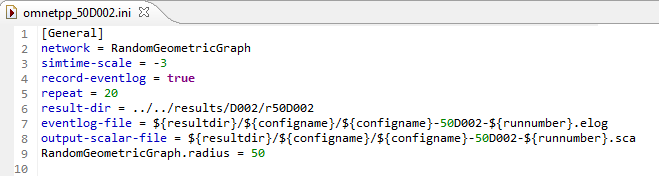
# Capitolo 5 – Architettura del sistema

## Simulatore

(va nel capitolo 2-background?)In questa sezione andremo a illustrare i diversi file utilizzati per l’implementazione fatta sul simulatore e i relativi contenuti, per introdurre alcune terminologie utilizzate in questo capitolo. In generale ogni simulazione necessita di un NED file che definisca il layout della rete, di un INI file che definisca i parametri per inizializzare la simulazione, la rete e tutti i suoi componenti, tra cui anche in quale NED file si trova la rete da usare e una serie di C++ file, uno per ogni componente che ha un “comportamento” da seguire ovvero dove scriveremo le azioni che il nostro componente dovrà fare all’accadere di certi eventi (nel nostro caso la rete in sé non ha comportamento quindi non ha il suo c++ file).

### INI file

(va nel capitolo 2-background?)Come detto nella Sezione 5.1, un INI file è di fatto un file di inizializzazione necessario alla corretta configurazione della rete, dei suoi componenti e dell’ambiente di simulazione se necessario; un INI file ha estensione “.ini”. Questo framework offre la possibilità di poter differenziare all’interno di un .ini file diverse configurazioni , tra le quali noi possiamo scegliere al momento dell’esecuzioni tramite una finestra di dialogo a video al momento dell’esecuzione oppure specificarne direttamente una nella configurazione di lancio dell’editor. Ciò torna utile quando si voglio preparare script per lanciare una lunga lista di simulazioni senza dover ogni volta selezionare a mano la configurazione desiderata. Di default si ha anche una configurazione generale e tutti i parametri che scriveremo dentro questa configurazione saranno automaticamente usati da tutte le altre configurazioni, previa forzata ridefinizione. Nella figura 5.1 abbiamo un pezzo di uno dei file d’inizializzazione utilizzato per le simulazioni. In particolare è il file relativo alla simulazione di raggio 50 m e densità 0.02 unità/m2. Come si vede in figura 5.1, nella sezione *General* abbiamo deviti i parametri comuni a tutte le configurazioni, quali il modello di rete da utilizzare, la scala del tempo di simulazione, il numero di ripetizioni da eseguire per ogni singola configurazione, settaggi vari per il salvataggio dei dati raccolti e infine il parametro del raggio.



*Figura 5.1: sezione General del file omnetpp:50D002.ini.*

### NED file

(va nel capitolo 2-background?) I file NED sono file necessari alla descrizione della struttura della rete, della sua topologia, di tutto ciò che riguarda il suo layout. Tramite questi file possiamo definire il layout di un singolo modulo semplice, di un modulo composto, di un canale di connessione o dell’intera rete. I file NED hanno estensione “.ned”. Se il componente modellato nel file ned ha un comportamento dinamico o deve eseguire azioni all’accadimento di un evento, allora deve essere presente anche un file con estensione C++ e con lo stesso nome del file ned.

Nel caso nel file ned si costruisca una rete, bisogna specificare anche i collegamenti tra i singoli nodi, specificando nell’apposita sezione i collegamenti tra determinati nodi, oppure specificando funzioni atte a creare i collegamenti in modo dinamico e/o con eventuali elementi di casualità.

### 

## Modello di rete

Come detto nella Sezione 4.1, il modello di rete utilizzato è quello del Random Geometric Graph, una rete di nodi rappresentata da un grafo i cui nodi sono geometricamente connessi a tutti i nodi entro un dato raggio ρ da essi. Nella figura 5.1 sono rappresentati due esempi di RGG; il grafico di destra è una visualizzazione di un esempio di rete rappresentato dal tool grafico di OMNeT++.

La generazione della nostra rete è dipendente da:

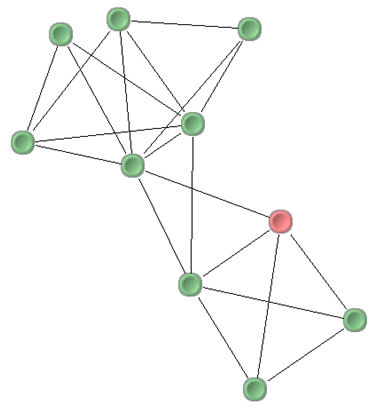
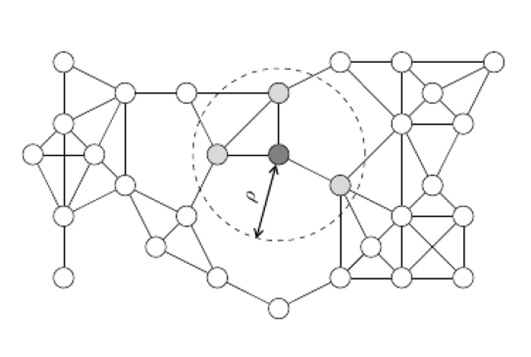
* Numero di nodi che compongono la rete,
* Raggio d’azione del BLE,
* Densità dei nodi.

