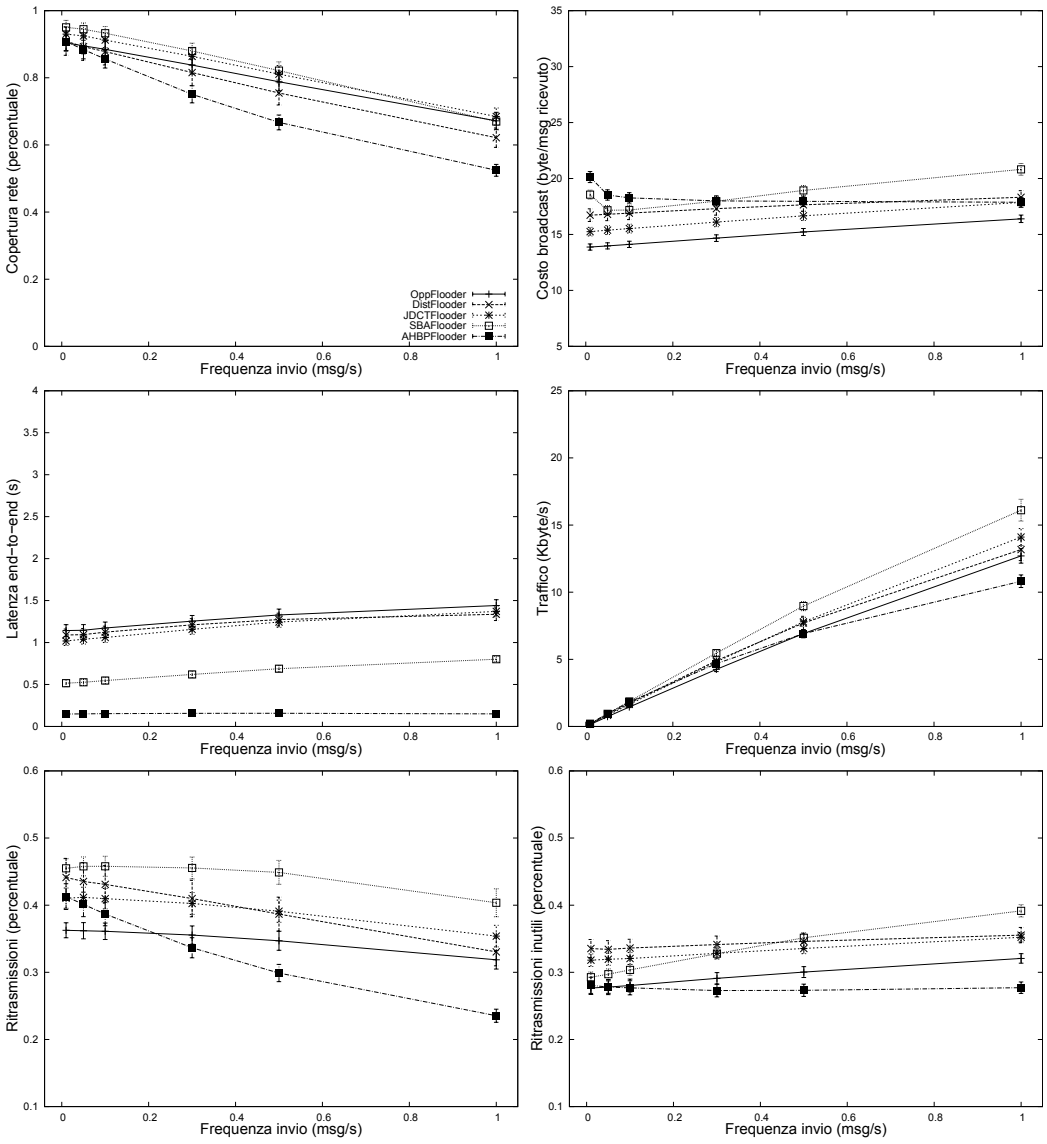

Figura 7.5: Risultati simulazioni - frequenza invio variabile (nodi fissi)

dalle collisioni.

OppFlooder risulta essere, ancora una volta, il metodo più efficiente e ciò è dimostrato dal minor costo del broadcast in tutte le situazioni; in particolare si può dire che OppFlooder sia particolarmente consigliabile in situazioni di elevata congestione della rete, poiché garantisce una copertura della rete pari a quella ottenuta da SBAFlooder e JDCTFlooder, ma a un costo inferiore.

7.2.2 I risultati negli scenari con nodi mobili

Le simulazioni in un contesto mobile, effettuate poiché si ritiene particolarmente interessante valutare i cinque metodi scelti in condizioni in cui l'utilizzo delle WSN diventerà sempre più una realtà, dovrebbero introdurre un ulteriore grado di difficoltà. Ci si aspetta che i problemi più rilevanti riguardino i due metodi che sfruttano la conoscenza dei vicini perché, come già fatto notare nel capitolo 3.6.1, l'affidabilità di tali informazioni può essere minata in maniera rilevante proprio dallo spostamento dei nodi. Si noti che il modello random waypoint utilizzato per la mobilità, quando applicato a reti non toroidali, fa in modo che i nodi tendano a raggrupparsi al centro della rete e ciò, come vedremo, ha una certa influenza sui risultati ottenuti.

Come nel paragrafo precedente, anche in questo caso sono presentati prima i risultati relativi allo scenario base e quindi quelli delle simulazioni in cui varia uno dei parametri della rete.

Lo scenario base

L'unica differenza introdotta in questo scenario base riguarda appunto la mobilità dei nodi: invece di essere completamente fermo, ogni nodo si sposta all'interno dell'area della rete con una velocità variabile, compresa tra 0 e 2 metri al secondo.

Rispetto alla situazione con i nodi fissi i risultati, raccolti con l'esecuzione delle simulazioni in questo scenario base e presentati in figura 7.6, mostrano alcune interessanti differenze. In primo luogo, la copertura della rete cresce lievemente per tutti i metodi, escluso AHBPFlooder; questo effetto dipende dal raggruppamento dei nodi al centro della rete, dovuto all'utilizzo del modello random waypoint per la mobilità. Tale raggruppamento causa inoltre una riduzione, più o meno marcata a seconda del metodo, della percentuale di trasmissioni, mentre il numero di ritrasmissioni inutili cresce a causa del movimento dei nodi. Anche la netta diminuzione della latenza, che risulta addirittura dimezzata per OppFlooder, DistFlooder e JDCTFlooder, è provocata dalla concentrazione di nodi al centro della rete. Il minor numero di trasmissioni influisce, ovviamente, sul traffico generato per l'invio dei messaggi di broadcast, che quindi diminuisce visibilmente per OppFlooder,

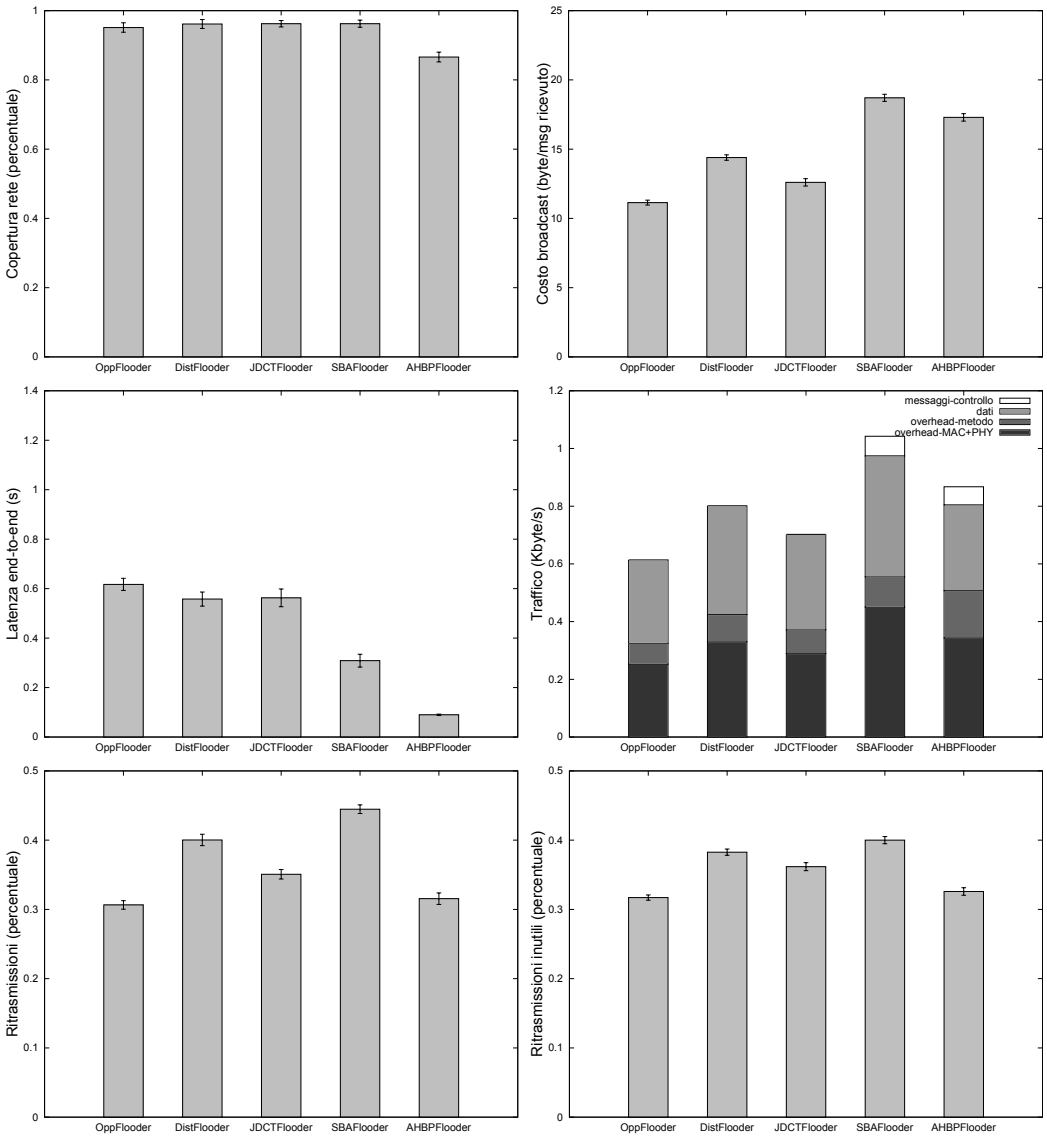

Figura 7.6: Risultati simulazioni - scenario base (nodi mobili)

