# Capitolo 2 – Stato dell’arte

## Risparmio energetico

## Bluetooth 4.0 Low Energy

## 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. Quindi ogni nodo ha una visione parziale e locale dell’overlay della rete. Le topologie di rete vengono rappresentate da grafi bidirezionali, in quanto i canali di comunicazione tra i nodi di una rete P2P per le situazioni analizzate, sono bidirezionali. Il modello di rete P2P è una 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 viene chiamato *peer*. La struttura Client-Server viene creata solo nel momento di dover instaurare una connessione tra due nodi, ma più che Client-Server, sarebbe più corretto definirla come Mittente-Destinatario in quanto 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 tutt’ora quella della condivisione dei file, in inglese *file sharing*, per la quale sono nati tanti sistemi quali Gnutella, FastTrak, Napster, eMule, la rete Torrent, Freenet etc.

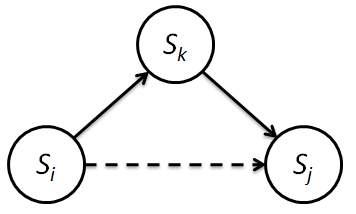
Alcune caratteristiche delle reti Peer-to-Peer:

* *Basso costo di 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 a cui tutti i Client si devono connettere per avere un informazione, causando un calo nella velocità di trasferimento, in una 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

Visto che abbiamo preso in considerazione l’uso dei grafi, introduciamo due parametri relativi ai 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 viene 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 si indente la varianza del grado dei nodi del grafo.



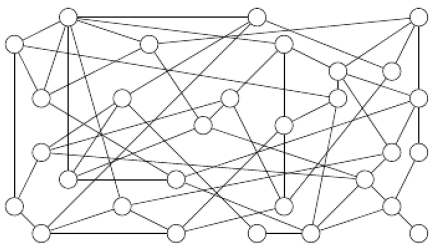
*Figura 2.1: Esempio di edge dependency.*

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

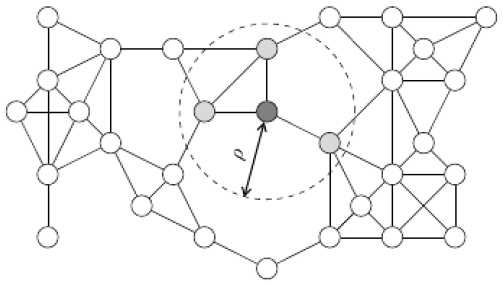
Un Bernoulli Graph *β*(*N,pN)* è un grafo bidirezionale con N nodi nel quale le connessioni hanno una probabilità di esistere pari a *pN*. Per ogni coppia di nodi vi è una probabilità *pN* che il link tra loro esista, indipendentemente da ogni altra connessione. In figura 2.2 è riportato un esempio.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



*Figura 2.2: Esempio di Bernoulli Graph.*

### Random Geometric Graph

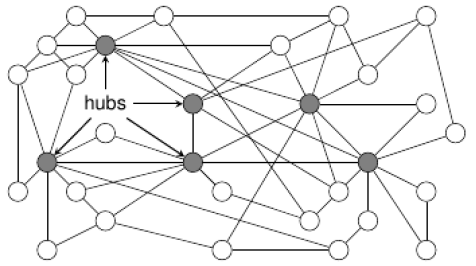
Un Random Geometric Graph *G*(*N,ρ)* è un grafo bidirezionale casuale inserito in un’area limitata. Il grafo viene generato posizionando in maniera casuale e uniforme gli N nodi all’interno dell’area. Poi due nodi sono connessi se essi si trovano ad una distanza geometrica pari o inferiore a *ρ*. In 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 effettuare trasmissioni. Un Random Geometric Graph presenta un’alta edge dependency, dovuta dall’importanza della vicinanza fisica tra i nodi, e una bassa degree variance.



*Figura 2.3: Esemprio 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 alla sue connessioni distribuite secondo legge potenza. Reti di questo tipo vengono generati partendo con un set di nodi m0, poi ad 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 venga collegato ad 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 *2m*, e molti nodi con un grado basso chiamati *periferici* con grado medio compreso tra *m* e *2m*.. In figura 2.4 è riportato un esempio.



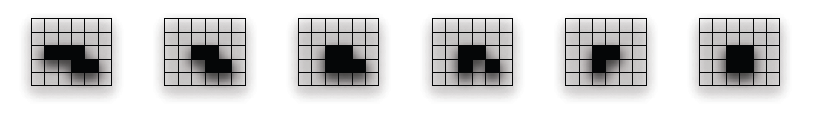
*Figura 2.4: Esempio di Scale-free Graph.*

## 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 di 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 un nodo vicino causale a cui con cui comunicare e tentare di condividere l’informazione.