La scelta di operare a diversi livelli di densità è dovuta al fatto di aver voluto studiare il comportamento della nostra soluzione in alcuni scenari urbani possibili per diverse distribuzioni geografiche sul territorio, in altre parole poter studiare il variare del comportamento del nostro algoritmo sia in situazioni di centri urbani normalmente molto densamente popolati, ad esempio medie/grandi città, sia in centri urbani con una più bassa concentrazione urbana che rappresentano la maggioranza dei paesi italiani.

Per queste considerazioni ovviamente abbiamo paragonato un nodo a un abitante, perché mediamente ogni persona possiede uno smartphone. Abbiamo scelto come prima e più grande densità 0.02 abitanti per metro quadro perché ci è sembrato una densità già sufficientemente alta per avere ottimi valori in termini di perfomance, quindi simulare a densità superiori non ci è sembrato utile. Quello che invece abbiamo voluto studiare è come si degradano le prestazioni al diradarsi della concentrazione urbana, ed ecco il perché della scelta delle alte densità.

Le densità scelte sono:

1. D = 0.02 nodi/m2,
2. D = 0.01 nodi/m2,
3. D = 0.008 nodi/m2,
4. D = 0.005 nodi/m2,
5. D = 0.001 nodi/m2.



*Figura 5.2: esempio di Random Geometric Graph.*

Come abbiamo detto sopra, la generazione dell’area dipende anche dal numero di nodi e anche per questo abbiamo scelto di fare più prove con diversi valori di utenti.

I numeri di nodi scelti sono:

1. N = 2,
2. N = 5,
3. N = 10,
4. N = 30,
5. N = 50,
6. N = 80,
7. N = 100,
8. N = 200,
9. N = 500,
10. N = 1000.

L’aver fissato i valori di densità ci ha permesso di calcolare poi, per ogni numero di nodi scelto, di impostare la grandezza dell’area in maniera sensata, senza avere casi di forte dispersione in cui ovviamente il nostro algoritmo non funzionerebbe perché non pensato per affrontare tali situazioni.

Infine l’ultimo parametro che abbiamo voluto far variare è stato il raggio d’azione del BLE. Le specifiche tecniche dicono che il raggio d’azione è di circa 50 m, ma chi ha definito il protocollo BLE (chi è? Azienda Blutooth?) non ha né imposto né definito, alcuna distanza specifica che le case costruttrici di trasmettitori BLE devono seguire; cosa che fu fatta nel caso del suo predecessore Bluetooth Classic. Ciò fa sì che non si abbia un valore unico di raggio d’azione, ma ogni costruttore implementa il proprio trasmettitore più o meno potente. In media è stato rilevato che il raggio d’azione è circa 50 metri (*cit?)*. Dato che i nodi della nostra rete sono smartphone e trasmettitori installati dalle varie case produttrici sono tutti diversi, abbiamo scelto di simulare per due valori di raggio, uno pessimistico e uno ottimistico.

I valori di raggio ρ scelti sono:

* R = 15 m (pessimistico),
* R = 50 m (ottimistica, come la media rilevata).

Abbiamo scelto di non spingerci oltre col valore di raggio poiché ci è sembrato difficile che dispositivi mobili come i cellulari possano arrivare a 100 m o anche oltre come alcuni chip per impiego industriale.

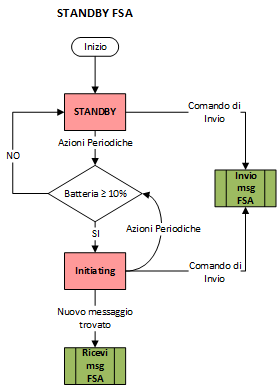
In simulazione, la generazione della rete è fatta in modo automatico e la distribuzione dei nodi nell’area in modo automatico, come pure la costruzione dei collegamenti tra i nodi. I vari parametri sono stati inseriti negli appositi file d’inizializzazione. Dopo che ogni nodo è stato creato e disposto nell’area, nel file ned che definisce la rete, vi è una sezione chiamata c*onnections* in cui si definiscono i criteri con cui stabilire le connessioni tra i nodi. In questa sezione valutiamo per ogni nodo della rete, quali degli altri nodi sono entro il raggio ρ definito nei parametri d’inizializzazione e inserisce una connessione bidirezionale tra essi. Sempre nel file ned che definisce la rete, è specificato il tipo di connessione come un collegamento avente velocità di trasferimento di 1Mbps, la velocità di trasmissione da specifiche BLE.

*(ci vuole uno screen della pezzo di codice ??)*

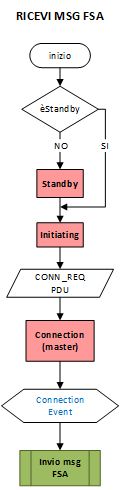
## Algoritmo Dynamic Fanout

Come detto nella Sezione 4.4 la nostra soluzione è una personalizzazione dell’algoritmo di gossip Fixed Fanout, presentato nella Sezione 4.3.1. Abbiamo modificato il modo in cui è determinato il limite del numero di trasmissioni che ogni nodo può effettuare, rendendolo dinamico e non più statico come l’algoritmo originale. Per questo motivo abbiamo chiamato la nostra soluzione: Dynamic Fanout. Abbiamo inoltre inserito due metodi di terminazione della trasmissione, cercando un compromesso tra efficienza e risparmio energetico per tutte le situazioni incontrate negli scenari possibili che abbiamo studiato. I parametri che abbiamo modellato sono il DF e l’AL. Entrambi sono di tipo contatore e il DF in particolare il DF contiene anche una componente di tipo “blind*”* dipendente dallo stato interno della batteria del dispositivo.