DistFlooder e JDCTFlooder, mentre SBAFlooder e AHBPFlooder sono penalizzati dal maggiore traffico aggiuntivo, causato da una più frequente emissione dei messaggi di controllo (le quattro componenti del traffico sono state spiegate nella presentazione dell'altro scenario base). In realtà, il traffico generato da AHBPFlooder è comunque inferiore rispetto a quello rilevato nello scenario base con nodi fissi, ma a ciò corrisponde anche una copertura della rete visibilmente minore rispetto a tutti gli altri metodi. Per quanto riguarda

il costo del broadcast, OppFlooder risulta essere ancora una volta il metodo più efficiente, seguito da JDCTFlooder e DistFlooder, mentre AHBPFlooder e SBAFlooder rimangono più distanziati rispetto alla situazione con nodi fissi. In definitiva si può dire che la mobilità impatta negativamente sul funzionamento dei metodi, e ciò è dimostrato dal maggior numero di ritrasmissioni inutili, ma lo scenario che viene a crearsi, caratterizzato da una maggiore densità effettiva dovuta al raggruppamento dei nodi, favorisce una maggiore copertura della rete e un minor numero di trasmissioni.

L'introduzione della mobilità nello scenario base offre i primi indizi sul fatto che i metodi che devono mantenere una topologia potrebbero non riuscire a competere con quelli che non utilizzano tale conoscenza. Inoltre, anche in questo caso, sembra che OppFlooder sia la tecnica da preferire.

Lo scenario con densità variabile

Proseguiamo la presentazione e l'analisi dei risultati relativi al contesto mobile con quelli ottenuti dalle simulazioni compiute variando la densità della rete.

Gli andamenti dei risultati acquisiti nel contesto mobile (figura 7.7) sono, in generale, piuttosto simili a quelli relativi alle simulazioni con nodi fissi. Tuttavia è possibile fare alcune interessanti considerazioni. In primo luogo l'effetto introdotto dal modello random waypoint sulla copertura della rete è ancora più evidente alle basse densità; a quelle alte invece le differenze sono minime e solo AHBPFlooder mostra un leggero degrado dovuto alla mobilità dei nodi. Per quanto riguarda il traffico e il costo del broadcast è rilevante notare come i valori calano per OppFlooder, DistFlooder e JDCTFlooder, mentre aumentano per SBAFlooder e AHBPFlooder. Ciò è una chiara indicazione che questi ultimi sono meno efficienti in presenza di mobilità. La percentuale totale di trasmissioni è generalmente più bassa, mentre il numero di ritrasmissioni inutili cresce leggermente, come avviene nello scenario base. Nei due grafici in basso è ancora più interessante notare quanto sia maggiore la differenza di comportamento tra SBAFlooder e AHBPFlooder rispetto all'assenza di mobilità: il primo ritrasmette di più, anche inutilmente, ma riesce a fornire una copertura della rete più elevata, mentre il secondo richiede meno trasmissioni, non lontano dalle percentuali di OppFlooder, ma

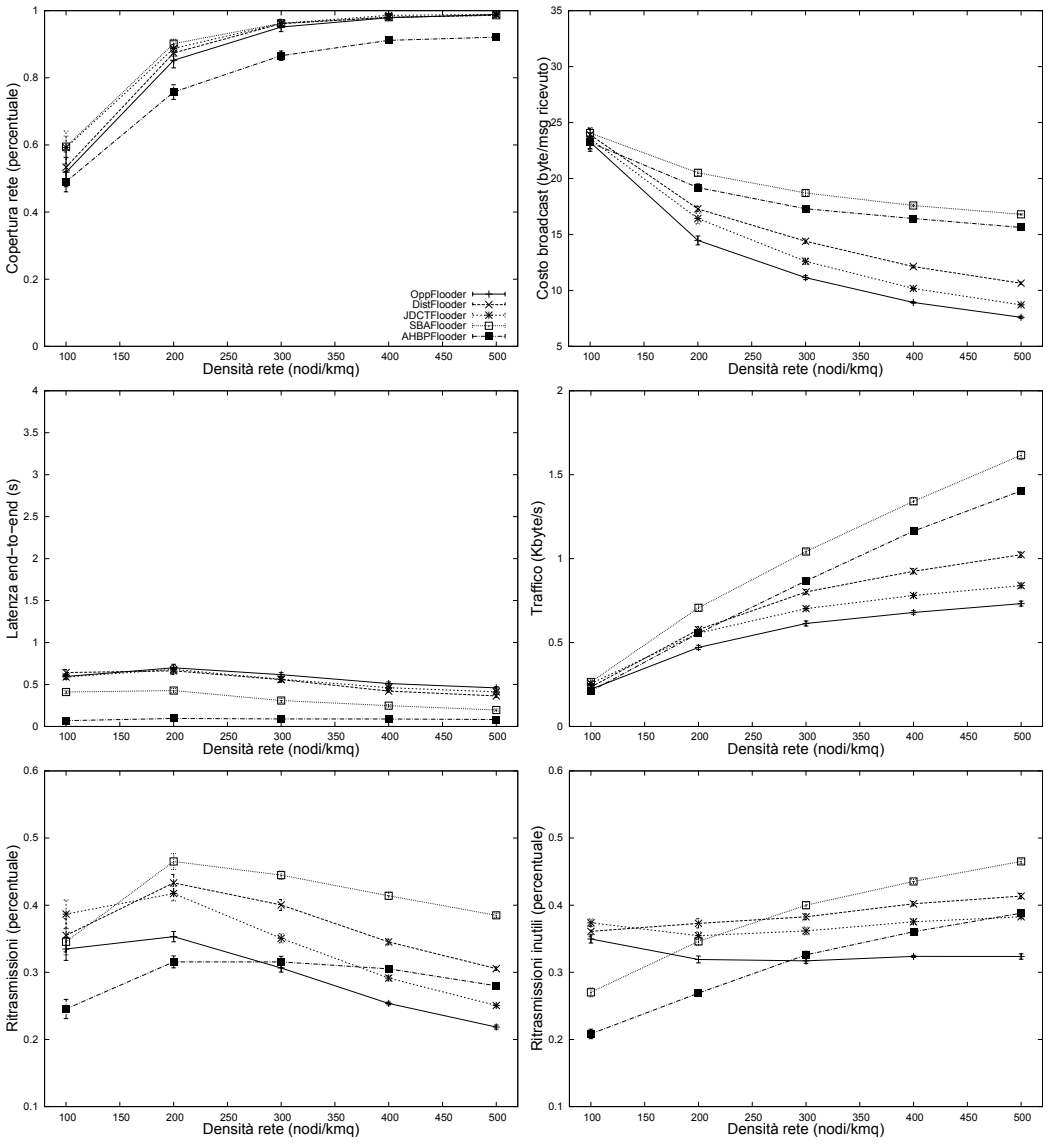

Figura 7.7: Risultati simulazioni - densità variabile (nodi mobili)

ciò avviene a discapito della copertura. La latenza, pur essendo più bassa, anche in questo caso è solo marginalmente influenzata dalla variazione della densità.

Il metodo più efficiente rimane sempre OppFlooder, ma il numero dei "diretti avversari" si restringe, a causa delle difficoltà incontrate da SBAFlooder e AHBPFlooder.

