# Capitolo 2 – Stato dell’arte

## Risparmio energetico

* Un discorso generale sul consumo energetico e che da molti anni viene preso in considerazione seriamente il discorso dello studio di come sia possibile risparmiare energia nel settore IT… se possibile anche qualche link a qualche numero di consumi
* Server Farm /Google /FB …
* Anche nel mobile è nata questa esigenza dopo le prime generazioni di smartphone
* L’uso del Mobile Cloud Computing per lo scarico di punto di elaborazione e di storage
* Parlare degli studi sui sistemi che auto-adattivi soggetti a vincoli di QoS
* Sistemi con multipli vincoli di QoS…come fanno il trade off?.

--------------------------------------------------------------------------------------------

### Introduzione

Nel settore dell’IT è ormai da parecchi anni che si guarda al problema dei consumi energetici e alle possibili soluzioni per ridurli. La tendenza verso cui questo settore sta andando è quella di spostare la parte di computazione e di storage sul Web, a carico a grandi aziende che offrono servizi di hosting o clouding. In generale ogni azienda che possiede almeno una *server farm* medio - grande, si dedica allo studio del risparmio energetico, in quando la spesa per l’energia per il ramo IT non è indifferente ai fini del budget aziendale. Sono stati fatti studi con lo scopo di trovare possibili soluzioni in grado di ridurre gli sprechi. Anche se non ce ne rendiamo conto, nei sistemi IT vi sono molti sprechi di energia. [*Receipe for Efficiency: Principles of Power-Aware Computing*] illustra che una stima del costo energetico di un Data Center di Google sia nell’ordine dei milioni di dollari per anno e che il consumo energetico del settore IT a livello mondiale sia stato stimato attorno ai 40 miliardi di dollari nel 2009. Questo fa capire come con l’evolvere della tecnologia, essa diventi sempre più potente e performante, ma necessiti anche di una quantità maggiore di energia. Per questo motivo i discorsi di consumo energetico, della riduzione degli sprechi e dell’ottimizzazione nell’uso delle risorse sono divenuti sempre più importanti e oggetto di ricerca.

### Sistemi Self-Adaptative

In letteratura si trovano molti studi riguardanti possibili soluzioni per una migliore gestione delle risorse e minimizzazione dello spreco di energia. Questi tipi di sistemi si chiamano *Self-Adaptative*, poiché riescono ad auto regolare parametri interni con l’obiettivo di consumare il minimo quantitativo di energia necessario. In applicazioni reali, bisogna però tenere conto non solo dei vincoli energetici, ma anche della qualità del servizio. E’ logico pensare che un’azienda con un data center offra servizi sul web, quindi queste soluzioni self-adaptative devono trovare un compromesso tra consumo di energia e qualità del servizio (QoS) da garantire. Questo trade-off è il motivo per cui, al giorno d’oggi, non si sono trovate soluzioni definitive. In letteratura vi sono molti articoli riguardanti sistemi auto adattativi con vincoli di QoS, come il [*Accurate Modeling and Efficinet QoS Analysis of Adaptive Systems under Bursty Workload*], [*Dynami power management for QoS-aware applications*], [*On the relationships between WoS and softare adaptability at the architectural level*] e [*QoS and energy management with Petri nets: A self-adaptive framework*]. L’idea di base è di poter gestire la quantità di risorse per offrire un determinato servizio, in modo da allocarne il minor numero possibile garantendo allo stesso tempo una qualità del servizio accettabile senza che l’utente ne percepisca un degrado. Questo tipo di adattamento è ovviamente molto dinamico e molto dipendente dalla distribuzione delle richieste che vengono fatte per quel particolare servizio. Ad esempio, nel [*QoS and energy management with Petri nets: A self-adaptive framework*] è discussa la situazione di un server mail aziendale ed è presentato su grafico l’utilizzo del server mail nel tempo, nell’arco di una settimana. Dal grafico risulta che tra un giorno e il successivo, quindi di notte, vi è un uso limitato, mentre nella parte diurna dei giorni lavorativi si hanno alti livelli di utilizzo. Nel week-end invece un uso trascurabile. In questo caso tramite un monitoraggio, si può creare un sistema self-adaptive in grado di reagire in anticipo alle variazioni delle richieste in arrivo al server mail, allocando o deallocando risorse preventivamente. Più l’attività che si cerca di studiare non ha una distribuzione di richieste ben definita o è soggetta ad alta variabilità e magari a imprevedibili momenti di burst di richieste, più sarà difficile creare un sistema auto adattativo in grado di reagire preventivamente nel modo corretto. In questo caso lo studio del trade-off è più complicato e bisogna affidarsi a più euristiche. In generale, il concetto è di avere un sistema in grado di allocare solo le risorse necessarie a gestire l’attuale carico di lavoro col minor spreco di energia e garantendo una certa QoS. Il sistema deve anche poter anticipare un cambiamento nella frequenza di richieste in ingresso al sistema, allocando o deallocando risorse, sempre con l’obiettivo di avere attive solo le risorse necessarie a gestire il carico di lavoro.

Altri esempi di più recente attualità riguardano il mondo mobile. Dall’avvento dei primi smartphone, gli utenti si sono resi subito conto che uno dei problemi di questi dispositivi è l’eccessivo consumo energetico e la conseguente poca autonomia rispetto ai cellulari della precedente generazione. Infatti, le case produttrici implementano su tutti i loro dispositivi mobili moduli di risparmio energia, in grado di agire sul sistema in base all’impostazione scelta. Anche se le impostazioni variano da brand a brand, in generale si può scegliere tra una modalità normale, senza risparmio energetico, una modalità di semi risparmio energetico e una o due modalità di super risparmio energetico in cui viene disabilitato praticamente ogni tipo di servizio non essenziale, lasciando al dispositivo le sue funzioni core e niente altro. Per quanto riguarda i dispositivi mobili ci si sta muovendo verso il Mobile Cloud Computing, come discusso in [*A Survey of Mobile Cloud Computing*]. Data la potenza di elaborazione con cui sono equipaggiati i dispositivi mobili oggi giorno, essi sono a tutti gli effetti dei computer in miniatura e quindi possono fare tutto; a prova del sempre crescente numero di applicazioni create e immesse sul mercato. Tutto ciò si traduce in un consumo energetico per sostenere l’esecuzione di tutte queste applicazioni e in un problema d’immagazzinamento dati. Tramite il *Mobile Cloud Computing*, i dispositivi mobili posso scaricare il peso dell’elaborazione di certi dati e la memorizzazione dei file direttamente nel Cloud, con il risultato di risparmiare fino al 25% di batteria e molto spazio di memorizzazione.

Un altro ambito in cui è molto importante lo studio e l’ottimizzazione del consumo energetico è quello dell’hardware. Ad esempio nei microprocessori dei computer, ma non solo, sono presenti moduli in grado di monitorare lo stato del chip, la frequenza di lavoro e altri parametri. Nello specifico il modulo *Dynamic frequency/voltage scaling* (DVS) permette di regolare il voltaggio e quindi la frequenza di lavoro del processore in maniera dinamica, senza dover spegnere il componente. *Nell’articolo* [*Dynami power management for QoS-aware applications*] sono presentate le specifiche dell’*Advanced Configuration and Power Interface* (ACPI) e proposte come open standard per la gestione della configurazione e dell’energia di sistemi singoli o insieme di sistemi.

## Bluetooth 4.0 Low Energy

### Introduzione

-intro

-telefoni

-sensoristica

--------------------------------------------------------------------------------------------

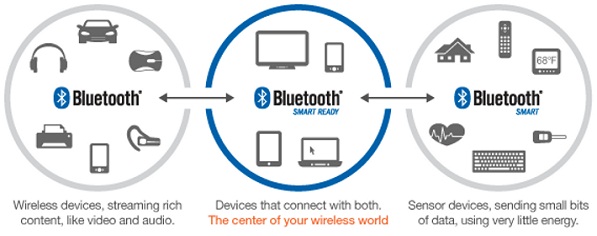
La tecnologia Bluetooth 4.0 Low Energy, nome in codice Seattle, è una tecnologia wireless ed è stata rilasciata a metà 2010, col rilascio delle le specifiche relative allo standard; ora alla versione 4.2 [https://www.bluetooth*.org/en-us*/specification/adopted-specifications]. Dato un sempre maggior impiego dei dispositivi Bluetooth in svariati ambiti come l’IT, moduli di rilevamento e sensori, healtcare [https://www.bluetooth.org/en-us/bluetooth-brand/smart-marks] e altri ancora, questa tecnologia è stata progettata per risolvere uno dei punti deboli delle versioni precedenti: l’alto consumo energetico richiesto. Sulla tecnologia Bluetooth Low Energy, l’azienda lancia la gamma di prodotti denominati Bluetooth Smart identificando una serie di mercati in cui è richiesta una tecnologia a basso consumo energetico come le nascenti strutture delle smart home, healtcare, sport & fitness. Questa tecnologia è stata progettata e migliorata per tutti questi dispostivi che utilizzano la batteria “a bottone” infatti, alcuni vantaggi presentati dall’azienda sono:

* Basso consumo energetico,
* Possibile autonomia di mesi o anni per sistemi alimentati da batterie a bottone,
* Ampia compatibilità con molti smartphone, computer e tablet.