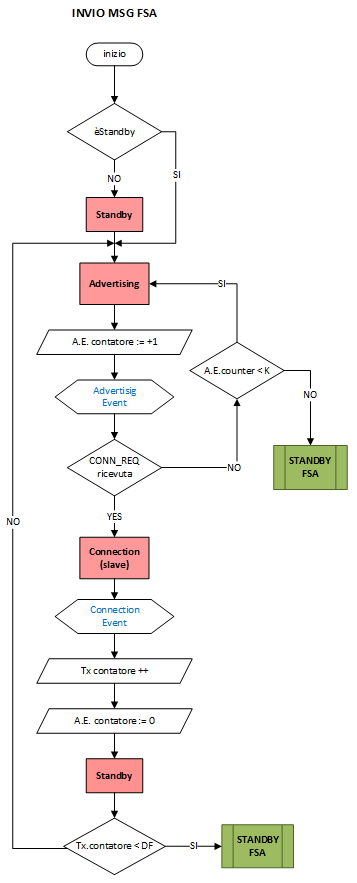
Il nostro algoritmo agisce fondamentalmente sulla macchina a stati del BLE, cambiando stato in occorrenza di particolari eventi, quali per esempio la presenza di un nuovo messaggio da inviare o ricevere, aggiornando periodicamente lo stato dell’algoritmo con i nuovi parametri aggiornati, controllando che eventuali trasmissioni rimangano nei limiti prefissati dai parametri e quando uno dei due è raggiunto agire di conseguenza. Abbiamo progettato l’algoritmo in modo che usi il dispositivo finché la sua batteria è più del 10%, poiché non volevamo che l’algoritmo stesso potesse consumare le ultime riserve di energia del dispositivo, lasciando il tempo all’utente di poter raggiungere un eventuale fonte di ricarica o permettergli di usare servizi più vitali che il dispositivo stesso può offrire. Per questo motivo l’algoritmo se nei suoi controlli periodici rileva che la batteria del dispositivo è inferiore al 10%, ferma il BLE in stato di standby rendendolo quindi inabile a ricevere e inviare successivi messaggi finché il livello di batteria non risale sopra il 10%. Nelle figure 5.3, 5.4 e 5.5 sono riportati i diagrammi che descrivono il generale comportamento dell’algoritmo di DF e sono rispettivamente quello di *Standby*, quello di *Ricezione di un nuovo messaggio* e quello di *Invio di un messaggio*.



*Figura 5.3: Diagramma di flusso della macchina a stati Standby dell'algoritmo DF.*



*Figura 5.4: Diagramma di flusso della macchina a stati Ricevi Messaggio dell'algoritmo DF.*



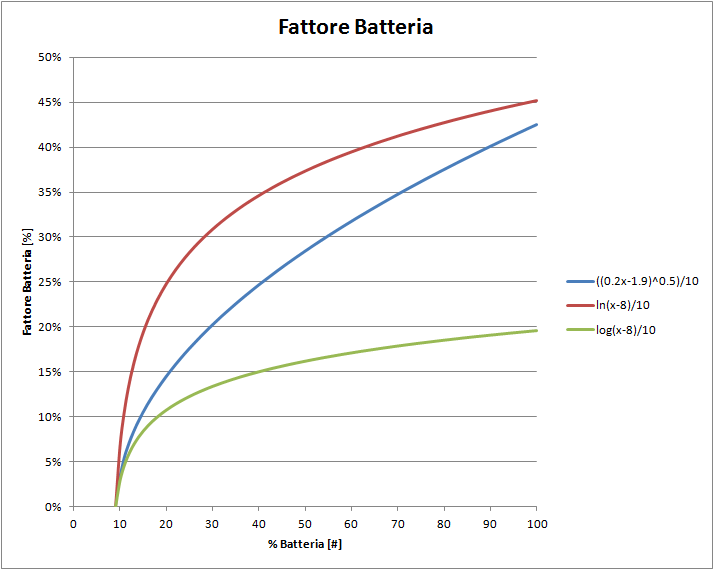
*Figura 5.5: Diagramma di flusso della macchina a stati Invia Messaggio dell'algoritmo DF.*

### Dynamic Fanout

Il Dynamic Fanout ha il compito di fermare il dispositivo dopo un certo numero di trasmissioni effettuate con successo per una certa informazione. Questo valore limite è calcolato in modo dinamico, dipendente dal livello di batteria del dispositivo e dal numero di nodi che il dispositivo riesce a percepire, quindi al numero di nodi ai quali può potenzialmente connettersi. Abbiamo voluto che questo parametro avesse un particolare andamento e rapidità di risposta in particolari situazioni e altri andamenti in altri momenti. Per ottenere ciò abbiamo provato a modellare il comportamento di questo parametro attraverso diverse funzioni ottenendo quindi comportamenti più o meno conservativi, secondo la funzione scelta.

