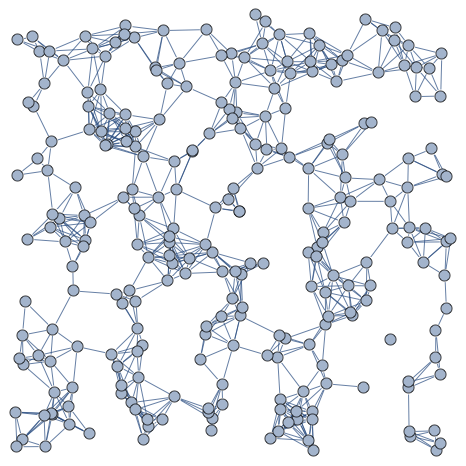
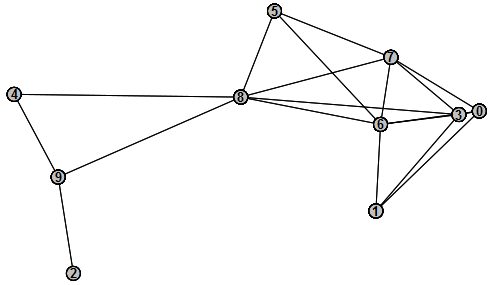
# Capitolo 4 – Progetto logico della soluzione del problema

Dopo aver visto dallo studio di fattibilità che era sostenibile, in termini di consumo energetico, la diffusione d’informazioni sfruttando lo standard Bluetooth 4.0 LE, abbiamo cominciato a progettare una possibile soluzione.

Abbiamo quindi pensato di modellizzare la diffusione delle informazioni allo stesso modo di come si diffondono le epidemie, poiché negli scenari in cui ci siamo posti, un’informazione dovrebbe diffondersi allo stesso modo. Abbiamo ricercato tra gli algoritmi di gossip, chiamati anche algoritmi epidemici, un possibile algoritmo adatto alla nostra situazione.

## Modello di rete

Una rete di dispositivi mobili che comunicano con lo standard Bluetooth 4.0 LE, equivale a una rete di nodi che possono comunicare con tutti i nodi in un determinato raggio da essi. Per questo motivo abbiamo utilizzato un modello di tipo Random Geometric Graph e nella figura 4.1 ne sono riportati due esempi. Questo modello di rete utilizza il concetto di distanza geometrica per determinare quali nodi sono collegati a quali nodi. E’ evidente come questo tipo di modello possa essere utilizzato per modellare reti wireless o reti in cui la caratteristica della distanza fisica tra i nodi è importante e definiamo ρ il valore che rappresenta la soglia entro cui sono possibili trasmissioni tra i nodi. Con questo principio abbiamo potuto simulare il comportamento delle trasmissioni effettuate tramite bluetooth. La posizione dei nodi in una determinata area è stata determinata casualmente, come poi vedremo nel capitolo successivo come saranno scelte le dimensioni delle aree. Questo tipo di modello di rete presenta una bassa *degree-variance*, *spiegazione ?*, ma un’alta *edge-dependecy, spiegazione ?*. Un aspetto fondamentale di questo tipo di reti è che sono più connesse quanto più i nodi sono vicini tra loro, perciò si è reso importante studiare le performance della nostra soluzione in relazione alla distribuzione di nodi per metro quadro. Come si può vedere dalla figura 4.1, difficilmente avremo nodi con un elevato numero di connessioni, al più potrebbe capitare che si formino gruppi di nodi molto vicini quindi molto connessi ma tra gruppi diversi vi siano solo poche connessioni. Può capitare che uno o più nodi siano troppo lontani dagli altri nodi e non riescono a raggiungere ed essere raggiunti da nessuno, quindi saranno isolati.

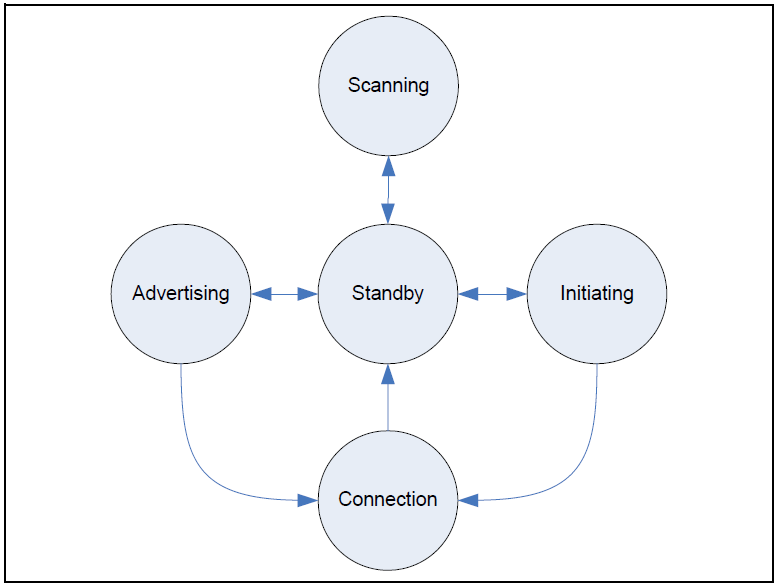
*Figura 4.1: due esempii di Random Geometric Graph*