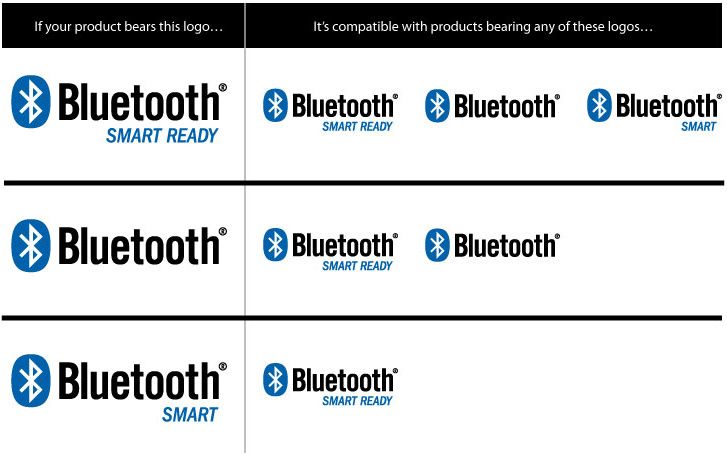
Sempre nel 2010 si venne a creare un problema di retro compatibilità dei dispositivi Bluetooth Smart con tutti i dispositivi equipaggiati con tecnologie Bluetooth Classic (il Bluetooth Classic è la vecchia tecnologia dalla versione 1 fino a prima della versione 4). E’ risultato che la tecnologia Smart era completamente non compatibile con i dispositivi Bluetooth Classic. Per questo motivo nel 2011 l’azienda cambiò il logo del marchio Bluetooth Smart [http://www.bluetooth.com/Pages/Bluetooth-Brand.aspx], dividendolo in:

* Bluetooth Smart,
* Bluetooth Smart Ready.

L’azienda creò una versione parallela alla Smart, chiamata Smart Ready che implementa l’architettura del Bluetooth 4.0, quindi in grado di comunicare con il Bluetooth sia Smart sia Smart Ready, ma che è anche in grado di comunicare con il Bluetooth Classic, inserendo quell’elemento di retro compatibilità che mancava. Tutti i dispositivi come smartphone, computer e tablet utilizzano tecnologia Bluetooth Smart Ready, mentre dispositivi progettati per funzionare con batterie a bottone o disponibilità energetiche limitate sono equipaggiate solamente con la tecnologia Bluetooth Smart. In figura 2.1 è rappresentato lo schema di compatibilità, mentre in figura 2.2 è riportata la tabella di compatibilità pubblicata dall’azienda.



*Figura 2.1: Schema di compatibilità.*



*Figura 2.2: Tabella compatibilità tecnologie.*

Tutti gli smartphone, dal 2010 fino ad oggi sono stati equipaggiati con la tecnologia Low Energy Smart Ready. Si è, infatti, costatato un forte incremento nel mercato, di dispositivi che sfruttano tale tecnologia, come i bracciali con tecnologia fitbit, la serie degli i-Watch e gli smartwatch di Android che rendono l’orologio una vera e propria estensione del telefono. Tutti questi dispositivi sfruttano la tecnologia Bluetooth 4.0 Low Energy rendendo gestibile l’uso di così tante periferiche Bluetooth senza avere drammatiche conseguenze sulla durata delle batterie. Ora l’azienda Bluetooth sta già lavorando a una nuova versione del Bluetooth Low Energy, che è la versione 4.1. Essa mira a ridurre alcuni conflitti di banda con la tecnologia LTE. Infatti, come riportato da [*https://it.wikipedia.org/wiki/Bluetooth*], la 4.0 si trova tra le bande 40 e 41 del LTE. La versione 4.1 effettua un controllo di banda prima di iniziare le trasmissioni per evitare il conflitto. Uno dei primi smatphone a essere equipaggiato con la nuova versione 4.1 del BLE è stato il Google Nexus 6.

### Caratteristiche generali

-consumo energia ridotto rispetto al classic

-fsa

-pacchetti dati ?

--------------------------------------------------------------------------------------------

Nella sezione seguente presentiamo, in via generale, alcune caratteristiche del Bluetooth 4.0 Low Energy, anche con qualche comparazione col predecessore Bluetooth Classic per valutare i miglioramenti e i vantaggi che questa nuova tecnologia porta.

#### Consumo energia

La tecnologia Bluetooth Low Energy presenta un consumo ridotto di energia rispetto alla versione precedente, il Bluetooth Classic. Per il Bluetooth Classic vi sono tre classi distinte in cui sono categorizzati i dispositivi, con le relative distanze raggiunte e i consumi associati, mentre per il Bluetooth Low Energy, l’azienda Bluetooth non ha specificato alcun valore massimo di distanza o consumi e quindi il tutto è lasciato come libera scelta ai produttori dei trasmettitori. Questo si traduce ovviamente in una stima per il calcolo del consumo medio di questa nuova tecnologia. Nella tabella 2.1 sono riportati i valori di consumi energetici e distanze, relativi al Bluetooth Classic [http://www.bluetooth.com/Pages/Basics.aspx] e [*http://it.wikipedia.org/wiki/Bluetooth*], mentre nella tabella 2.2 sono riportati i valori per il Bluetooth Low Energy riportati nelle specifiche ufficiali [*https://www.bluetooth.org/en-us/specification/adopted-specifications*] e anche in [*http://tesi.cab.unipd.it/44150/1/tesi\_tibertoa.pdf*]. Come si può vedere il consumo di energia del Low Energy è ridotto di almeno un ordine di grandezza rispetto al precedente standard Classic.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Potenza (mW)** | **Potenza (dBm)** | **Distanza (m)** |
| **Classe 1** | 100 | 20 | ~100 |
| **Classe 2** | 2.5 | 4 | ~10 |
| **Classe 3** | 1 | 0 | ~1 |

*Tabella 2.1: Caratteristiche Bluetooth Classic.*

|  |  |  |
| --- | --- | --- |
| **Potenza massima all’output** | **Potenza minima all’output** | **Distanza (m)** |
| 10 mW (10dBm) | 0.01 mW (-20dBm) | ~50 |

*Tabella 2.2: Caratteristiche Bluetooth Low Energy.*

Come discusso nel paper [sensors-12-11734 ], il consumo molto basso da parte del Low Energy, permette a dispositivi alimentati con batterie a bottone, un ciclo di vita che varia tra i 2 giorni e i 14 anni. Inoltre rispetto allo standard Classic, che consente un massimo di 7 dispositivi slave per ogni master, la tecnologia Low Energy offre più flessibilità rendendo questo valore dipendente dall’applicazione e può variare tra 2 e 5.917 slave per master. Per quanto riguarda la distanza di trasmissione del Bluetooth Low Energy, è stato rilevato essere in media attorno ai 50 m.

#### Link Layer – stati

Come descritto nelle specifiche ufficiali [*https://www.bluetooth.org/en-us/specification/adopted-specifications*], l’operatività del Link Layer può essere descritta dai seguenti stati:

* Stato di Standby
* Stato di Advertising
* Stato di Scanning
* Stato di Initiating
* Stato di Connection

Questi rappresentano i nodi dell’automa a stati finiti che modella il comportamento del Link Layer. Ogni macchina a stati può essere in un solo stato alla volta, ma un Link Layer può avere più istanze di macchina a stati, se il suo hardware lo consente. È necessario però che almeno una delle sue macchine sia in grado di entrare nello stato di Advertising o nello stato di Scanning. Nel caso di macchine a stati multiple, vi sono restrizioni sulle combinazioni possibili di stati attivi tra tutte le macchine a stati; per ulteriori dettagli si faccia riferimento alla documentazione ufficiale [*https://www.bluetooth.org/en-us/specification/adopted-specifications*].