Il calcolo del DF è composto da due fattori: un primo fattore che tiene conto del livello di batteria del dispositivo, chiamato *Fattore Batteria*, un secondo fattore che ha la funzione di correggere l’andamento globale della funzione risultante all’aumentare del numero di nodi. La nostra idea è di ottenere dal Fattore Batteria una percentuale che rappresenta la quantità di nodi, tra quelli percepiti, che il dispositivo prenderà in considerazione come suo limite di trasmissione cioè come suo DF. Più batteria un dispositivo ha più alta sarà la percentuale di nodi cui potremo trasmettere l’informazione. In figura 5.6 riportiamo in grafico tre funzioni batteria studiate, ognuna con un andamento diverso; esse sono rispettivamente:

* (curva rossa)
* (curva blu)
* (curva verde)

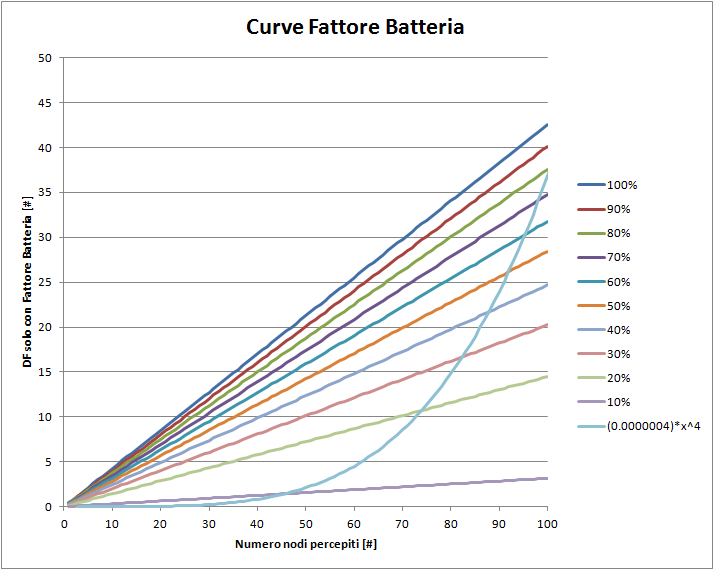


*Figura 5.6: Andamenti Funzioni Batteria.*

Tutte e tre le funzioni sono divise per un fattore dieci per necessità di ottenere un valore percentuale. Dalla figura 5.6 si nota come diversi tipi di funzioni diano differenti curve di risposta, più o meno conservative e più o meno reattive per valori tra 10% e il 20% di batteria. Noi abbiamo scelto di utilizzare la funzione con la radice quadrata (curva blu) rispetto alla funzione col logaritmo naturale (curva rossa), poiché non troppo aggressiva per valori bassi di batteria ma nemmeno troppo conservativa e perché al crescere della percentuale di batteria le due funzioni tendono allo stesso valore massimo o quasi.

Fattore Batteria:

In fase sperimentale, questa nostra scelta si è rivelata poi abbastanza buona dal punto di vista prestazionale. Il Fattore Batteria è soltanto un fattore di “scelta” sulla totalità dei nodi, quindi un andamento fortemente crescente tra i 10% e il 30% porterebbe ad avere dispositivi, con relativamente poca batteria, che continuano a trasmettere informazione se possibile. Se il numero di nodi percepiti è basso, non crea alcun problema, ma se il numero di dispositivi aumenta, ciò comporta una forzatura nelle trasmissioni e un consumo eccessivo di batteria perché il dispositivo tenderebbe a fare un numero di trasmissioni quasi uguale a dispositivi col 60-70% di batteria. Ora abbiamo solo un fattore che definisce il DF come una parte dei nodi percepiti. E’ ovvio notare che ciò scala linearmente col numero dei nodi e porterebbe, pur avendo una riduzione dovuta al Fattore Batteria, ad avere un DF sempre crescente, tendente a infinito e inutile perché non sarebbe mai raggiunto, ma permetterebbe comunque a tutti i dispositivi di sprecare energia. Per affrontare questo problema abbiamo pensato di inserire nella valutazione del numero di nodi un Fattore di Correzione che permetta di controbilanciare la lineare crescita del Fattore Batteria. Quello che abbiamo pensato è stato inserire un Fattore di Correzione che andasse ad agire con decisione al crescere del numero di nodi, limitandone o addirittura diminuendo il DF finale.



*Figura 5.7: Curve Fattore Batteria più cura Fattore di Correzione.*

Abbiamo scelto di inserire un fattore correttivo perché abbiamo fatto l’assunzione che più il numero di nodi aumenta, più ogni dispositivo può trasmettere un poco meno, nell’ottica che il lavoro di diffusione sia distribuito su tutti i nodi adiacenti senza sovraccaricare il singolo dispositivo. Questa soluzione permette inoltre di diminuire il traffico sia sui canali di trasmissione e, insieme all’AL, sia sui canali di advertising. Nella figura 5.7 sono riportate in grafico dieci curve di Fattore Batteria, una ogni 10% di batteria, e il Fattore di Correzione. Il Fattore di Correzione è una funzione di quarto grado cui è stato applicato un coefficiente piccolo per ritardarne l’effetto correttivo e verso valori alti di numero di nodi.