Come è possibile intuire, la densità di nodi e il raggio d’azione del BLE influenzano molto le performance dell’algoritmo di distribuzione. Più la densità aumenta più il grafico si dirada e si perdono connessioni col risultato di incrementare le probabilità di avere nodi isolati, diventando sempre più sensibile al fattore batteria dei singoli nodi. Infatti basta che un nodo esaurisca la batteria che magari il resto della rete non è più raggiungibile in quanto mancano altre connessioni ridondanti da poter usare. Al contrario invece,più la densità diminuisce più i nodi sono vicini rendendo la rete connessa e creando connessioni ridondanti, decrementando notevolmente la possibilità di avere nodi isolati e meno sensibile alla decadenza energetica del singolo nodo. Per questo motivo abbiamo testato il nostro algoritmo con diverse densità, anche di differenti ordini di grandezza in modo da coprire casistiche limite per le ipotesi di lavoro fatte, appunto per verificare i limiti della nostra soluzione.

## Bluetooth 4.0 Low Energy

“… mancano un po’ di spiegazioni sul BLE … “

Il Link Layer del protocollo BLE funziona secondo una macchina a stati, come mostrato in figura 4.2.



*Figura 4.2: Bluetooth LE macchina a stati del Link Layer*

“…altre info sul BLE….”

## Algoritmo di Propagazione

Abbiamo scelto di applicare un algoritmo di tipo “rumor mongering”, in altre parole un algoritmo pensato per simulare la diffusione dei pettegolezzi tra le persone. Gli algoritmi di tipo rumor mongering sono di algoritmi SIR, *Suscettible-Infected-Removed*, cioè algoritmi in cui inizialmente tutti i nodi sono suscettibili alla malattia, una volta ricevuta la malattia diventano infettati iniziano a loro volta a diffonderla ; infine in base a certi criteri un nodo infetto smette di diffondere la malattia e diventa rimosso dal processo di diffusione. Nel nostro caso, la malattia è la notizia da diffondere.

I criteri di terminazione sono:

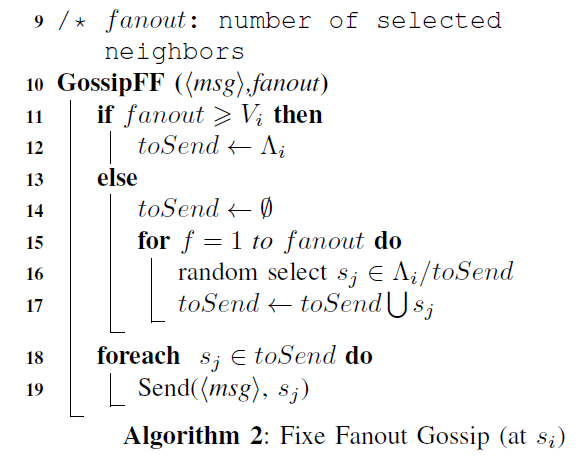
* ***Blind***: un nodo infetto decide di interrompere la diffusione in base al suo stato interno.
* ***Feedback***: un nodo contattato risponde dicendo se giù conosce oppure no l’informazione. L’infetto, in base alla risposta decide.
* ***Coin***: il nodo infetto si fermerà con una probabilità 1/k, dove k è il numero di nodi contattati.
* ***Counter***: il nodo infetto incrementa un contatore per ogni informazione inviata con successo. Si ferma quando il contatore supera una soglia prefissata.

Vi sono anche tre tipologie di approcci alla diffusione:

* ***Push***: il nodo infetto invia attivamente a nodi non ancora infetti l’informazione.
* ***Pull***: il nodo infetto conserva l’informazione, mentre devono essere i nodi suscettibili a chiedere al nodo infetto se ci sono nuove informazioni e nel caso farsele mandare.
* ***Push&Pull***: è l’unione delle precedenti, ovvero il nodo infetto manda attivamente ai nodi adiacenti un avviso che lui ha una nuova informazione e i nodi interessati contatteranno direttamente il nodo infetto per farsi mandare la nuova informazione.

### Algoritmo Fixed Fanout

Tra i vari algoritmi di gossip abbiamo scelto di usare l’algoritmo di Fixed Fanout. Quest’algoritmo prevede che ogni nodo, una volta infettato, si prodighi a infettare a sua volta un certo numero di nodi ai quali è connesso e questo numero prefissato, si chiama fixed fanout, caratteristica che dal nome all’algoritmo stesso. Quest’algoritmo utilizza come strategia di stop, il contatore d’invii e il fixed fanout è il limite. In figura 4.2 è riportato lo pseudocodice che descrive tale algoritmo.



*Figura 4.3: pseudocodice algoritmo di Fixed Fanout*

## Soluzione: Algoritmo Dynamic Fanout

La nostra soluzione è una personalizzazione di questo algoritmo. L’idea di base è la stessa, ma per affrontare il problema del consumo energetico dei dispositivi, abbiamo modificato un parametro e introdotto uno nuovo con lo scopo di far sprecare meno energia possibile ai dispositivi per garantirne una maggior operatività ed efficacia.