Lo stato di Standby è uno stato in cui il dispositivo non trasmette e non riceve alcun pacchetto. Questo stato è raggiungibile da qualsiasi altro stato.

Lo stato di Advertising, trasmetterà pacchetti di advertising e, se possibile, ascolterà e risponderà a quei pacchetti trasmessi in risposta al suo pacchetto di advertising. Dispositivi nello stato di Advertising sono chiamati advertiser e lo stato di Advertising è raggiungibile dallo stato di Standby.

Lo stato di Scanning è uno stato di osservazione. Un dispositivo nello stato di Scanning rimane in ascolto per i pacchetti di advertising. Un dispositivo nello stato di Scanning è chiamato scanner e lo stato di Scanning è raggiungibile solo dallo stato di Standby.

Un Link Layer nello stato di Initiating rimarrà in ascolto per pacchetti di advertising trasmessi da determinati dispositivi e risponderà a determinati pacchetti se ha l’intenzione di aprire una connessione con quel dispositivo mittente. Un dispositivo in stato di Initiating è chiamato initiator. Lo stato di Initiating è raggiungibile dallo stato di Standby.

Lo stato di Connection può essere raggiunto sia dallo stato di Advertising sia dallo stato di Initiating. Lo stato di Connection si divide a sua volta in due stati:

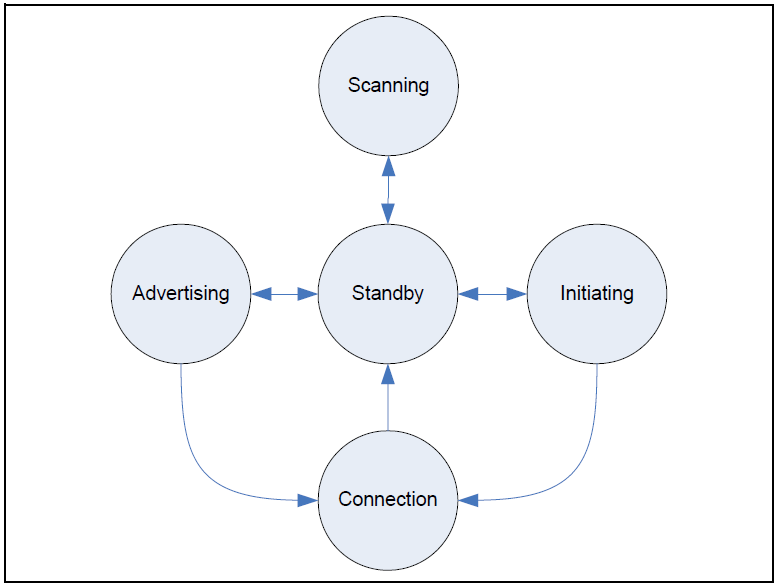
* Master
* Slave

Se raggiunto dallo stato di Advertising si entrerà nello stato di Connection Slave, mentre se si raggiunge lo stato di Connection dallo stato di Initiating si entra nello stato Connection Master.

Un singolo dispositivo col ruolo di Master comunicherà con un dispositivo col ruolo di Slave, scegliendo le impostazioni e i valori dei parametri di connessione.

Un singolo dispositivo col ruolo di Slave può comunicare con un solo Master alla volta.

In figura 2.3 è riportato lo schema della macchina a stati del Link Layer.



*Figura 2.3: Link Layer Macchina a Stati.*

## Reti Peer2Peer

### Introduzione

La nostra ricerca si è orientata su modelli di reti in grado di rappresentare reti Peer-to-Peer, chiamate anche Peer2Peer o P2P. Una rete P2P è una rete in cui non vi è una struttura gerarchica e ogni nodo è considerato allo stesso livello di tutti gli altri. Senza la presenza di nodi più importanti o nodi rappresentati centri di conoscenza della rete, nessun nodo può avere una visione completa della rete stessa e non può sapere se la porzione di rete da lui vista rappresenta l’intera rete. Ogni nodo quindi, ha una visione parziale e locale dell’overlay della rete. Le topologie di rete sono rappresentate da grafi bidirezionali, poiché i canali di comunicazione tra i nodi di una rete P2P per le situazioni analizzate, sono bidirezionali. Il modello di rete P2P è un’architettura logica di una rete di nodi paritari, senza alcuna struttura Client-Server fissi [https://it.wikipedia.org/wiki/Peer-to-peer]; ogni nodo è paritario a tutti gli altri, infatti, ogni nodo è chiamato *peer*. La struttura Client-Server è creata solo nel momento di dover instaurare una connessione tra due nodi, ma più che Client-Server, sarebbe più corretto definirla come Mittente-Destinatario perché non vi sono compiti o azioni predefinite e pre-allocate nelle due parti. Può essere quindi definita una struttura particolare della struttura generica Client-Server.

La principale applicazione di questo modello di rete è stata ed è tuttora quella della condivisione dei file (*file sharing)*, per la quale sono nati tanti sistemi quali Gnutella, FastTrak, Napster, eMule, la rete Torrent, Freenet etc.

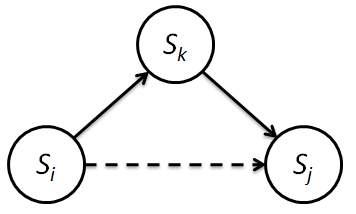
Alcune caratteristiche delle reti Peer-to-Peer:

* *Basso costo d’implementazione*: non è richiesto l’uso di potenti macchine Server, ma è sufficiente che ogni peer possa sostenere le transazioni dell’utente locale e degli altri peer che vogliono connettersi a lui.
* *Amministrazione decentralizzata*: non vi è un server centralizzato di stoccaggio delle informazioni, ma le informazioni sono in possesso dei singoli utenti, localmente, che poi mettono a disposizione della rete.
* *Maggiore velocità di trasmissione*: non avendo un Server centralizzato cui tutti i Client si devono connettere per avere un’informazione, causando un calo nella velocità di trasferimento, in un’architettura P2P l’informazione può essere reperita anche da più nodi contemporaneamente, infatti, è possibile reperire parti diverse della stessa informazione da nodi diversi e alla fine riassemblare il tutto, col grosso vantaggio di poter avere l’informazione in tempi brevi.
* *Sicurezza*: senza la presenza di Server centralizzati, ogni nodo deve garantire per se e per i contenuti che distribuisce, inoltre è anche esposto a ciò che riceve dalla rete che non è controllato da nessuna terza parte.

### Modelli di rete

Poiché abbiamo preso in considerazione l’uso dei grafi, introduciamo due parametri riguardanti i grafi che ne descrivono alcuni aspetti. Essi sono la *edge dependency* e la *degree variance*.

* ***Edge dependency:*** anche chiamata interdipendenza tra archi. Dato un grafo come in figura 2.1, la edge dependency è definita come la probabilità subordinata che esista un arco tra gli stati *Si* ~ *Sj*, sapendo che esiste un arco tra gli stati Si ~ *Sk* e tra *Sk* ~ *Sj*. Formalmente, questa probabilità si esprime così e si può considerare alta se è maggiore della sua probabilità semplice .
* ***Degree variance:*** con degree variance s’indente la varianza del grado dei nodi del grafo.



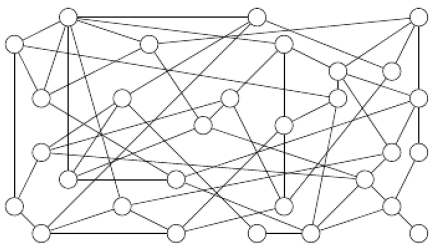
*Figura 2.4: Esempio di edge dependency.*

I tre principali modelli di reti presentati in [*Ruijing Hu, Julien Sopena, Luciana Arantes, Pierre Sens,Isabelle Demeure - A fair comparison of gossip algorithms over large-scale random topologies, Universit\_e Pierre et Marie Curie, CNRS/INRIA, Paris, France Institut Telecom, Telecom ParisTech, CNRS, Paris, France 2012*] per la costruzione di reti P2P casuali, sono:

* Bernoulli Graph,
* Random Geometric Graph,
* Scale-free Graph

### Bernoulli Graph

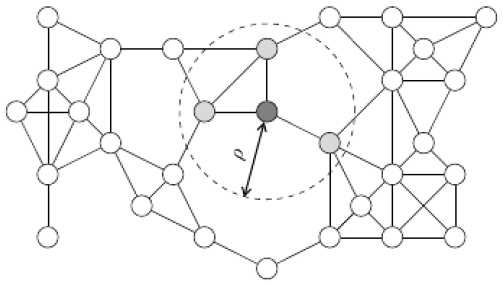
Un Bernoulli Graph *β*(*N,pN)* è un grafo bidirezionale con N nodi nel quale le connessioni hanno una probabilità di esistere pari a *pN*. Per ogni coppia di nodi vi è una probabilità *pN* che il link tra loro esista, indipendentemente da ogni altra connessione. In figura 2.2 è riportato un esempio. Un Bernoulli Graph presenta bassa edge dependency e bassa degree variance.