Fattore di Correzione:

In figura 5.8 invece è mostrato su grafico come l’andamento del DF (curva verde) quando uniamo il Fattore Batteria (curva blu) col Fattore di Correzione (curva rossa). Il DF è quindi calcolato con la seguente formula:

Dynamic Fanout:

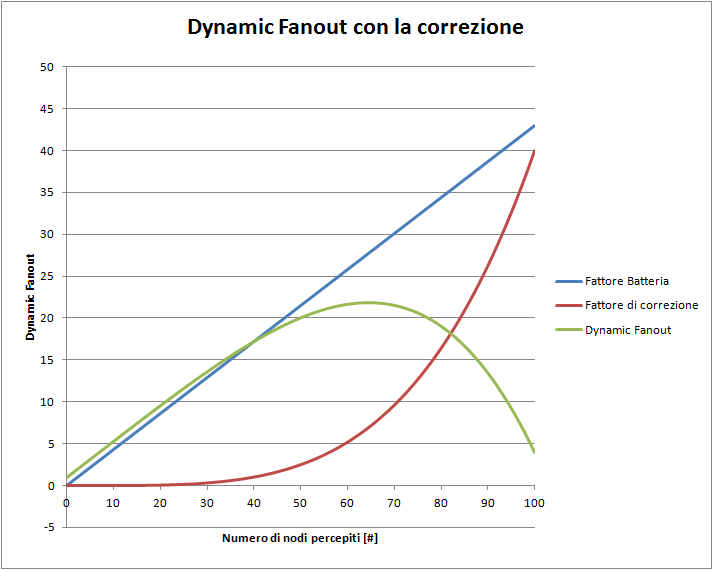
Ciò che si ottiene come DF è un andamento crescente fino a raggiungere il massimo, che varia al variare del livello di batteria, e poi decresce a causa della forte correzione. La traslazione dovuta al termine costante serve a garantire in qualsiasi caso un DF di uno, nella situazione di solo uno o due nodi percepiti. Com’è logico notare dal grafico, per valori superiori a 100 nodi o per curve di batterie riguardanti livelli energetici inferiori al 100% il valore del DF è zero o negativo. Per questo motivo abbiamo aggiunto un asintoto orizzontale, in questo modo ogni curva avrà un suo asintoto e ci sarà mai un DF nullo o negativo. Abbiamo deciso di inserire un asintoto per grandi valori perché in queste particolari situazioni di altissima densità una dipendenza dal numero di nodi è inutile ai fini dell’efficienza e del risparmio energetico. L’asintoto tiene comunque conto del livello di batteria per il discordo di risparmio energetico e di non sforzare troppo il dispositivo. Abbiamo pensato che l’asintoto debba dipendere valere una percentuale del valore massimo raggiunto da ogni curva, così da renderlo dipendente dal livello di batteria del dispositivo.

Asintoto:

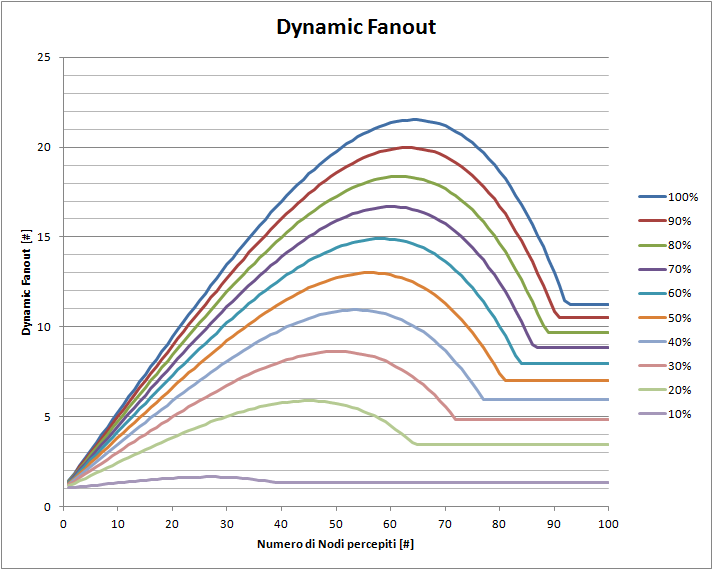
Quest’asintoto è preso in considerazione solo dopo aver superato il massimo della funzione; per costruzione sappiamo che ve n’è solo uno. Nella formula dell’asintoto abbiamo inserito una traslazione di un’unità verso l’alto per rimaner coerenti con le scelte fatte fin ora; come vedremo dopo, se non vi sono elementi di traslazione anche nell’asintoto non vi saranno.

L’algoritmo semplicemente valuta due casi: per numero di nodi minore di 30 prende in considerazione solo la curva, per numero di nodi maggiore valuta il massimo tra la curva e l’asintoto.

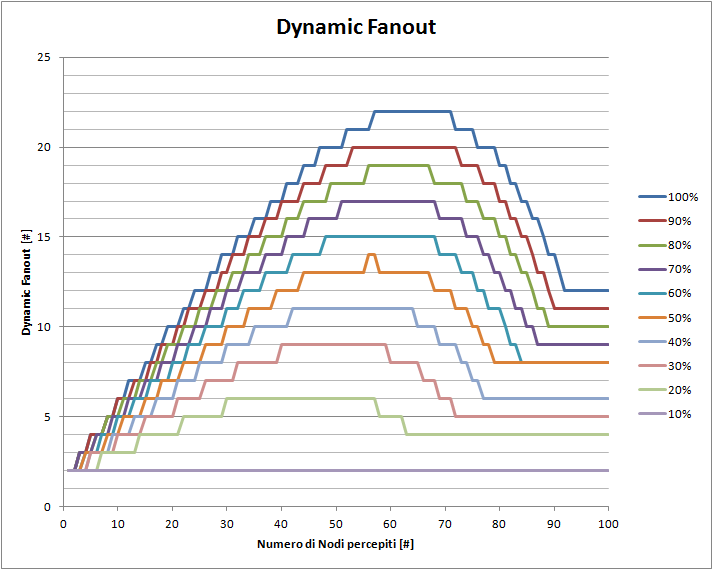
Infine in figura 5.9 è riportato su grafico il risultato finale del calcolo del Dyanmic Fanout con asintoti orizzontali. Come prima, sono mostrate solo le dieci curve per i dieci principali livelli di operatività.