I due parametri in questione sono:

* ***Dynamic Fanout***
* ***Advertising Limit***

Come si evince, come nel caso del Fixed Fanout, anche nella nostra soluzione il parametro principale attribuisce il nome all’algoritmo.

Così com’è, il nostro algoritmo lavorerebbe solo su un’informazione e basta. Per poterlo generalizzare e permettergli di lavorare con più informazioni, dobbiamo renderlo un algoritmo di tipo SIRS, *Suscettible-Infected-Removed-Suscettible*. Ciò fa si che una volta entrato in stato rimosso, un nodo possa tornare di nuovo suscettibile per altre informazioni e quindi partecipare alla diffusione di altre informazioni, nel caso vi siano.

Per modellare un comportamento di tipo SIRS abbiamo accoppiato ogni stato dell’algoritmo a uno o più stati del BLE, descritti nella Sezione 4.2. Le relazioni sono:

* Suscettibile 🡨🡪 Initiating
* Infetto 🡨🡪 Advertising
* Rimosso 🡨🡪 Standby/Scanning

Eventualmente anche lo stato intiating per un nodo può essere visto come stato rimosso per informazioni vecchie o già transitate da quel nodo per le quali non vi è più alcun interesse.

### Dynamic Fanout

Il Dynamic Fanout nasce con lo stesso principio del Fixed Fanout, ovvero è il limite di trasmissioni che ogni nodo fa prima di smettere di diffondere l’informazione. Il motivo per cui si chiama “dinamico” è perché l’abbiamo reso dipendente dallo stato dell’ambiente circostante, ovvero dal numero di dispositivi che esso è in grado di raggiungere e dallo stato della batteria. In questo modo, questo limite varia dinamicamente adattandosi ai cambiamenti dell’ambiente e del dispositivo stesso, cercando di calibrare un limite che possa offrire buona diffusione, ma che sia anche proporzionato alle “energie” che il dispositivo ha, per non fargli fare uno sforzo eccessivo e prolungarne l’operatività. Abbiamo progettato il Dynamic Fanout in modo che abbia un comportamento molto reattivo per numeri molto bassi in termini di numero di nodi e di livello di batteria, per garantire che, soprattutto in presenza di pochi nodi, si riesca comunque a raggiungere il maggior numero di nodi.

### Advertising Limit

Abbiamo progettato l’Advertising Limit come limite alla pubblicità, cioè alle azioni di push, inutili e come secondo criterio di stop alla diffusione dell’informazione. L’idea è che quando si pubblicizza un messaggio e nessuno risponde, le possibilità sono principalmente tre: i nodi adiacenti sono impegnati in altre azioni e non possono rispondere, i nodi adiacenti sono comunque suscettibili ma hanno scartato la mia pubblicità perché bloccati in attesa di qualche timeout del protocollo BLE, nessuno dei nodi adiacenti non è interessato alla mia informazione. Nei primi due casi, l’azione logica da fare è attendere un certo periodo e riprovare a pubblicizzare la mia informazione, mentre nel terzo caso non ha più senso pubblicizzare ancora perché nessuno è più interessato e quindi è meglio fermarsi. Ogni nodo ha un contatore di pubblicità andate a vuoto; quando questo contatore raggiunge il valore di Advertising Limit, il nodo smette di trasmettere e passa in stato di rimosso. Questo parametro è molto utile perché più il numero di nodi aumenta, più è difficile per ogni nodo raggiungere il limite di trasmissioni imposto dal Dynamic Fanout, quindi ogni nodo non terminerebbe mai di pubblicizzare l’informazione in suo possesso, con il risultato di consumare energia inutilmente e non andando mai in stato rimosso; proprio in questi casi interviene l’ Advertising Limit a fermare le trasmissioni del nodo.

## Simulazione

(va nel capitolo 2?)Come strumento di simulazione è stato scelto il software OMNeT++, un framework e libreria ad eventi discreti principalmente per la simulazione di reti, scritto in C++. Questo programma offre molti strumenti utili alla simulazione e all’analisi delle performance di reti di qualunque genere. Inoltre le sue librerie coprono quasi la totalità dei protocolli di comunicazione, anche se non quello del BLE. Come poi illustreremo nel Capitolo 5, OMNeT++ permette di registrare i log temporali delle trasmissioni e di raccogliere statistiche su parametri su cui si vogliono fare analisi a posteriori. Permette inoltre, tramite appositi file, di impostare diverse configurazioni per il sistema che noi progettiamo e tramite i quali possiamo specificare la quantità di esecuzioni che vogliamo lanciare per ogni configurazione. Ciò permette di avere una piccola base statistica su cui fare analisi. Nel caso si voglia fare una sola esecuzione, è possibile visualizzarne l’andamento su di un tool grafico che rappresenta la rete nel suo complesso e i pacchetti in movimento da sorgente a destinazione; eventuali messaggi vengono stampati sulla console di questa tool grafico. La nostra scelta per quanto riguarda la simulazione e la raccolta/analisi dei dati è stata quindi l’uso di questo valido strumento.