*Figura 2.5: Esempio di Bernoulli Graph.*

### Random Geometric Graph

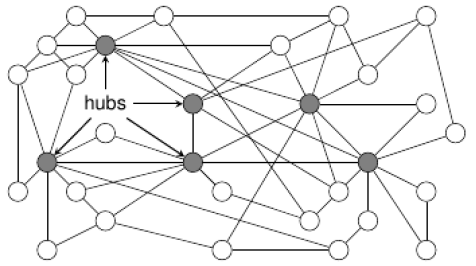
Un Random Geometric Graph *G*(*N,ρ)* è un grafo bidirezionale casuale inserito in un’area limitata. Il grafo è generato disponendo in maniera casuale e uniforme gli N nodi all’interno dell’area. Poi due nodi sono connessi se essi si trovano a una distanza geometriche pari o inferiori a *ρ*. In figura 2.3 è riportato un esempio. E’ immediato notare come reti di questo tipo siano estremamente adatte alla rappresentazione di reti wireless, caratterizzate dalla distanza fisica tra i nodi e da un valore soglia ρ entro il quale è possibile eseguire trasmissioni. Un Random Geometric Graph presenta un’alta edge dependency, dovuta all’importanza della vicinanza fisica tra i nodi, e una bassa degree variance.



*Figura 2.6: Esempio di Random Geometric Graph.*

### Scale-free Graph

Un Scale-free Graph *S*(*N,m)* è un grafo bidirezionale con una bassa edge dependency e un’alta degree variance, dovuta alle sue connessioni distribuite secondo legge potenza. Reti di questo tipo sono generate partendo con un set di nodi m0, poi a ogni ciclo si aggiunge un nodo e si collegano i suoi *m* archi ad altri nodi già presenti nel grafo. La probabilità che un nuovo nodo sia collegato a un nodo già esistente è proporzionale al grado di quest’ultimo. Collegamenti di questo tipo si dicono preferenziali e fanno si che si creino pochi nodi con un grado molto alto, che vendono chiamati *hub* con un grado medio di circa *2 m*, e molti nodi con un grado basso chiamati *periferici* con grado medio compreso tra *m* e *2 m*. In figura 2.4 è riportato un esempio.



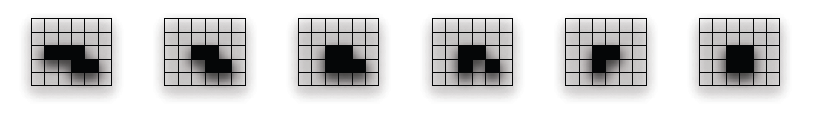
*Figura 2.7: Esempio di Scale-free Graph.*

## Algoritmi di Gossip

I protocolli Epidemici, o di Gossip, sono paradigmi computazionali e di comunicazione orientati a sistemi distribuiti su larga scala caratterizzati da alta dinamicità. Il metodo più semplice per la diffusione d’informazioni in una rete è effettuare un broadcast cercando di propagare al maggior numero di nodi l’informazione, ma con lo svantaggio di saturare tutti i canali di comunicazione. Per questo motivo i protocolli di gossip utilizzano un approccio probabilistico per la gestione della diffusione dell’informazione attraverso la rete, con lo scopo di massimizzare la diffusione dell’informazione sovraccaricando il meno possibile i canali di comunicazione. In un protocollo epidemico, un nodo che ha un’informazione sceglierà in modo casuale un nodo vicino con cui comunicare e tentare di condividere l’informazione.

Uno dei protocolli epidemici più famoso è “Game of life” [[Martin Gardner](https://it.wikipedia.org/wiki/Martin_Gardner), [The fantastic combinations of John Conway's new solitaire game "Life"](http://www.ibiblio.org/lifepatterns/october1970.html), [Scientific American](https://it.wikipedia.org/wiki/Scientific_American" \o "Scientific American), 223, ottobre 1970, pp. 120-123], un [automa cellulare](https://it.wikipedia.org/wiki/Automa_cellulare) sviluppato dal matematico dal [matematico](https://it.wikipedia.org/wiki/Matematico) [inglese](https://it.wikipedia.org/wiki/Inghilterra) [John Conway](https://it.wikipedia.org/wiki/John_Conway) sul finire degli [anni sessanta](https://it.wikipedia.org/wiki/Anni_1960). L’ambiente per quest’algoritmo è un insieme di celle, dove ogni cella ha 8 celle vicine come mostrato in figura 2.5. Lo scopo dell’algoritmo è di diffondere il seme della vita, quindi ogni cella può essere occupata o no in base alle seguenti regole:

* Nascita: se una cella non occupata ha tre celle vicine occupate, diventa anch’essa occupata.
* Morte: se una cella occupata ha 0 o 1 cella vicina occupata, muore di solitudine, oppure se ha dalle 4 alle 8 celle vicine occupate, muore di sovraffollamento.
* Sopravvissuta: se una cella occupata ha 2 o 3 celle vicine occupate, essa sopravvive alla generazione successiva.



*Figura 2.8: Esempio di Game of life.*

### Classificazione degli algoritmi epidemici

Come discusso in [http://www2.cs.uni-paderborn.de/cs/ag-madh/WWW/Teaching/2004SS/AlgInternet/Submissions/09-Epidemic-Algorithms.pdf] in ogni algoritmo epidemico vi è quella che è chiamata popolazione, un insieme di unità interattive comunicanti. Queste unità usano un set di regole che specificano le modalità di diffusione di una specifica informazione ad altre unità. La scelta di queste regole è molto legata alla scelta dell’algoritmo e al comportamento che quest’ultimo deve avere. L’importante è che a ogni istante di tempo t, ogni unità sia in uno dei seguenti tre stati, riguardo a una specifica informazione:

* Suscettible (suscettibile): l’unità non conosce nulla dell’informazione in questione ma è disposta a venirne a conoscenza.
* Infected (contagiata): l’unità è a completa conoscenza dell’informazione in questione e utilizza il set di regole per diffondere a sua volta l’informazione.
* Removed (Rimossa): l’unità è a completa conoscenza dell’informazione in questione ma non la diffonde.

Basandoci sui tre stati appena elencati, possiamo definire diverse classi di algoritmi, in cui è indicato per ogni classe come in generale sono trattate le informazioni.

Le principali classi sono:

* Suscettible – Infected (SI)
* Suscettible – Infected – Suscettible (SIS)
* Suscettible – Infected – Removed (SIR)

Esistono anche altre classi che estendono ulteriormente queste classi, aggiungendo alcuni stati intermedi.

#### Suscettible – Infected (SI)

In questo modello si ha che i nodi possono essere suscettibili a un’informazione e quando ne vengono a conoscenza, diventano contagiati e vi rimangono fintanto che tutta la popolazione non diventa contagiata. Ciò però richiede ulteriori controlli esterni per decidere quando terminare la diffusione dell’informazione.

#### Suscettible – Infected – Suscettible (SIS)

A differenza del modello SI, nel modello SIS un’unità contagiata può decidere di fermare la diffusione dell’informazione prima che tutta la popolazione sia contagiata. Ogni unità rimossa può tornare a essere contagiata se riceve nuovamente l’informazione che aveva smesso di trasmettere e ricominciare a trasmetterla di nuovo finché non perde nuovamente interesse nel farlo.

#### Suscettible – Infected – Removed (SIR)

Questo modello è molto simile al modello SIS, con la differenza che un’unità rimossa rimane rimossa per sempre per quella determinata informazione e non potrà più esser contagiata da quell’informazione. Ciò non impedisce che tale unità possa diventare poi suscettibile a nuove informazioni.