*Figura 5.8: Andamento Dynamic Fanout con correzione.*



*Figura 5.9: Curve del Dynamic Fanout.*

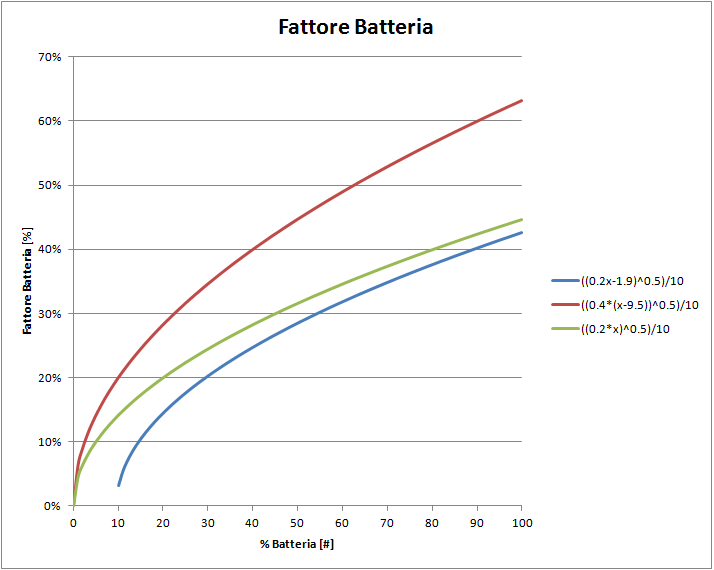


*Figura 5.10: Curve del Dynamic Fanout arrotondate per eccesso.*

Dato che il DF esprime il limite di trasmissioni che un nodo può fare, non ha senso avere valori razionali perché è logico che il contatore di trasmissioni conteggi solo per numeri interi. Abbiamo applicato un arrotondamento per eccesso alle curve mostrate in precedenza in figura 5.9. Mostriamo il risultato in figura 5.10. Applicando tale arrotondamento abbiamo ottenuto un innalzamento del valore minimo a 2. Ciò non crea alcun problema, anzi rende l’algoritmo ancora più reattivo per valori piccoli di numero nodi, mentre resta trascurabile il suo effetto per valori medio - grandi.

Quello riportato in figura 5.10 è il risultato scelto. Abbiamo studiato anche molte alte funzioni con andamenti più o meno conservativi e/o permissivi e reattivi. Di seguito riportiamo solo due casi: uno più permissivo e uno più conservativo con le relative formule delle funzioni utilizzate. Riporteremo solo i grafici finali, quelli degli andamenti del DF con asintoti senza gli arrotondamenti, per comprendere meglio le caratteristiche delle curve. Ciò che varierà saranno le funzioni utilizzate per il calcolo del Fattore Batteria e del Fattore di Correzione, mentre il metodo del calcolo del DF, asintoto compreso, rimane uguale.

In figura 5.11 sono riportate le due nuove funzioni per i due Fattori Batteria che caratterizzano i diversi DF, più la funzione utilizzata nella soluzione per fare un confronto. La curva blu è la soluzione che abbiamo implementato, la curva rossa è la funzione utilizzata per il caso permissivo, mentre la curva verde è la funzione utilizzata per il caso conservativo.



*Figura 5.11: Funzioni Fattore Batteria: blu) soluzione, rosso) permissiva, verde) conservativa.*

Uno dei casi di DF permissivo nasce dalla sola manipolazione dei termini della funzione utilizzata nella nostra soluzione. Il Fattore Batteria ha coefficiente sulle x pari a 0.4, più grande di quello utilizzato nella soluzione (0,2) e inoltre non ha nemmeno il fattore di traslazione. Ciò lo fa crescere più velocemente, vedi figura 5.11 (curva rossa).

Fattore Batteria:

Ovviamente aumentando tale coefficiente si ottiene una crescita sempre più ripida, quindi una reattività sempre maggiore. Forzare la reattività del sistema sì può giovare alle prestazioni, ma fino ad un certo punto perché alla fine il limite DF sarà talmente alto per ogni situazione di nodi percepiti che tale limite non sarà mai raggiunto diventando pressoché inutilizzato.