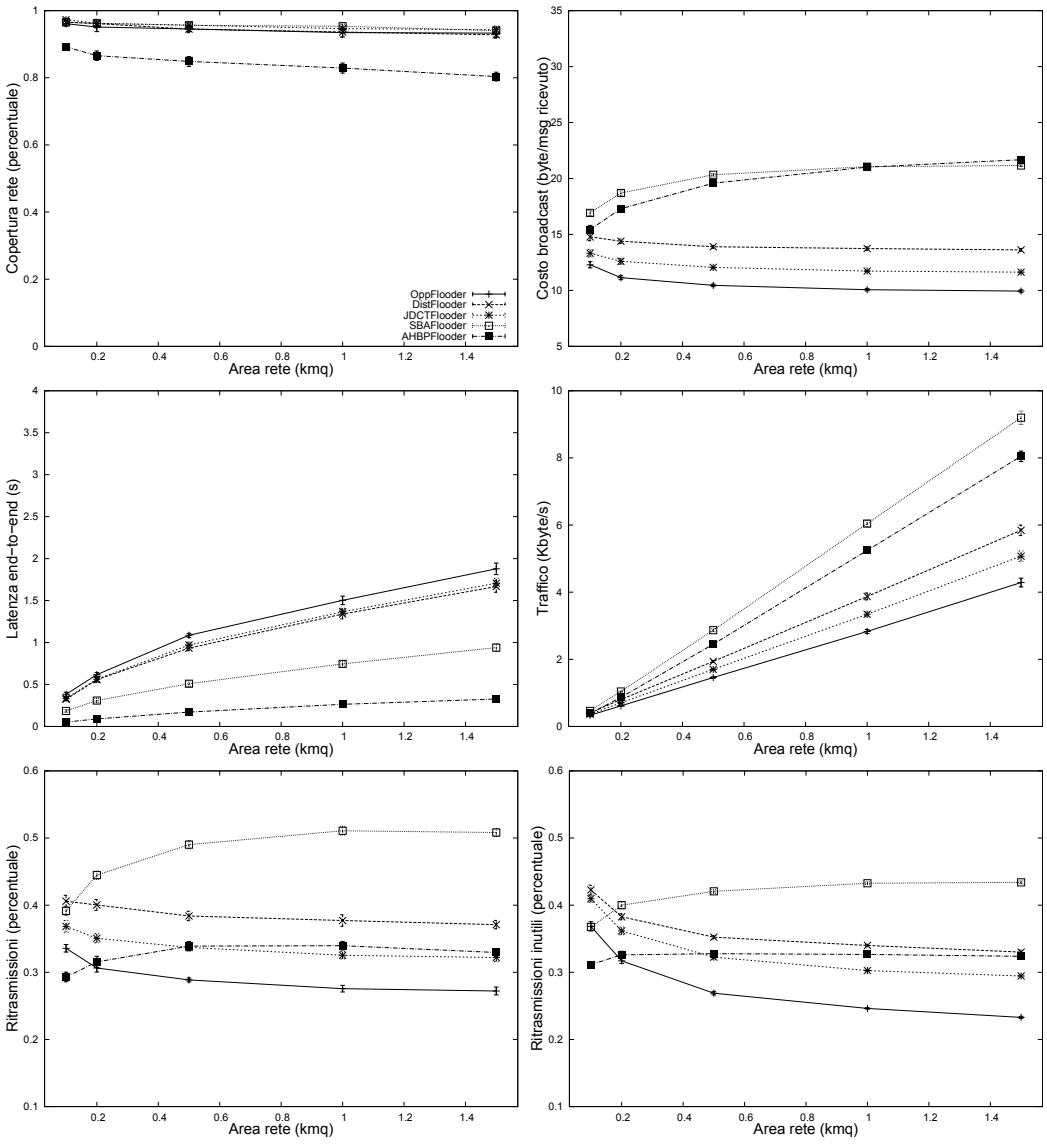

Figura 7.8: Risultati simulazioni - area variabile (nodi mobili)

Lo scenario con area variabile

In base ai risultati appena presentati per lo scenario con densità variabile, sembra sensato aspettarsi, anche in questo caso, degli andamenti che differiscono più o meno allo stesso modo in confronto agli esiti ottenuti in assenza di mobilità.

I risultati ottenuti (figura 7.8) confermano questa aspettativa. La copertura della rete appare sempre scarsamente influenzata dal variare delle

dimensioni della rete, ma AHBPFlooder risulta maggiormente staccato dagli altri a causa degli effetti della mobilità. Il tempo medio necessario a consegnare i messaggi cresce sempre con l'aumentare dell'area, anche se risulta decisamente inferiore, per il motivo già spiegato nello scenario base, rispetto al caso con nodi fissi. Gli effetti delle variazioni dell'area sono evidenti anche sul traffico generato, ma con i nodi mobili tale indice risulta essere leggermente inferiore per OppFlooder, DistFlooder e JDCTFlooder, mentre cresce notevolmente per SBAFlooder; gli effetti sul traffico si riflettono ovviamente sul costo del broadcast, che diminuisce per i primi metodi, mentre aumenta per SBAFlooder. Il costo di broadcast continua a essere influenzato dalle variazioni di area, soprattutto tra i valori più piccoli, anche se in modo piuttosto debole. Le stesse identiche considerazioni possono essere fatte per la percentuale di nodi che trasmettono, mentre le ritrasmissioni inutili sono più elevate rispetto allo scenario con nodi fissi, soprattutto per quanto riguarda SBAFlooder.

I risultati ottenuti dalle simulazioni con area variabile offrono le stesse evidenze rilevate nello studio dello scenario base e da quello del caso con densità variabile: OppFlooder, pur non essendo il migliore per copertura della rete, è indubbiamente il metodo più efficiente.

Lo scenario con frequenza d'invio dei messaggi variabile

Analizziamo infine i risultati ottenuti nelle simulazioni riguardanti la congestione della rete, per capire quali effetti essa abbia sul funzionamento dei cinque metodi di broadcast quando è presente pure la mobilità dei nodi.

Anche gli effetti della congestione della rete in mobilità si rivelano essere piuttosto simili a quelli riscontrati in condizione di rete fissa. Figura 7.9 però mostra alcune particolarità degne di nota, che andiamo subito ad analizzare. Per prima cosa risultano immediatamente evidenti le difficoltà incontrate da SBAFlooder nell'operare in un contesto mobile e congestionato: rispetto al caso con nodi fissi, la copertura della rete non è più ai livelli di OppFlooder e JDCTFlooder, la percentuale di trasmissioni totali e inutili cresce notevolmente, così come anche il traffico e il costo di broadcast; infine anche la latenza peggiora, avvicinandosi a quella dei tre metodi che non sfruttano la conoscenza dei vicini. La seconda cosa interessante è chiaramente visibile nel

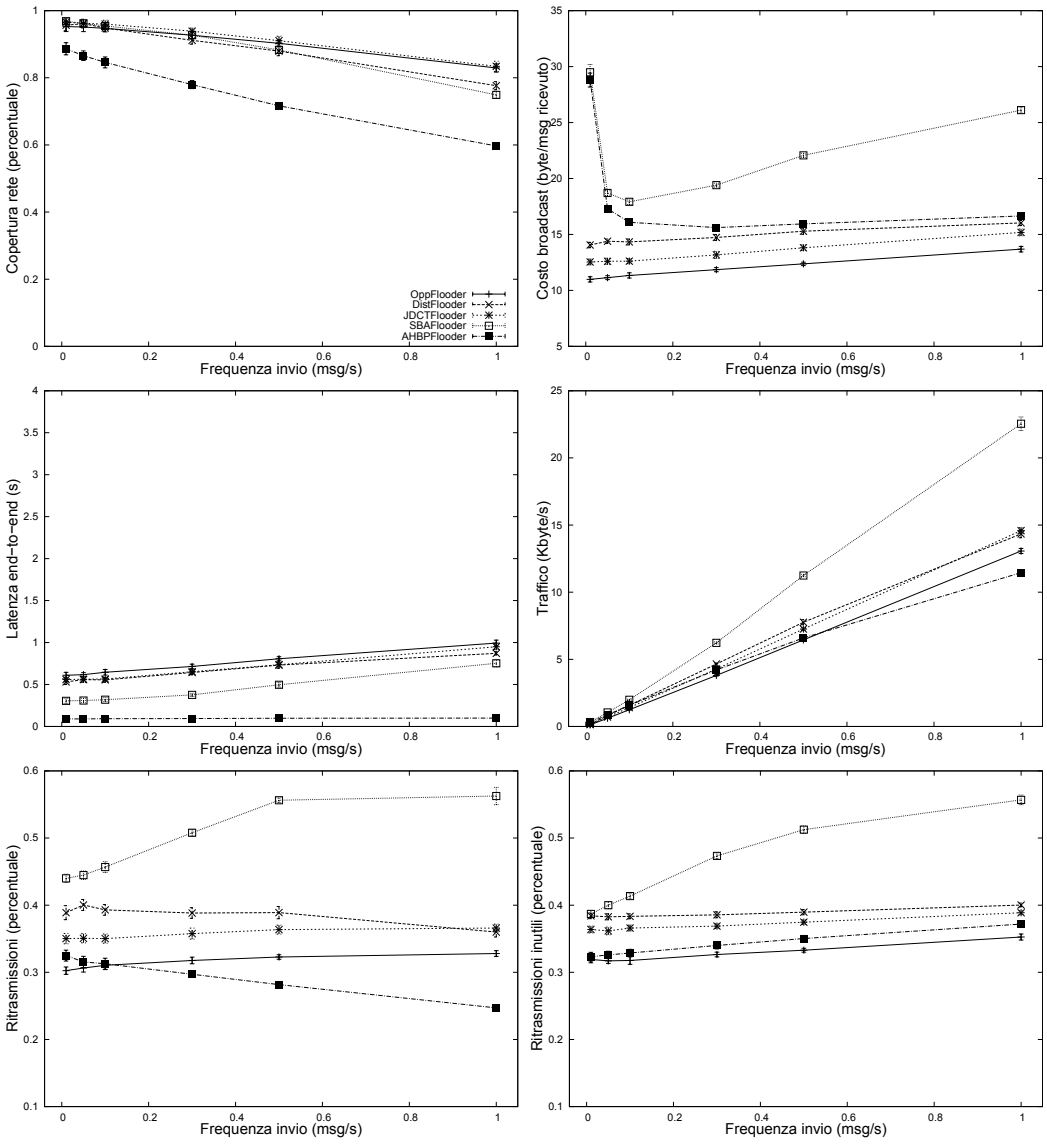

Figura 7.9: Risultati simulazioni - frequenza invio variabile (nodi mobili)

grafico del costo del broadcast: in presenza di mobilità e quando i messaggi di broadcast vengono inviati molto sporadicamente, l'efficienza di SBAFlooder e di AHBPFlooder è seriamente compromessa dai messaggi di controllo, che in questo caso sono emessi più frequentemente. Per il resto rimangono valide le considerazioni relative agli effetti della congestione della rete fatte nello stesso scenario, ma con i nodi fissi, mentre il modello usato per la mobilità influenza sempre positivamente, in generale, la copertura della rete, e per

OppFlooder, DistFlooder e JDCTFlooder anche l'efficienza.