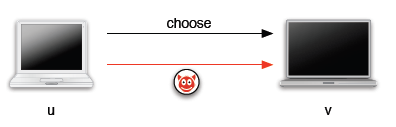
### Metodi di diffusione

In generale, ogni algoritmo di gossip prevede che a ogni iterazione dell’algoritmo ogni unità della popolazione se deve comunicare con un nodo, esso sia scelto in maniera casuale tra i nodi dell’intera popolazione. Poi ogni algoritmo specifica, con vincoli più stringenti, regole diverse di selezione del destinatario. In generale però vi sono tre metodi con cui le unità di una popolazione possono diffondere le informazioni ad altre unità:

* Push
* Pull
* Push&Pull

#### Push

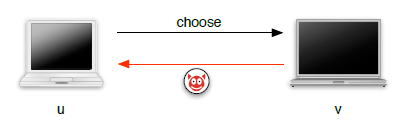
L’algoritmo Push prevede che i nodi contagiati prendano l’iniziativa di diffondere l’informazione, quindi a ogni istante t, il nodo contagiato sceglie un nodo casuale e prova a contagiarlo, come mostrato in figura 2.6. Questa strategia è molto efficace all’inizio della diffusione, quando vi è un alto numero di unità suscettibili e poche contagiate o rimosse, quindi la probabilità che ogni nodo contagiato ha di scegliere un nodo suscettibile è alta. Questa probabilità decresce col passare del tempo perché la quantità di nodi suscettibili diminuisce e il numero di nodi contagiati o rimossi aumenta, rendendo questo metodo poco affidabile nel lungo periodo. Questo metodo non garantisce che tutta la popolazione sia contagiata.



*Figura 2.9: Algoritmo di diffusione Push.*

#### Pull

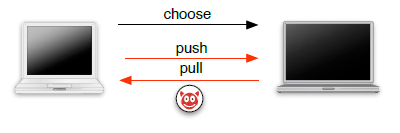
L’algoritmo Pull prevede che un nodo contagiato non si muova attivamente nel diffondere l’informazione, ma che siano i nodi suscettibili a fare richiesta di nuove informazioni ai nodi contagiati. A ogni istante t, un nodo suscettibile seleziona casualmente un altro nodo e gli chiede se ha nuove informazioni. Se il nodo contattato ne ha, allora restituisce l’informazione; in figura 2.7 è riportato un esempio. Quest’algoritmo di propagazione è poco efficace all’inizio dell’epidemia perché vi è solo un nodo contagiato e la probabilità di scegliere proprio lui è uno sulla grandezza della popolazione. Col passare del tempo, più l’informazione si diffonde, più alta sarà la probabilità di selezionare un nodo contagiato. Questo metodo non garantisce che il processo di diffusione abbia inizio poiché vi è una probabilità che nessun nodo suscettibile contatti il nodo contagiato, ma vi è anche la possibilità che tutti i nodi scelgano il nodo contagiato generando così un rapido inizio di epidemia. Questo metodo però garantisce che, se l’epidemia inizia, essa si diffonderà a tutta la popolazione.



*Figura 2.10: Algoritmo di diffusione Pull.*

#### Push&Pull

I metodi Push e Pull presentano vantaggi e svantaggi in differenti momenti del processo di diffusione, l’unione dei due ha lo scopo di unirne i vantaggi dei due metodi. In questo caso un nodo contagiato utilizzerà una strategia Push, mentre i nodi suscettibili utilizzeranno una strategia Pull finché non saranno contagiati. In figura 2.8 è rappresentato un esempio di strategia Push&Pull.



*Figura 2.11: Esempio di diffusione Push&Pull.*

### Applicazioni

Tra gli svariati campi in cui trovano utilizzo gli algoritmi di gossip, come presentato in [http://www2.cs.uni-paderborn.de/cs/ag-madh/WWW/Teaching/2004SS/AlgInternet/Submissions/09-Epidemic-Algorithms.pdf], vi è quello del corretto Replicated Database Maintenance, quello dell’Epidemic Routing e il Mantenimento della visione del Network Overlay. In merito al Replicated Database Maintenance il primo algoritmo studiato fu il Direct Mail, una soluzione che prevede che il nodo in possesso della nuova informazione debba contattare direttamente ogni altro nodo per diffondere l’aggiornamento. Questo metodo è posso affidabile perché è molto sensibile allo stato della rete su un unico nodo e sull’unico mittente. Per ovviare a tali problemi sono stati introdotti due principali algoritmi di gossip: Anti-Entropy e il Rumor Mongering. L’Anti-Entropy è un algoritmo di tipo SI, che cerca come dice il nome di contrastare la crescita dell’entropia in un sistema di replicazione dati, facendo si che a ogni ciclo dell’algoritmo, a coppie, i nodi sincronizzino i loro dati. Unico problema è che per la sincronizzazione di database, a volte è necessario trasmettere sulla rete l’intero database. Il Rumor Mongering invece è un algoritmo di tipo SIR che ha lo scopo di propagare tra i nodi solo l’elenco degli aggiornamenti e mai l’intero database. A ogni ciclo ogni nodo che ha la lista aggiornamento non vuota, sceglie un nodo casuale e cerca di propagargli i suoi aggiornamenti. Con questo tipo di algoritmo nasce il problema di trovare un criterio col quale decidere quando smettere di propagare gli aggiornamenti.

Da questo sono nati diverse versioni dell’algoritmo di Rumor Mongering, in base ai diversi criteri usati per fermare la propagazione degli aggiornamenti. E’ quindi possibile suddividere la classe degli algoritmi di Rumor Mongering nelle seguenti sotto classi:

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

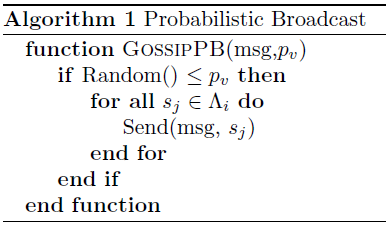
### Algoritmi di gossip per il P2P

Nel contesto degli algoritmi di gossip relativi a reti Peer2Peer, descriveremo quelli più utilizzati per la diffusione di messaggi. Essi sono il Probabilistc Broadcast (PB), Probabilistic Edge (PE) e Fixed Fanout (FF). Definiti i seguenti termini:

* *Λi* : insieme di nodi adiacente all’i-esimo nodo.
* *Vi*: la cardinalità di Λi.
* *msg*: il messaggio da diffondere.
* *pv*,*pe*,*fanout*: sono i valori probabilistici a ciascun algoritmo, rispettivamente PB,PE e FF.

### Probabilistic Broadcast

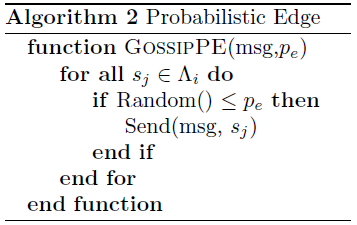
Il Probabilistic Broadcast, prevedere che chi ha l’informazione o la ricevuta, la diffonda con una certa probabilità *pv* ai nodi vicini. In figura 2.9 è riportato lo pseudo codice che descrive l’algoritmo.



*Figura 2.12: Pseudo codice Probabilistic Broadcast.*

### Probabilistic Edge

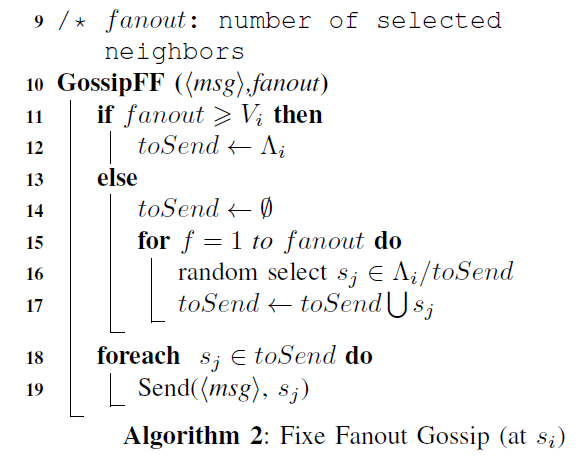
Il Probabilistic Broadcast, prevedere che chi ha l’informazione o l’ha ricevuta, la diffonda su ogni arco uscente con una certa probabilità *pe* per ogni arco. In figura 2.10 è riportato lo pseudo codice che descrive l’algoritmo.



*Figura 2.13: Pseudo codice Probabilistic Edge.*

### Fixed Fanout

Il Fixed Fanout, prevedere che chi ha l’informazione o l’ha ricevuta, la diffonda a *fanout* nodi adiacenti selezionati a caso tra tutti i nodi vicini. In figura 2.11 è riportato lo pseudo codice che descrive l’algoritmo.