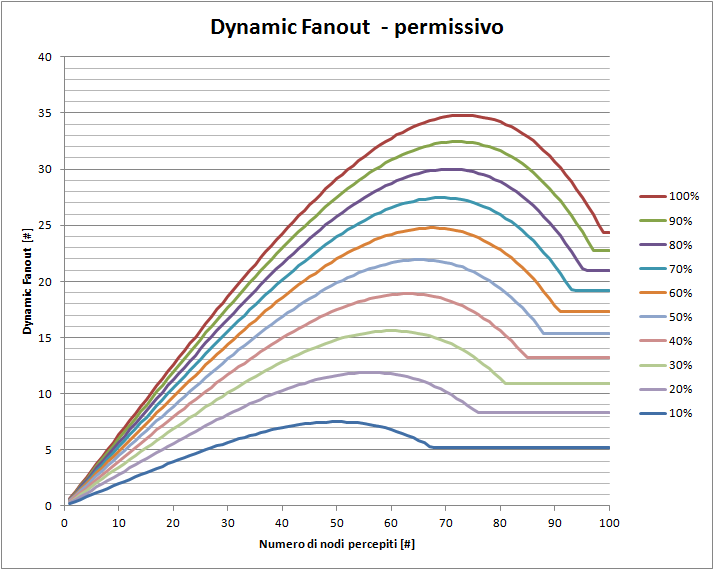
Il Fattore di Correzione rimane lo stesso.

Fattore di Correzione:

L’asintoto invece è stato alzato al 70% del valore massimo:

Asintoto:

Il risultato finale è riportato nel grafico in figura 5.12. Come si può notare dal grafico, abbiamo ottenuto un incremento del valore massimo di circa 60% o addirittura superiore, dipende dalla curva.



*Figura 5.12: Dynamic Fanout permissivo.*

Tra le varie casistiche studiate uno dei casi di DF conservativo trovato ha un Fattore Batteria ha simile a quello utilizzato nella soluzione, infatti, il suo andamento come si vede in figura 5.11 (curva verde) è molto simile, ma implementa un Fattore di Correzione diverso che agisce prima. Anche in questo caso è stato tolto il fattore di traslazione perché altrimenti il risultato sarebbe stato molto conservativo per certe curve, escludendo inoltre tutti i dispositivi nell’intorno del 10% di batteria perché il limite sarebbe sempre stato zero.

Fattore Batteria:

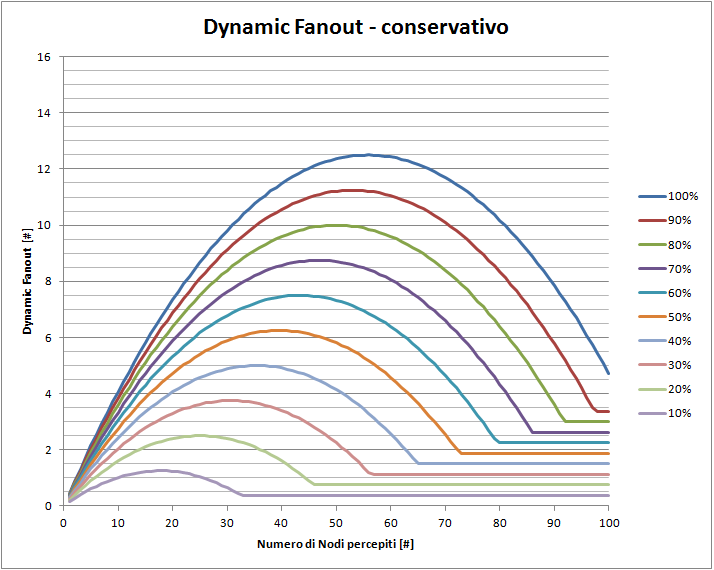
Il Fattore di Correzione è cambiato; ora il suo coefficiente è stato aumentato e il grado della funzione diminuito, per rendere percepibile la sua correzione molto prima.

Fattore di Correzione:

Anche l’asintoto è stato cambiato e scelto più basso, solo il 30% del massimo delle curve.

Asintoto:

Il risultato finale è riportato nel grafico in figura 5.13. Come si può notare dal grafico, abbiamo ottenuto una diminuzione del valore dei massimi da un 40% a un 50% rispetto alla soluzione implementata.