OppFlooder, anche in quest'ultima situazione, conferma di essere il metodo migliore per il broadcasting, in quanto esibisce le migliori prestazioni sia nella copertura della rete sia nel costo del broadcast; è vero che AHBPFlooder è più performante per quanto riguarda le percentuali di trasmissione, ma è anche vero che garantisce una copertura della rete nettamente inferiore rispetto a OppFlooder.

7.2.3 L'impatto della mobilità sui metodi scelti

Abbiamo già visto, grazie ai risultati ottenuti dalle simulazioni compiute negli scenari in cui i nodi si muovono a velocità ridotta, che i metodi che basano il loro funzionamento sulla conoscenza dei vicini non riescono a fornire prestazioni in linea con le altre tre tecniche, in presenza di mobilità. In particolare, nel caso di SBAFlooder il problema è la scarsa efficienza, mentre AHBPFlooder garantisce una copertura della rete marcatamente inferiore rispetto a tutti gli altri metodi. Siamo però anche interessati a verificare quale sia l'effetto della variazione della velocità di spostamento dei nodi sul funzionamento dei cinque metodi considerati.

I grafici di figura 7.10 dimostrano, in maniera piuttosto evidente, quali sono i metodi adatti alla mobilità per loro natura e quali invece non lo sono. Escludendo infatti l'effetto dovuto al passaggio da un contesto fisso a uno mobile, OppFlooder, DistFlooder e JDCTFlooder risultano essere totalmente indifferenti all'aumento della velocità, e ciò accade per tutti gli indici di prestazione considerati. Sia SBAFlooder sia AHBPFlooder, pur essendo presentati come metodi capaci di operare in un contesto mobile, devono invece affrontare il problema dell'aggiornamento delle informazioni topologiche e ciò li rende molto meno adatti a situazioni in cui la mobilità assume un ruolo rilevante. Per evitare comportamenti quali quelli presentati in figura, in cui SBAFlooder arriva quasi a far trasmettere il 90% dei nodi e AHBPFlooder fornisce una copertura della rete inferiore agli altri con un costo di broadcast insoddisfacente, è necessario aumentare la frequenza di emissione dei mes-

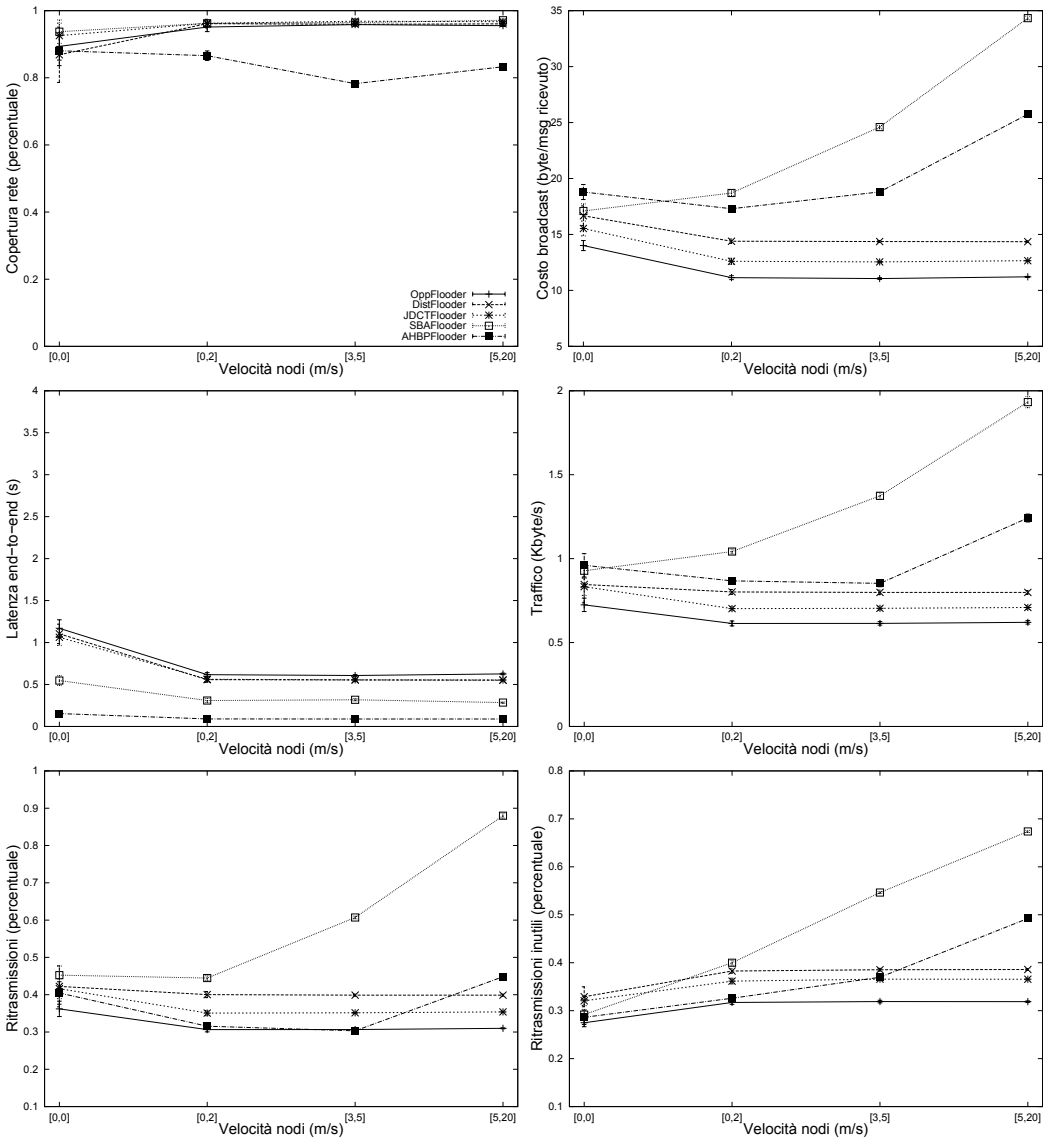

Figura 7.10: Risultati simulazioni - velocità nodi variabile

saggi di broadcast, ma ciò comporta automaticamente un overhead superiore e quindi un'efficienza inferiore.

Per concludere, si riconosce che le condizioni di mobilità media e soprattutto elevata sono meno probabili da trovare in una WSN rispetto ai casi con i nodi fissi o in movimento lento, ma questi risultati sono proprio serviti a portare chiaramente alla luce come i cinque metodi scelti si confrontano con la mobilità.

7.2.4 Considerazioni sulla comparazione dei metodi scelti

In questo sottoparagrafo conclusivo sono raccolti i risultati più importanti e le considerazioni di maggior interesse che nascono dall'analisi dei risultati dell'estesa campagna simulativa. Per prima cosa elenchiamo i due principali risultati:

- OppFlooder è sicuramente il metodo che esce vincitore da questa comparazione, in quanto riesce, praticamente in tutte le occasioni, a garantire l'efficienza più elevata e a raggiungere una copertura della rete generalmente in linea con i metodi che eccellono in questo campo, o di poco inferiore; a ciò si aggiunge il fatto che OppFlooder è una tecnica caratterizzata da un funzionamento molto semplice e che non richiede complicate scelte di parametri;

- i metodi che funzionano in base alla conoscenza dei vicini non sono particolarmente adatti ai contesti mobili, mentre si difendono meglio in reti fisse; considerando però la loro maggiore complessità, che ne rende meno agevole l'implementazione, risulta abbastanza chiaro che, in generale, questi metodi non sono i più convenienti da utilizzare.

Quindi evidenziamo brevemente i punti di forza e di debolezza rilevati per i cinque metodi confrontati e aggiungiamo qualche breve considerazione.

OppFlooder, come già anticipato nei risultati, è il metodo più semplice ed efficiente e si comporta in maniera molto buona in qualsiasi condizione affrontata nelle simulazioni. Un suo punto debole è la latenza, la più alta rilevata, ma ciò può essere aggirato diminuendo in maniera opportuna il ritardo di ritrasmissione. Si nota inoltre che, in generale, non è il miglior metodo per quanto riguarda la copertura della rete e non esiste un parametro attraverso cui sia possibile guadagnare un po' di copertura a discapito di un po' di efficienza, qualora sia desiderato.