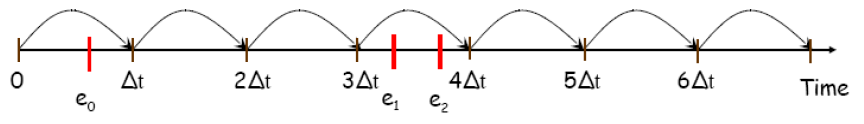


*Figura 2.14: Pseudo codice Fixed Fanout.*

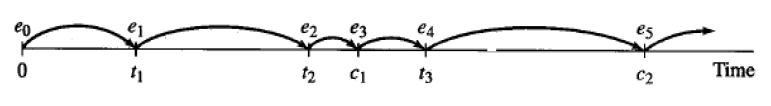
## Simulatori

La nostra ricerca si è focalizzata su simulatori di reti Peer-2-Peer, reti in cui tutti i nodi della rete sono allo stesso livello, sono paritetici e non vi è presenza di gerarchie tra nodi, e su simulatori di protocolli basati sia su cicli sia su eventi. In generale simulare qualcosa prevede l’esecuzione di una sequenza di operazioni nel tempo e, dove le entità che eseguono tali operazioni sono molteplici. Per poter gestire l’esecuzione contemporanea e/o concorrente di più entità il simulatore deve stabilire l’ordine in cui eseguire quale istruzione di quale nodo. La scelta sul come venga stabilito l’ordine di esecuzione caratterizza il motore del simulatore che potrà essere basato su cicli, su eventi o su entrambi. Una simulazione basata su cicli prevede che si definisca uno “time step” di simulazione, che è l’incremento del tempo virtuale di simulazione che definisce la cadenza di ogni ciclo. Tutte le operazioni definite come periodiche verranno eseguite ad ogni ciclo, contemporaneamente. Inoltre qualsiasi evento non periodico che occorre tra un ciclo e un altro verrà schedulato al ciclo successivo. Come si vede in figura 2.12, gli eventi e0, e1 ed e2 accadono tra un ciclo e il successivo, con un passo di simulazione di Δt. L’evento e0 verrà quindi processato all’instante Δt, mentre gli eventi e1 ed e2 verranno processati all’instante 4Δt. In questo modo si perde la distribuzione degli eventi, avendo lo scheduler che appiattisce la distribuzione temporale su un unico ciclo, ad ogni iterazione. Questo tipo di simulazioni sono molto vincolanti e non permettono una rappresentazione realistica di una generica classe di nodi. Si adatta molto bene a simulazioni di operazioni batch e/o periodiche. Per poter avere più precisione e dettaglio nella modellazione dell’esecuzione degli eventi vi è lo scheduling guidato dagli eventi che, come dice il nome, ordinerà le operazioni da eseguire in base al momento in cui esse hanno generato o genereranno un evento.

In figura 2.13 è riportato un esempio di scheduling guidato dagli eventi. Il tempo di simulazione non sarà più una progressione costante, ma seguirà il susseguirsi degli eventi, nell’ordine in cui essi avvengono. In questo caso la simulazione avrà inizio con l’evento e0, poi il simulatore salterà all’instante di tempo t1 in quanto lo scheduler prima di t1 non ha nessuna operazione che deve essere eseguita.



*Figura 2.15: Scheduling basato su cicli.*



*Figura 2.16: Scheduling guidato dagli eventi.*

All’istante t1 avviene l’evento e1 e viene processato, poi si passerà all’istante di tempo t2 perché l’evento e2 è previsto che avvenga in quell’istante e così via tutti gli altri. In generale gli eventi sono relativi per esempio alla ricezione di un messaggio da parte di un nodo. In figura 2.13 vediamo che vi sono gli eventi e3 ed e5 che sono schedulati agli istanti di tempo c1 e c2. Questo è un esempio di integrazione di eventi ciclici in un ambiente guidato dagli eventi. Gli eventi e3 ed e5 sono azioni periodiche e quindi devono essere eseguite ogni “time step” specificato, ma come fare se non vi è un “time step” di simulazione? Il simulatore genererà un evento ogni “time step” di tempo di simulazione associando come evento l’azione periodica. Ciò garantisce di poter avere con un unico motore di scheduling, sia eventi aperiodici che azioni periodiche. In questo caso abbiamo un simulatore basato sia sugli eventi sia sui cicli.

In letteratura esistono tanti simulatori di reti come Mosquite[[1]](#footnote-1) che è una libreria per CSIM, PADS[[2]](#footnote-2) sviluppato dal Dipartimento di Ingegneria Informatica dell’Università di Bologna in grado di simulare protocolli complessi su reti di larga scala o senza una ben definita struttura. Vi è anche PeerSim[[3]](#footnote-3) una libreria Java che fornisce gli strumenti di simulazione basata sia su cicli sia su eventi. Abbiamo poi concentato la nostra attenzione su OMNeT++[[4]](#footnote-4), un framework costruito sulla piattaforma Eclipse[[5]](#footnote-5) ma scritto in C++. OMNeT++ è uno simulatore basato solo su eventi ma dato il suo intenso sviluppo in ambito commerciale, vi sono disponibili diverse librerie e tool in grado di offrire molte funzionalità di simulazione di protocolli di rete e diverse tipologie di reti. Nel seguito presenteremo brevemente le caratteristiche di PeerSime e OMNeT++.

### PeerSim

PeerSim è un software sviluppato in ambiente accademico che consente di simulare reti con una alta flessibilità sul numero di nodi che compongono la rete, rendendolo quindi adatto a simulare reti di grandi dimensioni. PeerSim poi necessita che venga definito lo stack di protocolli che ogni nodo della rete implementerà; infatti la logica con cui questo simulatore costruisce la rete da simulare è quella di istanziare solo un nodo e poi esso viene copiato, o clonato, tante volte fino a raggiungere il numero di nodi nella rete. Per questo motivo si nota subito che PeerSim è uno strumento ottimizzato per la simulazione di reti anche di grandi dimensioni e soprattutto, come si può dedurre dal nome, reti soprattutto Peer-to-Peer. Non presenta alcun tipo di strumento grafico per la rappresentazione della rete o dei pacchetti in transito tra i nodi, la simulazione è solo computazione e stampa messaggi su console. Vi è la possibilità di utilizzare o estendere alcune funzioni di alcuni componenti del sistema di simulazione che permettono di stampare su file di testo le coordinate dei nodi della rete e/o i rami del grafo della rete nel formato usato da GNUplot[[6]](#footnote-6) e quindi poi visualizzare il layout di rete. PeerSim necessita che l’utente definisca in un apposito file di configurazione le specifiche di simulazione, della rete, dei protocolli implementati sui nodi, delle dipendenze tra i protocolli. Richiede inoltre di specificare i componenti di inizializzazione, i componenti di controllo e i componenti di osservazione. I componenti di inizializzazione sono componenti che il simulatore eseguirà solo all’inizio del processo di simulazione e hanno il compito di inizializzare i nodi della rete e la rete stessa, ad esempio per stabilire le connessioni tra i nodi. I componenti di controllo sono “agenti” che vengono eseguiti ciclicamente e hanno il compito di agire sulla rete durante la simulazione, per introdurre dinamicità nella stessa come per esempio accendere o spegnere nodi o canali di trasmissione per simulare malfunzionamenti oppure aggiungere o togliere nodi e collegamenti. Questi componenti di controllo sono opzionali. Infine i componenti di osservazione detti anche osservatori sono componenti che operano alla fine della simulazione oppure ciclicamente se resi cycle-based impostando il loro time step. Il loro compito è quello appunto di osservare la rete e permettere di raccogliere dati per analisi. I dati da raccogliere devono essere specificati dall’utente nell’apposita funzione di osservazione che viene eseguita dal motore di simulazione. Un esempio di osservatore è il componente che permette di scrivere su file il layout della rete in formato compatibile con GNUplot. Punto forte di questo simulatore quindi sono la flessibilità sul numero di nodi della rete, le ottime prestazioni di simulazione anche per grandi reti e la scelta di lasciare all’utente l’estensione dei componenti che governano la simulazione.