*Figura 5.13: Dynamic Fanout conservativo.*

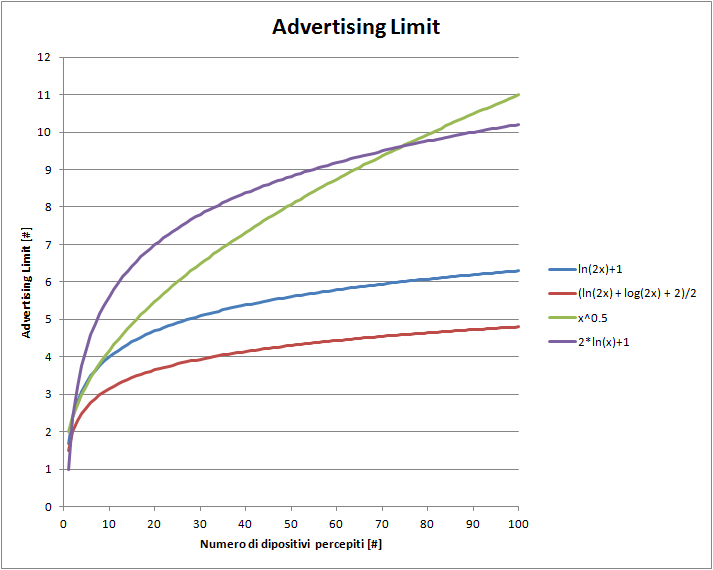
### Advertising Limit

L’Advertising Limit, come descritto nella Sezione 4.4.2, ha il compito di fermare il dispositivo dal continuare a pubblicizzare l’informazione dopo un certo numero di tentativi di advertising senza successo consecutivi. Questo valore limite è calcolato in modo dinamico, dipendente solamente dal numero di nodi che il dispositivo riesce a percepire, quindi al numero di nodi cui può potenzialmente connettersi. Non abbiamo inserito una dipendenza dal livello della batteria perché la richiesta energetica di un singolo messaggio di advertising è veramente molto bassa, tale che non grava praticamente nulla sul consumo energetico medio del dispositivo. Rimane comunque il fatto che continuare all’infinito a trasmettere qualcosa che richiede poca energia, si traduce in un considerevole consumo. Ecco per cui questo parametro tiene conto solo del numero di nodi percepiti, con lo scopo di far capire al dispositivo quanto i nodi intorno a lui non sono più interessati a ciò che ha da trasmettere e che quindi può fermarsi e mettersi in ascolto alla ricerca di altri nuovi messaggi da trasmettere.

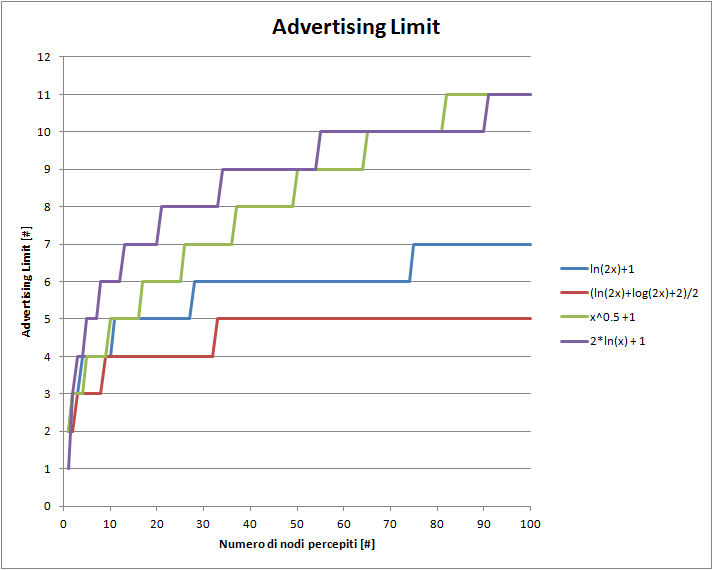
Esattamente come per il Dynamic Fanout, abbiamo ricercato delle funzioni che potessero dare a questo parametro il comportamento consigliato. Sempre come il DF, abbiamo pensato che fosse necessario che per valori bassi di numero di nodi percepiti, il parametro avesse una buona reattività nell’aumentare e adattarsi all’eventuale rapida crescita dell’ambiente. Dato che il processo di advertising richiede poca energia, non abbiamo trovato la necessità di cercare comportamenti conservativi, anzi è stato necessario modellare un comportamento più permissivo per avere buone prestazioni in fase di simulazione. Questo è dovuto dalla presenza dei timeout di attesa connessione da parte di chi sta richiedendo l’informazione. Quando un dispositivo pubblicizza un’informazione, tutti i nodi che ricevono la pubblicità e non hanno l’informazione ne faranno richiesta, ma solo il primo richiedente sarà accontentato mentre tutti gli altri staranno in attesa di connessione finché il rispettivo timeout scade. Ciò crea un problema, in altre parole mentre un dispositivo è in attesa di connessione, non può sapere se il mittente ha “scelto” lui come destinatario, l’unica cosa che può fare è aspettare che arrivi il messaggio. Durante quest’attesa il dispositivo rimane “sordo” a qualsiasi altra comunicazione quindi ignorerà anche altre pubblicità della stessa informazione fatte da altri nodi. Ci siamo resi subito conto che un solo evento di advertising andato a vuoto non significa assolutamente che tutti i nodi attorno al dispositivo sono già stati tutti infettati. Per questo motivo abbiamo progettato l’AL in modo che possa dare al nodo un buon grado di certezza nel capire che i nodi intorno a lui non sono più interessati.

Anche in questo caso abbiamo valutato più funzioni, dai comportamenti più o meno permissivi. In figura 5.14 sono riportate su grafico le seguenti funzioni:

* (curva blu)
* (curva rossa)
* (curva verde)
* (curva viola)



*Figura 5.14: Funzioni dell'Advertising Limit.*



*Figura 5.15: Advertising Limit arrotondato per eccesso.*

In figura 5.15 sono riportate le funzioni arrotondate per eccesso, utilizzate dall’algoritmo. Come si può vedere da entrambe le figure 5.14 e 5.15 abbiamo studiato quattro funzioni di cui due molto rapide nel crescere per valori bassi di numero di nodi e due meno reattive. La funzione che abbiamo scelto di implementare nell’algoritmo è la seguente (figura 5.14, curva blu):

Advertising Limit:

Nonostante abbiamo scelto una funzione non troppo reattiva, le prestazioni rilevate dalle simulazioni sono state molto buone ma abbiamo comunque valutato funzioni più veloci nell’adattarsi al cambiamento dell’ambiente circostante. Abbiamo studiato una funzione che fosse molto crescente per piccoli valori di numero di nodi e con un andamento più tranquillo al crescere del numero di dispositivi (curva viola) e una seconda funzione che desse una risposta meno rapida all’inizio, ma che mantenesse una discreta crescita al crescere del numero di nodi (curva verde).