DistFlooder non offre risultati buoni quanto OppFlooder, soprattutto per quanto riguarda l'efficienza, ma si è rivelato essere un valido metodo per il broadcast, nonostante la sua semplicità e il suo essere una tra le prime

tecniche presentate per limitare le trasmissioni ridondanti. In DistFlooder, oltre a contenere la latenza diminuendo il massimo tempo di attesa prima di una ritrasmissione, è anche possibile scegliere il compromesso desiderato tra efficacia ed efficienza, variando la soglia sulla distanza.

JDCTFlooder può essere considerato la seconda scelta di questa comparazione, in quanto i suoi risultati sono piuttosto simili a quelli di OppFlooder. JDCTFlooder risulta essere un po' più complicato di OppFlooder, ma ciò permette anche di scegliere accuratamente il livello di copertura desiderato, grazie alla possibilità di agire su ben due parametri. Anche in questo caso la latenza di rete può essere opportunamente diminuita.

SBAFlooder, dei due metodi che sfruttano la conoscenza dei vicini, è quello che garantisce la migliore copertura della rete, a discapito però dell'efficienza. In condizioni di rete fissa offre buoni risultati, anche se non all'altezza delle aspettative, e risulta preferibile ad AHBPFlooder, mentre è sconsigliato il suo utilizzo in presenza di mobilità.

AHBPFlooder, per concludere, è il metodo più controverso tra quelli selezionati per la comparazione: sulla carta dovrebbe essere il miglior metodo in assenza di mobilità, ma in simulazione ha mostrato le sue debolezze fino dai primi test. Nella campagna simulativa non è riuscito a reggere il confronto con i ben più semplici OppFlooder, DistFlooder e JDCTFlooder ed è riuscito a competere con SBAFlooder solo in presenza di mobilità.

Infine si ritiene decisamente rilevante far notare che i risultati misurati tramite la campagna di simulazioni si differenziano, a volte in maniera piuttosto evidente, da quelli presentati in letteratura in studi simili e ciò, molto probabilmente, è dovuto all'ambiente di simulazione utilizzato. Questo discorso viene analizzato nella prossima sezione.

7.3 I risultati degli esperimenti reali: comparazione con le simulazioni

In quest'ultima sezione del capitolo sono raccolti i risultati degli esperimenti sulle due WSN reali e delle simulazioni eseguite in condizioni di rete più simili

possibile a quelle presenti in Motelab e Indriya, come specificato nel paragrafo 6.4. Dato che l'obiettivo è validare gli esiti della campagna simulativa, si è ritenuto utile presentare i risultati in due forme distinte:

- metodo per metodo, inserendo nei grafici i risultati ottenuti sia con il simulatore (su rete quadrata e rettangolare) sia eseguendo gli esperimenti reali; in questo modo è possibile mostrare le differenze tra simulatore e reti sperimentali per ognuno dei metodi considerati;
- tutti i metodi, in grafici separati per le simulazioni e gli esperimenti reali e con valori relativi rispetto ai risultati di OppFlooder, così da poter facilmente valutare quanto l'ambiente di simulazione usato sia affidabile per eseguire delle comparazioni.

In tutti i casi, sì è deciso di presentare solamente i grafici relativi alla copertura della rete e al costo del broadcast. La motivazione principale è quella di focalizzare l'attenzione due indicatori ritenuti importanti; inoltre, come già visto nel capitolo 7.1, la percentuale di ritrasmissioni può essere fortemente influenzata dalla topologia della rete e quindi non è particolarmente sensato utilizzarla per i nostri scopi, visto che simulazioni e reti reali hanno topologie potenzialmente molto differenti. Per quest'ultimo motivo anche i grafici del traffico risultano essere di dubbia utilità, in quando quest'ultimo dipende in maniera evidente dal numero di ritrasmissioni. Si noti inoltre, che nei grafici a densità variabile, si è optato per far apparire la densità sulle ascisse, invece che i diversi valori di potenza radio.

I grafici di figura 7.11 rappresentano il confronto, con densità variabile, tra le simulazioni e gli esperimenti reali per ogni singolo metodo: in alto si trova OppFlooder, al centro JDCTFlooder e in basso SBAFlooder.

Per quanto riguarda OppFlooder e JDCTFlooder è facilmente rilevabile una notevole somiglianza nell'andamento dei risultati dei test effettuati sulle due WSN rispetto a quelli ottenuti tramite le simulazioni, con rete sia quadrata sia rettangolare. I valori ottenuti invece non sono sempre molto simili, ma ciò può essere imputato, almeno in parte, proprio alle differenze topologiche: infatti i risultati che differiscono maggiormente sono proprio quelli relativi a Motelab e Indriya, cioè ai due contesti reali, soprattutto alla

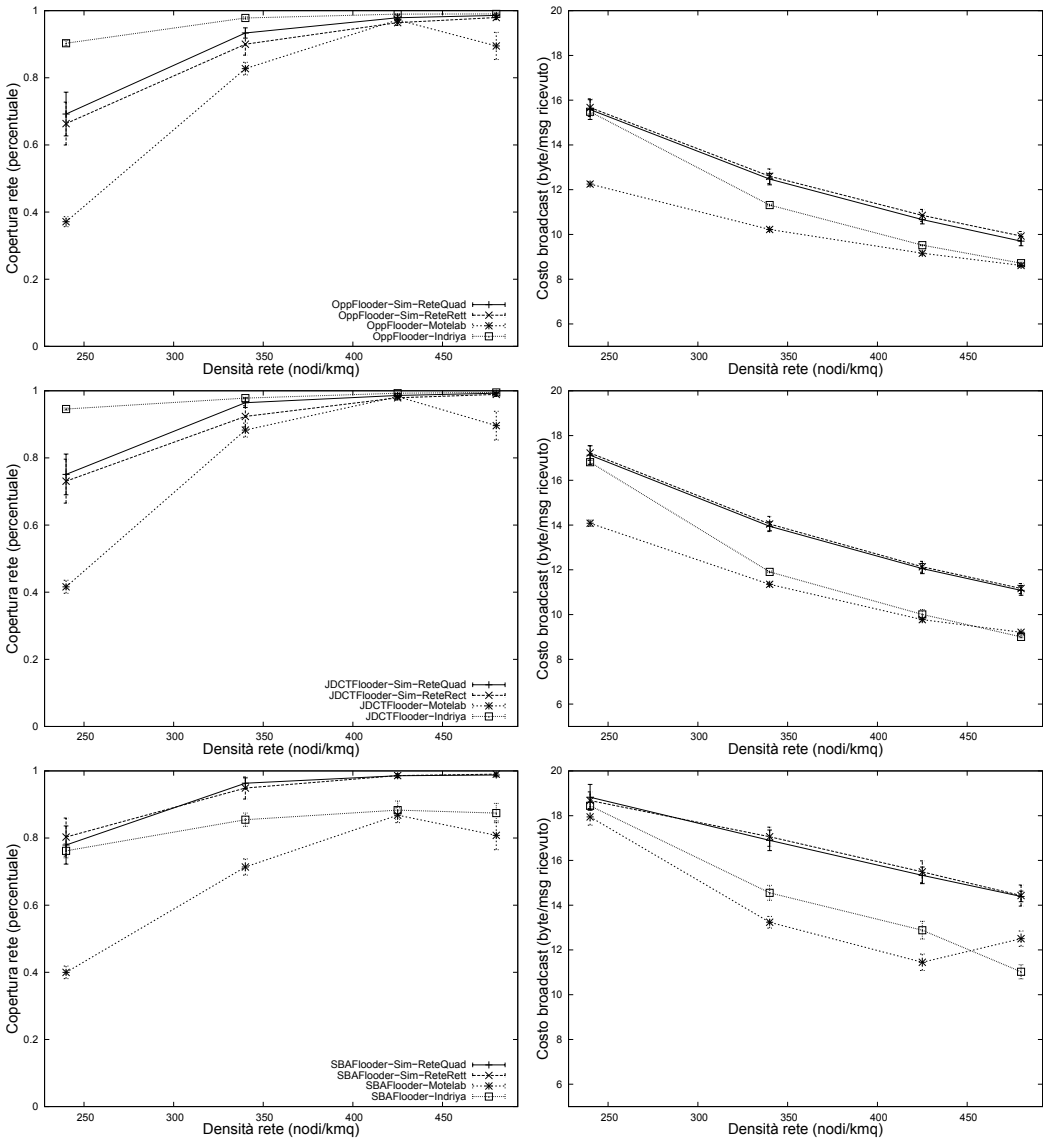

Figura 7.11: Comparazione simulatore/esperimenti reali - densità variabile (OppFlooder, JDCTFlooder e SBAFlooder)

densità minore e questo deriva proprio dalle differenze topologiche delle due reti (cfr. paragrafo 6.1). Dai grafici di OppFlooder e JDCTFlooder, è anche evidente il fatto che le simulazioni, per come sono state eseguite, restituiscono un valore medio degli indici considerati: infatti i loro risultati si trovano circa a metà strada rispetto a quelli di Motelab e Indriya.