### OMNeT++

OMNeT++ è un framework e libreria di simulazione basata sul linguaggio C++, estendibile e modulare. OMNeT++ è un software di simulazione molto diffuso sia nel settore commerciale sia nel settore scientifico per la simulazione di reti e protocolli di trasmissione. OMNeT++ è un software che offre un editor di sviluppo basato su Eclipse Sono disponibili una gran varietà di strutture base come reti wired e wireless ed è possibile aggiungere estensioni che permettono di ampliare la gamma di reti supportate. INET è una delle estensioni più corpose del framework e contiene una grossa quantità di reti e protocolli delle più diffuse strutture di rete utilizzate.

OMNeT++ è un simulatore basato solo su eventi, ma come spiegato nella Sezione 2.5, è possibile coprire anche situazioni di esecuzione di azioni periodiche, organizzandole come eventi programmati. OMNeT++ contiene anche un ambiente grafico, Tkenv, per la rappresentazione grafica della struttura della rete e degli eventuali link fisici tra i nodi; è in grado inoltre di eseguire un ri-ordinamento spaziale dei nodi della rete con l’obiettivo di rappresentare il grafico della rete col minor numero di intersezioni tra i collegamenti per rendere più leggibile lo schema ed eventuali animazioni date dalla simulazione.

OMNeT++ offre una architettura per i componenti che compongono la rete, detti moduli. I moduli sono anch’essi implementati col linguaggio C++ e successivamente assemblati in componenti più grandi, utilizzando un linguaggio a più alto livello, il linguaggio Network Definition (NED). Grazie a questa sua struttura modulare, è possibile riutilizzare facilmente i moduli, sia semplici sia complessi, già presenti nelle librerie di sistema oppure quelli creati dall’utente.

Con OMNeT++, come per PeerSim, è possibile specificare in appositi file di inizializzazione una lista di parametri necessari a configurare la rete durante la sua fase di inizializzazione prima di iniziare la simulazione vera e propria. In un singolo file di inizializzazione è possibile specificare più scenari di esecuzione e per ognuno specificare quindi valori diversi.

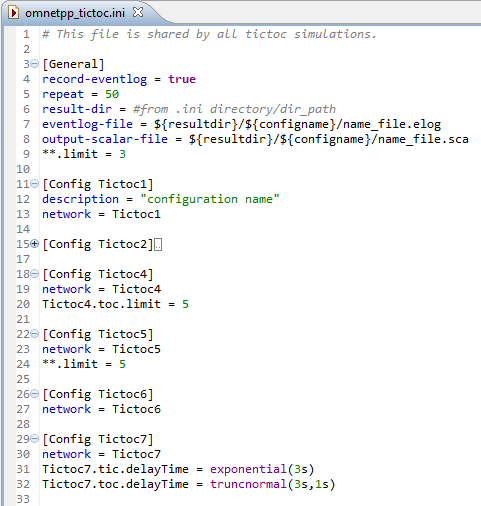
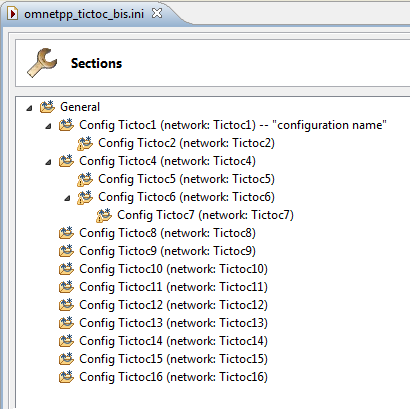
I parametri specificati possono essere letti dal file di Network Definition e usati per creare e impostare la rete secondo i parametri specificati dall’utente nel file di inizializzazione. Tramite il file di Network Definition si può propagare la lettura dei parametri fino ai file C++ che implementano i metodi che modellano il comportamento del componente.

OMNeT++ offre anche un sistema di raccolta dati, tramite il quale si possono poi analizzare i dati raccolti, filtrarli e ottenere differenti rappresentazioni grafiche dei dati stessi. La raccolta dati può essere sia la sequenza temporale dei risultati, sia una raccolta delle statistiche su dati specifici oppure entrambe le cose.

#### File di Inizializzazione

Un file di inizializzazione è un file necessario alla corretta configurazione della rete, dei suoi componenti e dell’ambiente di simulazione. Un file di inizializzazione ha estensione “.ini” e per comodità verrà anche chiamato “INI file”. In un file di inizializzazione è possibile fornire parametri organizzati in una struttura ad albero, in cui ogni ramo è uno scenario di simulazione indipendente dagli altri ma eredita tutti i parametri coi rispettivi valori, se non vengono ridefiniti, dello scenario padre. Al momento della simulazione vera e propria, il sistema chiederà all’utente di scegliere quale scenario di simulazione utilizzare se non ve ne è uno specificato direttamente nello script che lancia l’esecuzione di simulazione.

Nella figura2.14 sono riportati due esempi di file d’inizializzazione. Vediamo nella figura 2.14 a) come sotto la sezione “General” vengano specificati tutti i parametri comuni a tutte le configurazioni e di seguito vengono specificate tutte le altre configurazioni coi parametri specifici per quelle simulazioni. Mentre nella figura 2.14 b) è riportata una possibile struttura ad albero delle configurazioni.

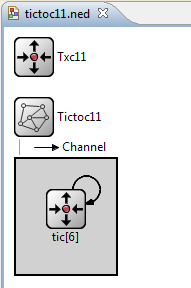
a) b)

*Figura 2.17: Esempio file di inizializzazione.*

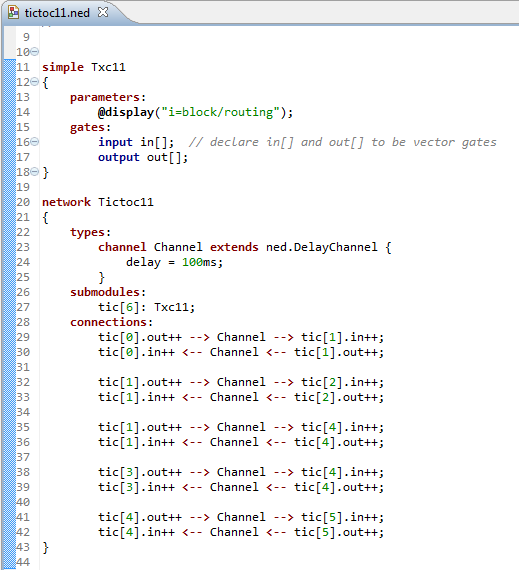
In generale ogni file di inizializzazione caratterizza un set di scenari da simulare, quindi si tende a includere nella sezione “General” anche tutti i parametri che servono a impostare correttamente la simulazione o la gestione del framework quali, per esempio l’unità di misura con la quale viene scandito il tempo di simulazione, il numero di ripetizioni per ogni caso di studio che si desidera fare, le varie directory in cui si vogliono salvare i risultati o i dati raccolti.

#### Linguaggio NED

OMNeT++ offre, tramite un linguaggio ad alto livello chiamato Network Definition, la possibilità di definire il layout della rete e la struttura dei singoli componenti e della composizione dei componenti più grandi. Il linguaggio di Network Definition viene usato per creare file di specifica con estensione “.ned”, che abbrevieremo con NED file. Il linguaggio NED è stato pensato per poter implementare singoli moduli in maniera indipendente dall’ambiente, rendendo così facile il loro riutilizzo o estensione. Il sistema ci offre due visibilità del layout di rete ad alto livello: Design e Source. Con la vista Design viene fornita una rappresentazione grafica della rete, se pur sempre ad alto livello, mentre la vista Source permette di vedere e implementare il file come puro codice sorgente. In figura 2.15 è riportato un esempio di NED file in vista Design, dove il sistema dice che nel file in questione sono stati usati componenti di tipo “Txc11”, i canali di comunicazioni utilizzati sono dei “Channel” e la rete si può riassumere come un vettore di nodi collegati tra di essi tramite diversi “Channel”. La vera rappresentazione del layout di rete viene fatta solo a runtime in fase di simulazione nel caso venga utilizzato lo strumento grafico Tkenv. In figura 2.16 invece è riportato lo stesso file della figura 2.15 ma in vista Source. Come si può vedere nella figura 2.16, in vista sorgente possiamo vedere tutto il codice corrispondente alla vista Design. Si nota che nella prima parte viene definito il singolo componente “Txc11”, poi viene definita la rete come un vettore di componenti “Txc11” e tra essi vi sono dei collegamenti di tipo “Channel”, che è definito dentro la rete.



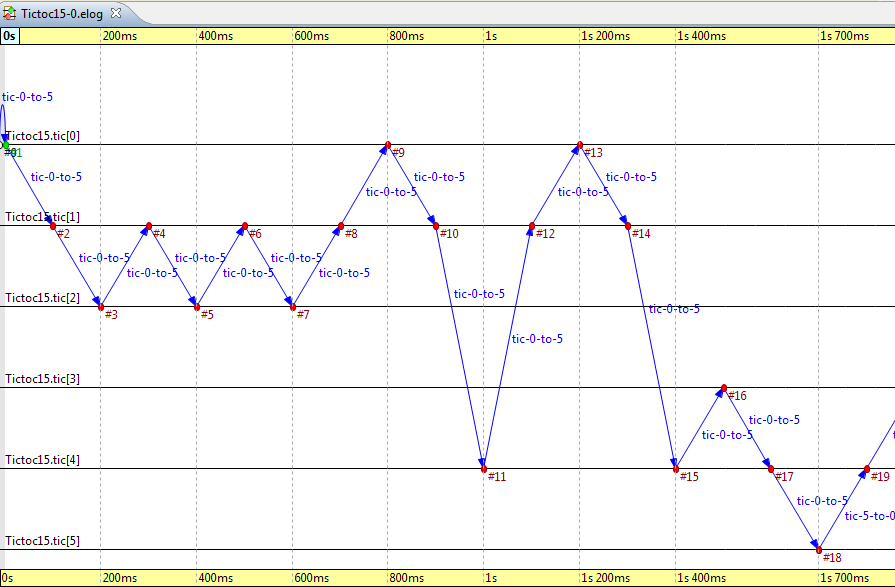
*Figura 2.18: Esempio di NED file, vista Design.*



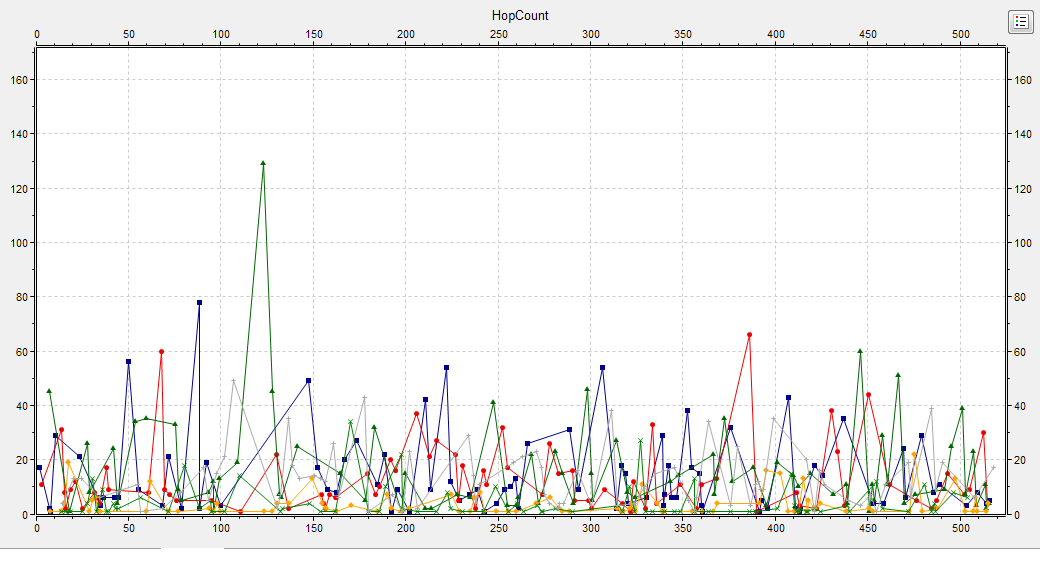
*Figura 2.19: Esemprio di NED file, vista Source.*

#### Raccolta Dati

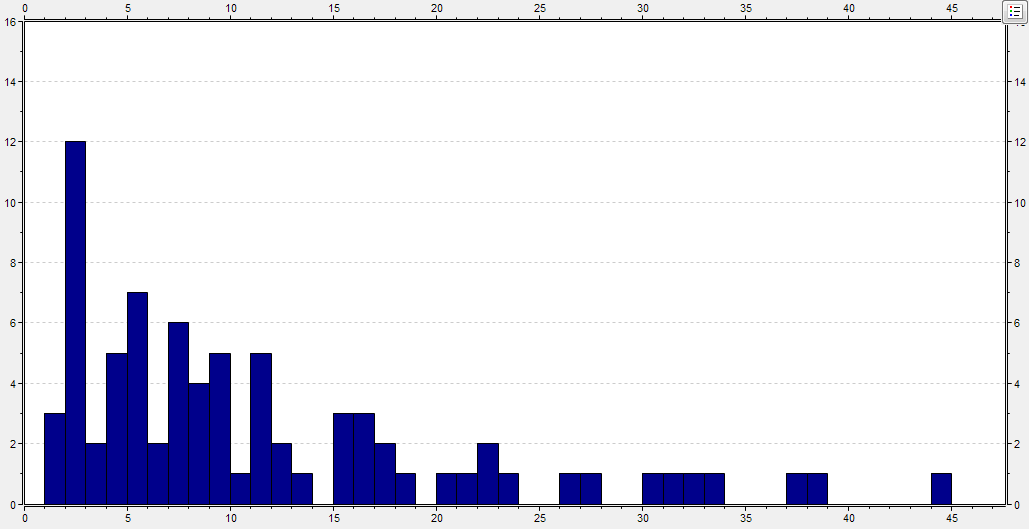
OMNeT++ fornisce metodi per la raccolta di dati e statistiche e la possibilità di analizzarli al termine della simulazione. Si possono raccogliere due categorie di dati: i log temporali, chiamati anche event-log e i dati relativi a statistiche. I log temporali non sono altro che rappresentazioni grafiche della successione degli eventi tra i nodi della rete. Molto utile per analizzare la sequenza temporale degli eventi, la loro durata e le loro interazioni, soprattutto quando si hanno parecchi eventi che avvengono in contemporanea; in figura 2.17 è riportato uno frammento di un esempio di event-log. Per quanto riguarda le statistiche, si possono raccogliere dati con riferimento temporale, chiamati vettori, oppure statistiche su dati, chiamati scalari, quali media, varianza, deviazione standard, somma, minimo, massimo ecc… . Sia sui vettori che sugli scalari è possibile poi fare delle operazioni di manipolazione dati, come eseguire raggruppamenti, applicare filtri, applicare operazioni ai dati o a gruppi di dati come l’operatore media; ad esempio si può applicare l’operatore media a un grafico temporale di vettori, per ottenere l’andamento temporale della media, utile per visualizzare il comportamento asintotico del sistema. In figura 2.18 è riportato un esempio di rappresentazione grafica di vettori temporali. In figura 2.19 invece è riportato un esempio di istogramma come rappresentazione della distribuzione del conteggio di hop necessari per raggiungere l’host destinatario.



*Figura 2.20: Esempio di event-log.*



*Figura 2.21: Esempio di grafico di vettori temporali.*



*Figura 2.22: Esemprio di grafico di statistiche.*

## Acronimi

|  |  |  |
| --- | --- | --- |
| Bluetooth 4.0 Low Energy | BLE | Lo standard Bluetooth 4.0 Low Energy |
| Bluetooth | BT | La tecnologia Bluetooth |
| Probabilistic Broadcast | PB | Algoritmo di gossip per reti P2P |
| Probabilistic Edge | PE | Algoritmo di gossip per reti P2P |
| Fixed Fanout | FF | Algoritmo di gossip per reti P2P |
| fixed fanout | ff | Parametro dell’algoritmo di Fixed Fanout |
| Algoritmo Dynamic Fanout | DFA | Algoritmo di propagazione dinamica e soluzione proposta |
| Dynamic Fanout | DF | Parametro dell’algoritmo di Dyanmic Fanout |
| Advertising Limit | AL | Parametro dell’algoritmo di Dynamic Fanout |
|  |  |  |
|  |  |  |

1. <http://www.mesquite.com/> [↑](#footnote-ref-1)
2. <http://pads.cs.unibo.it/doku.php?id=start/> [↑](#footnote-ref-2)
3. <http://peersim.sourceforge.net/> [↑](#footnote-ref-3)
4. <http://omnetpp.org/> [↑](#footnote-ref-4)
5. <https://eclipse.org/> [↑](#footnote-ref-5)
6. <http://www.gnuplot.info/> [↑](#footnote-ref-6)