Per SBAFlooder il discorso è un po' diverso, siccome la copertura della rete ottenuta su Motelab e Indriya è vistosamente inferiore in confronto ai casi simulati: l'andamento è ancora simile, ma è come se la copertura della rete massima fosse un valore inferiore al 100%. Sono quindi stati eseguiti alcuni esperimenti aggiuntivi, per identificare l'origine del problema. Si è scoperto che è il traffico generato a influenzare negativamente l'efficacia di SBAFlooder nei casi reali: sia diminuendo la frequenza di invio dei messaggi da 0.05 a 0.01 sia riducendo il numero dei nodi sorgente da 20 a 5 sono stati ottenuti risultati in linea con i grafici soprastanti e con i risultati di SBAFlooder in simulazione. Dato che il calo di prestazioni rilevato è direttamente collegato al traffico, è sensato ritenere che SBAFlooder sia limitato dalle prestazioni dei nodi TelosB, che probabilmente non riescono sempre a gestire tutte le operazioni necessarie per il funzionamento del metodo in un tempo sufficientemente breve e così può accadere che alcuni messaggi non vengano ritrasmessi, semplicemente perché la coda di ritrasmissione risulta piena. Ovviamente tale coda ha la stessa dimensione nei tre metodi considerati, cioè può contenere fino a 8 messaggi, e questo valore è stato fissato in modo da non superare la capacità massima della RAM dei nodi TelosB (si noti che SBAFlooder ne utilizza quanto più possibile).

In figura 7.12 sono riportati i risultati ottenuti aumentando la frequenza di invio dei messaggi, sempre con OppFlooder in alto, JDCTFlooder in mezzo e SBAFlooder in basso. Esaminando i grafici relativi ai primi due metodi è evidente quanto il simulatore restituisca valori simili a quelli delle reti reali, soprattutto per quanto riguarda la copertura della rete. In questi grafici si nota addirittura che sia Motelab sia Indriya sono meno influenzate dalla congestione rispetto alle simulazioni e questo potrebbe far pensare che in simulazione siano sovrastimate, seppur limitatamente, le collisioni; ciò sembra anche essere confermato da un maggior costo del broadcast rilevato in simulazione.

I risultati ottenuti dagli esperimenti reali con SBAFlooder confermano quanto detto nel caso a densità variabile: risulta infatti palese che l'efficacia di SBAFlooder, nelle due WSN sperimentali, sia fortemente limitata dalla congestione della rete.

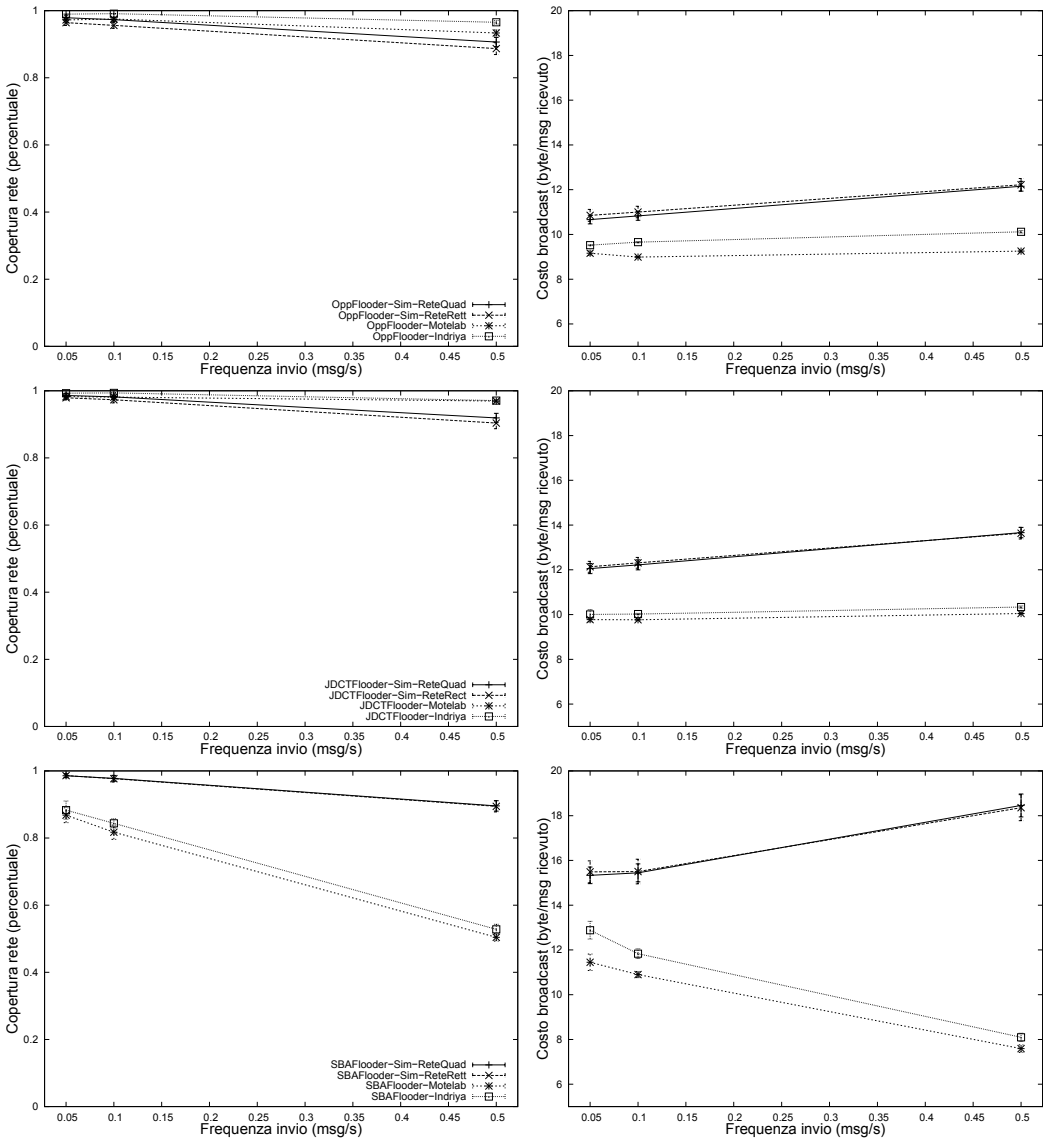

Figura 7.12: Comparazione simulatore/esperimenti reali - frequenza invio variabile (OppFlooder, JDCTFlooder e SBAFlooder)

I grafici appena presentati dimostrano che i risultati ottenuti con il simulatore sono ragionevolmente validi, anche tenuto conto delle condizioni non esattamente equivalenti in cui simulazioni e test sperimentali sono stati eseguiti. Per supportare questa affermazione, si ritiene utile anche mostrare il confronto tra i risultati in simulazione e negli esperimenti reali per quanto riguarda JDCTFlooder, questa volta non riportando i valori assoluti ma per-

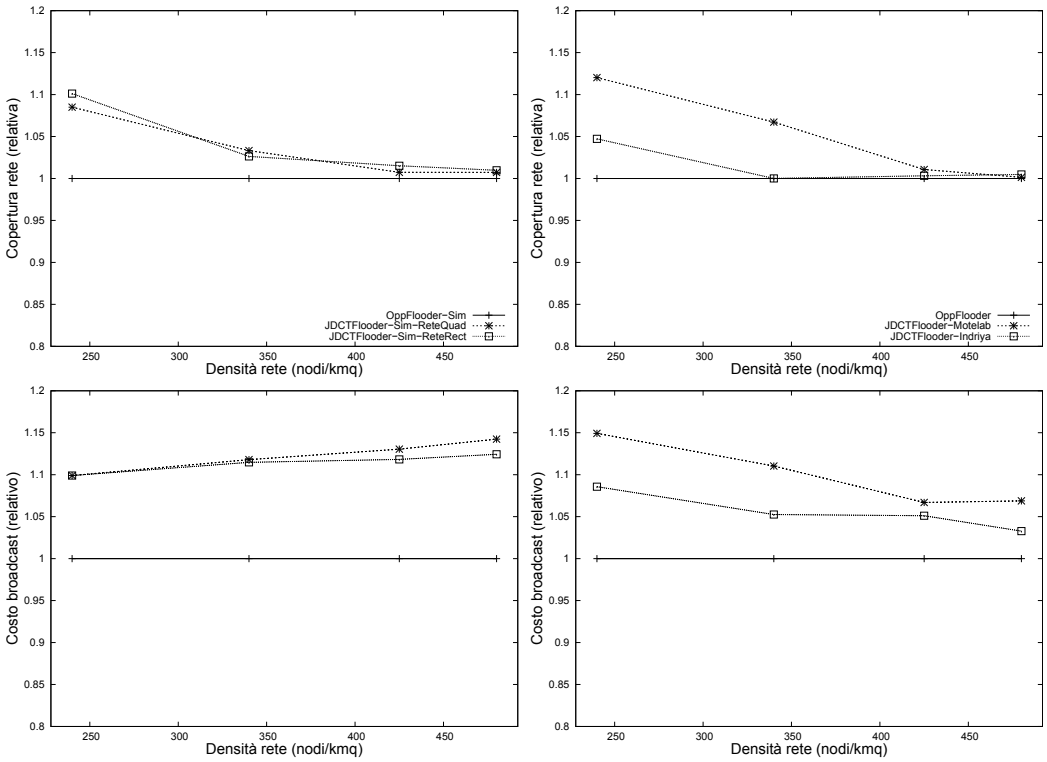

Figura 7.13: Comparazione simulatore/esperimenti reali - densità variabile (risultati percentuali rispetto a OppFlooder)

centuali rispetto al valore di OppFlooder, che viene posto pari a 1. In base ai problemi riscontrati con SBAFlooder, si è deciso di escludere i risultati ottenuti con questo metodo. In questo caso, per facilitare il confronto, i due grafici di sinistra sono relativi ai risultati delle simulazioni, mentre quelli di destra sono stati ottenuti con gli esperimenti su Motelab e Indriya. Si noti che, in ognuno dei seguenti grafici, OppFlooder è riportato una sola volta, ma la prestazione relativa di JDCTFlooder è stata calcolata, ovviamente, rispetto ai risultati di OppFlooder ottenuti nelle stesse identiche condizioni oppure sulla stessa rete sperimentale.

Figura 7.13 mostra questi ultimi risultati, quando la densità è variabile. In questa occasione, risulta abbastanza evidente che le prestazioni in simulazione di JDCTFlooder, in rapporto a quelle di OppFlooder, non esibiscono differenze particolarmente rilevanti rispetto a quelle reali, né per quanto riguarda la copertura della rete né nel caso del costo del broadcast. Inoltre,

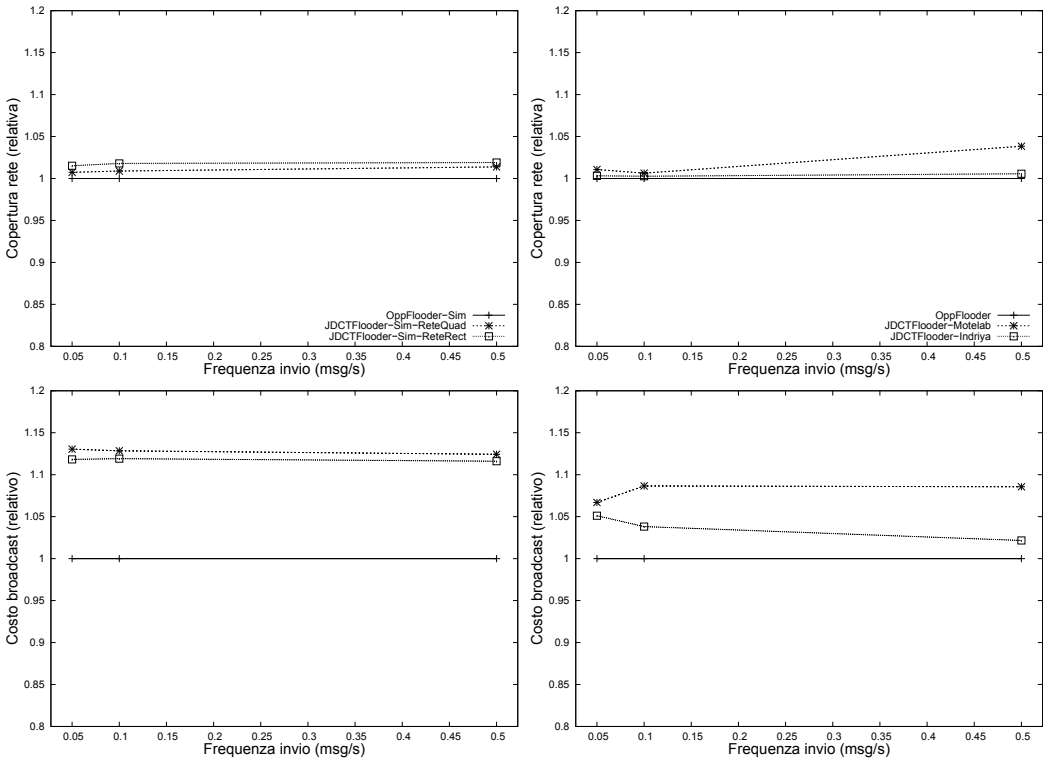

Figura 7.14: Comparazione simulatore/esperimenti reali - frequenza invio variabile (risultati percentuali rispetto a OppFlooder)

anche in questo caso, la topologia della rete può avere una certa influenza, e ciò è supportato dalle differenze, seppur modeste, che distinguono gli andamenti ottenuti su Motelab e Indriya.

In figura 7.14 è infine riportato il confronto, sempre per copertura di rete e costo del broadcast, nello scenario in cui varia la frequenza di invio dei messaggi. La comparazione tra OppFlooder e JDCTFlooder per quanto riguarda la copertura della rete è veramente molto simile in simulazione e negli esperimenti reali, mentre ciò non è così evidente osservando i due grafici del costo del broadcast; tuttavia il divario rimane piuttosto limitato.

Il fatto che, in tutti i risultati presentati, sia stato rilevato un traffico (qui non mostrato) inferiore negli esperimenti eseguiti su entrambe le WSN sperimentali rispetto alle simulazioni, può rendere particolarmente interessante eseguire un confronto tra i risultati ottenuti nei test reali e quelli di minimo teorico, per verificare se in scenari reali i metodi considerati si avvicinano

maggiormente alla massima efficienza. Per fare ciò è però necessario conoscere esattamente la topologia della rete reale, informazione non disponibili per Motelab e Indriya, e sarebbe preferibile eseguire gli esperimenti in campo aperto, in modo da evitare l'interferenza di ostacoli come i muri.

Per concludere, gli studi effettuati su reti sperimentali e opportunamente ripetuti al simulatore dimostrano che, almeno nei casi studiati, il modello di scheda fisica e il livello MAC della versione del Mobility Framework usata in OMNeT++ permettono di ottenere un ambiente di simulazione in grado di restituire risultati ragionevolmente aderenti al caso reale. Tuttavia, prendendo come esempio il caso di SBAFlooder sulle WSN sperimentali, è sempre necessario tenere in considerazione le numerose e varie limitazioni che possono essere presenti nelle reti reali ed eventualmente sfruttare le reti sperimentali liberamente utilizzabili, come Motelab e Indriya, per ottenere una conferma.

Capitolo 8

Conclusioni

Nella presente tesi di laurea è stato affrontato il problema del broadcast all'interno delle reti wireless, un'operazione particolarmente importante poiché spesso usata come componente di protocolli più complessi e che quindi deve essere effettuata con la massima efficienza. Particolare attenzione è stata rivolta nei confronti delle WSN. Tali reti sono generalmente costituite da nodi caratterizzati da una modesta potenza computazionale, da una velocità di trasmissione limitata e da scarse disponibilità energetiche; inoltre possono essere utilizzate anche in contesti mobili.

Sono stati considerati e quindi presentati, in una consistente trattazione dello stato dell'arte, numerosi metodi per il broadcast nelle reti wireless; l'analisi della letteratura scientifica ha messo in luce quanti sforzi si stiano facendo e quante tecniche siano state ideate per ottenere metodi di broadcast efficienti e possibilmente in grado di operare anche in contesti in cui è presente la mobilità dei nodi. Quattro di questi metodi sono stati scelti per essere comparati con OppFlooder, una semplice ed efficiente tecnica di broadcast, da noi escogitata; i cinque metodi da confrontare sono quindi stati presentati e analizzati nel dettaglio, in modo da evidenziare le caratteristiche principali e le possibili debolezze di ognuno.

Nell'ottica di valutare l'efficienza dei metodi scelti rispetto al caso migliore possibile, è stato realizzato un algoritmo centralizzato in grado di ricavare il minimo numero di trasmissioni necessarie per ottenere la massima copertura

su un modello, opportunamente semplificato, di una rete wireless. In tale algoritmo assume primaria importanza un'esecuzione quanto più possibile efficiente e ottimizzata, poiché è stato necessario abbattere più possibile il tempo, altrimenti decisamente considerevole, richiesto per eseguire tutte le ricerche di minimo teorico desiderate.

Un'estesa campagna di simulazioni, scelta in modo da rappresentare un ampio spettro di condizioni di rete, è stata quindi eseguita per comparare OppFlooder con i quattro metodi selezionati. Per tali simulazioni è stato usato OMNeT++, un simulatore a eventi discreti, e una sua estensione, chiamata Mobility Framework. È anche stata effettuata una serie di esperimenti su due WSN reali, in modo da poter verificare l'affidabilità dei risultati forniti dal simulatore.

L'esito principale della campagna di simulazioni è che OppFlooder è un'ottima tecnica per il broadcast in reti come le WSN: non solo fornisce prestazioni migliori rispetto agli altri quattro metodi in numerose situazioni, soprattutto per quanto riguarda l'efficienza, ma è anche in grado di adattarsi particolarmente bene a tutte le condizioni di rete in cui è stato testato. Grazie ai test sulle reti sperimentali, è stato possibile accertare che l'ambiente di simulazione utilizzato, con particolare riferimento al modello della radio, del canale e al livello MAC adottati, fornisce risultati ampiamente compatibili con i casi reali, a differenza invece di molti risultati, riportati nella letteratura scientifica, che evidentemente non sono stati ottenuti ponendo la necessaria attenzione sulle caratteristiche del simulatore impiegato.

In conclusione, alla luce dei risultati ottenuti, si ritiene possa essere particolarmente interessante estendere questo lavoro in due differenti ambiti. In primo luogo è auspicabile un possibile miglioramento di OppFlooder, che cerchi di limitare la percentuale di ritrasmissioni inutili senza però introdurre meccanismi che riducano eccessivamente la semplicità e l'adattabilità, caratteristiche fondamentali di questa tecnica. Inoltre si giudica importante approfondire il discorso relativo al confronto tra le simulazioni e le reti reali, per esempio eseguendo esperimenti su topologie per quanto possibile uguali sul simulatore e nella realtà, confrontando i risultati ottenuti anche rispetto al minimo teorico e includendo in questi test anche altri simulatori di rete.

Bibliografia

[1] J. Blum, M. Ding, A. Thaeler, and X. Cheng. Connected dominating set in sensor networks and MANETs. In *Handbook of Combinatorial Optimization*, volume 5, pages 329–369. Kluwer Academic Publishers, 2004.

[2] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[3] B.N. Clark, C.J. Colbourn, and D.S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990.

[4] M.D. Colagrosso. Intelligent broadcasting in mobile ad hoc networks: Three classes of adaptive protocols. *EURASIP Journal on Wireless Communicaions and Networking*, 2007(1):25–25, 2007.

[5] G. Cugola and M. Migliavacca. A context and content-based routing protocol for mobile sensor networks. In *Proceedings of the 6th European Conference on Wireless Sensor Networks*, pages 69–85, Cork, Ireland, 2009.

[6] R. Gandhi, S. Parthasarathy, and A. Mishra. Minimizing broadcast latency and redundancy in ad hoc networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 222–232, Annapolis, Maryland, USA, 2003.

[7] D. Ganesan, D. Estrin, B. Krishnamachari, S. Wicker A. Woo, and D. Culler. Complex behavior at scale: An experimental study of low-power wireless sensor networks, 2002.

[8] M.R. Garey and D.S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1990.

[9] Z. Haas. A new routing protocol for the reconfigurable wireless networks. In *Proceedings of the IEEE 6th International Conference on Universal Personal Communications*, pages 562–566, San Diego, California, USA, 1997.

[10] Z. Haas, J.Y. Halpern, and L. Li. Gossip-based ad hoc routing. Technical report, Cornell University, Ithaca, New York, USA, 2001.

[11] L. Hogie, P. Bouvry, M. Seredynski, and F. Guinand. A bandwidth-efficient broadcasting protocol for mobile multi-hop ad hoc networks. In *Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, page 71, Morne, Mauritius, 2006.

[12] D.B. Johnsort. Routing in ad hoc networks of mobile hosts. In *Proceedings of the 1st Workshop on Mobile Computing Systems and Applications*, pages 158–163, Santa Cruz, California, USA, 1994.

[13] A. Jüttner and A. Magi. Tree based broadcast in ad hoc networks. *Mobile Networks and Applications*, 10(5):753–762, 2005.

[14] K-W. Kim, K-K. Kim, C-M. Han, M.M-O. Lee, and Y-K. Kim. An enhanced broadcasting algorithm in wireless ad-hoc networks. In *Proceedings of the International Conference on Information Science and Security*, pages 159–163, Seoul, South Corea, 2008.

[15] D. Kouvatsos and I-H. Mkwawa. Broadcasting methods in mobile ad hoc networks: An overview. In *Proceedings of the 3rd International Working*

Conference on Performance Modellingand Evaluation of Heterogeneous Networks, pages T09/1–T09/14, Ilkley, UK, 2005.

[16] P. Kyasanur, R.R. Choudhury, and I. Gupta. Smart gossip: An adaptive gossip-based broadcasting service for sensor networks. In *Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor System*, pages 91–100, Vancouver, Canada, 2006.

[17] L. Layuan, Z. Feng, L. Chunlin, and S. Qiang. A distributed broadcast algorithm for wireless mobile ad hoc networks. In *Proceedings of the 13th International Multimedia Modeling Conference, part II*, pages 494–501, Singapore, 2007.

[18] S-H. Lee and Y-B. Ko. An efficient neighbor knowledge based broadcasting for mobile ad hoc networks. In *Proceedings of the 6th International Conference on Computational Science, part II, LNCS 3992*, pages 1097–1100, Reading, UK, 2006.

[19] Y. Li, S. Peng, and W. Chu. An efficient distributed broadcasting algorithm for wireless ad hoc networks. In *Proceedings of the 6th International Conference on Parallel and Distributed Computing Applications and Technologies*, pages 75–79, Washington, District of Columbia, USA, 2005.

[20] H. Lim and C. Kim. Multicast tree construction and flooding in wireless ad hoc networks. In *Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 61–68, Boston, Massachusetts, United States, 2000.

[21] H. Liu, X. Jia, P-J. Wan, X. Liu, and F. Yao. Efficient flooding scheme based on 1-hop information in mobile ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 18(5):658–671, 2007.

[22] W. Lou and J. Wu. A cluster-based backbone infrastructure for broadcasting in MANETs. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 221.1, Nice, France, 2003.

[23] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, Atlanta, Georgia, USA, 2002.

[24] D. Manjunath, M.C. Chan, and A.L. Ananda. Indriya: A low cost, 3D wireless sensor network testbed. Nota: in attesa di essere pubblicato.

[25] S-Y. Ni, Y-C. Tseng, Y-S. Chen, and J-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162, Seattle, Washington, USA, 1999.

[26] K. Obraczka, K. Viswanath, and G. Tsudik. Flooding for reliable multicast in multi-hop ad hoc networks. *Wireless Networks*, 7(6):627–634, 2001.

[27] L. Orecchia, A. Panconesi, C. Petrioli, and A. Vitaletti. Localized techniques for broadcasting in wireless sensor networks. In *Proceedings of the 2004 joint workshop on Foundations of mobile computing*, pages 41–51, Philadelphia, Pennsylania, USA, 2004.

[28] O. Osechas, J. Thiele, J. Bitsch, and K. Wehrle. Ratpack: Wearable sensor networks for animal observation. In *Proceedings of the 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 538–541, Vancouver, Canada, 2008.

[29] W. Peng and X-C. Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, pages 129–130, Boston, Massachusetts, USA, 2000.

[30] W. Peng and X-C. Lu. AHBP: An efficient broadcast protocol for mobile ad hoc networks. *Journal of Computer Science and Technology*, 16(2):114–125, 2001.

[31] W. Peng and X-C. Lu. Efficient broadcast in mobile ad hoc networks using connected dominating sets. *Journal of Software*, 12(4):529–536, 2001.

[32] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd Workshop on Mobile Computing Systems and Application*, pages 90–100, New Orleans, Louisiana, USA, 1999.

[33] A. Quayyum, L. Viennot, and A. Laouiti. Multipoint relaying for flooding broadcast messages in mobilewireless networks. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, page 298, Hawaii, USA, 2002.

[34] M. Pallikonda Rajasekaran, S. Radhakrishnan, and P. Subbaraj. Elderly patient monitoring system using a wireless sensor network. *Telemedicine and e-Health*, 15(1):73–79, 2009.

[35] L. Ruan, H. Du, X. Jia, W. Wu, Y. Li, and K-I. Ko. A greedy approximation for minimum connected dominating sets. *Theoretical Computer Science*, 329(1-3):325–330, 2004.

[36] A. Schoofs, C. Daymand, R. Sugar, U. Mueller, A. Lachenmann, S.M. Kamran, A. Gefflaut, L. Thiem, and M. Schuster. Poster abstract: Ip-based testbed for herd monitoring. In *Proceedings of the 8th International Conference on Information Processing in Sensor Networks*, pages 365–366, San Francisco, California, USA, 2009.

[37] Sito web Senslab: http://www.senslab.info.

[38] F. Stann, J. Heidemann, R. Shroff, and M.Z. Murtaza. RBP: Robust broadcast propagation in wireless networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 85–98, Boulder, Colorado, USA, 2006.

[39] I. Stoianov, L. Nachman, S. Madden, and T. Tokmouline. Pipenet: A wireless sensor network for pipeline monitoring. In *Proceedings of the 6th*

international conference on Information processing in sensor networks, pages 264–273, Cambridge, Massachusetts, USA, 2007.

[40] I. Stojmenovic, M. Seddigh, and J. Zunic. Internal nodes based broadcasting in wireless networks. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 9*, page 9005, Hawaii, USA, 2001.

[41] J. Sucec and I. Marsic. An efficient distributed network-wide broadcast algorithm for mobile ad hoc networks. Technical report, 2000, Rutgers University, Piscataway, New Jersey, USA.

[42] L. Tan, X. Zhan, J. Li, and F. Zhao. A novel tree-based broadcast algorithm for wireless ad hoc networks. *International Journal of Wireless and Mobile Computing*, 1(2):156–162, 2006.

[43] Y-C. Tseng, S-Y. Ni, and E-Y. Shih. Adaptive approaches to relieving broadcast storms in a wireless multihop mobile ad hoc network. *IEEE Transactions on Computers*, 52(5):545–557, 2003.

[44] A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–10, Marseille, France, 2008.

[45] G. Virone, A. Wood, L. Selavo, Q. Cao, L. Fang, T. Doan, Z. He, R. Stoleru, S. Lin, and J.A. Stankovic. An assisted living oriented information system based on a residential wireless sensor network. In *Proceedings of the IEEE 1st Conference on Distributed Diagnosis and Home Healthcare*, pages 95–100, Arlington, Virginia, USA, 2006.

[46] Sito web progetto WASP: http://www.wasp-project.org.

[47] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *Proceedings of the 2nd European Workshop on Wireless Sensor Networks*, pages 108–120, Istanbul, Turkey, 2005.

[48] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: A wireless sensor network testbed. In *Proceedings of the 4^{t}h international symposium on Information processing in sensor networks*, pages 483–488, Los Angeles, California, 2005.

[49] B. Williams and T. Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 194–205, Lausanne, Switzerland, 2002.

[50] N. Xu, S. Rangwaka, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 13–24, Baltimore, Maryland, USA, 2004.