Uno dei protocolli epidemici più famoso è “Game of life” [https://it.wikipedia.org/wiki/Gioco\_della\_vita], 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 questo algoritmo è quello di 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 meno 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 celle vicine occupate, 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 2.5: Esempio di Game of life.*

### Classificazione degli algoritmi epidemici

Come detto 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 della diffusione di una specifica informazione che potrebbe essere utile ad altre unità. La scelta di queste regole è fortemente legata alla scelta dell’algoritmo e al comportamento che quest’ultimo deve avere. L’importante è che ad ogni istante di tempo t, ogni unità sia in uno dei seguenti tre stati, relativamente a una specifica informazione:

* Suscettible (suscettibile): l’unità non conosce nulla dell’informazione in questione ma è disposta a venirne a conoscenza.
* Infected (infettata): 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 viene indicato per ogni classe come in generale viene trattate le informazioni.

Le principali classi sono:

* Suscettibile – Infetta (SI)
* Suscettibile – Infetta – Suscettibile (SIS)
* Suscettibile – Infetta – Rimossa (SIR)

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

#### Suscettibile – Infetta (SI)

In questo modello si ha che i nodi possano essere suscettibili a un informazione e quando ne vengono a conoscenza diventano infetti e vi rimangono fintanto che tutta la popolazione non diventa infetta. Ciò però necessita di ulteriori controlli esterni per decidere quanto terminare la diffusione dell’informazione.

#### Suscettibile – Infetta - Suscettibile (SIS)

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

#### Suscettibile – Infetta – Rimossa (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 infettata da quell’informazione. Ciò non impedisce che tale unità possa diventare poi suscettibile a nuove informazioni.

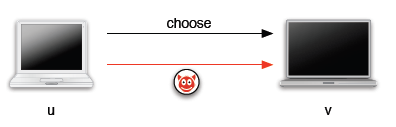
### Metodi di diffusione

In generale, ogni algoritmo di gossip prevede che ad ogni iterazione o istante t, in base se si usa un modello discreto o continuo, ogni unità della popolazione se deve comunicare con un nodo, esso venga scelto in maniera casuale tra i nodi della popolazione. Poi ogni algoritmo specifica regole diverse di selezione del destinatario o vincoli più stringenti. 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

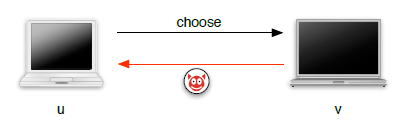
L’algoritmo Push prevede che i nodi infettai prendano l’iniziativa di diffondere l’informazione, quindi ad ogni ad ogni istante t, il nodo infetto sceglie un nodo casuale e prova ad infettarlo, come mostrato in figura 2.6. Questa strategia è molto efficace all’inizio della diffusione, quando vi è un alto numero di unità suscettibili e poche infette o rimosse, quindi la probabilità che ogni nodo infetto 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 infetti o rimossi aumenta, rendendo questo metodo poco affidabile nel lungo periodo. Questo metodo non garantisce che tutta la popolazione venga infettata.



*Figura 2.6: Algoritmo di diffusione Push.*

#### Pull

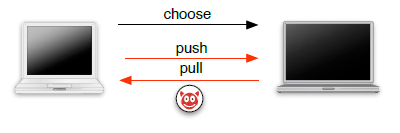
L’algoritmo Pull prevede che un nodo infetto non si muova attivamente nel diffondere l’informazione, ma che siano i nodi suscettibili a fare richiesta di nuove informazioni ai nodi infetti. Ad ogni istante t, un nodo suscettibile seleziona casualmente un altro nodo e gli chiede se ha informazioni nuove. Se il nodo contattato ne ha, allora restituisce l’informazione; in figura 2.7 è riportato un esempio. Questo algoritmo di propagazione è poco efficace all’inizio dell’epidemia in quanto vi è solo un nodo infetto 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 infetto. Questo metodo non garantisce che il processo di diffusione abbia inizio in quanto vi è una probabilità che nessun nodo suscettibile contatti il nodo infetto, ma vi è anche la possibilità che tutti i nodi scelgano il nodo infetto generando così un rapido inizio di epidemia. Questo metodo però garantisce che, se l’epidemia inizia, essa si diffonderà a tutta la popolazione.



*Figura 2.7: 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 infetto utilizzerà una strategia Push, mentre i nodi suscettibili utilizzeranno una strategia Pull finché non saranno infettati. In figura 2.8 è rappresentato un esempio di strategia Push&Pull.



*Figura 2.8: Esempio di diffusione Push&Pull.*

### Applicazioni

Tra gli svariati campi in cui trovano utilizzo gli algoritmi di gossip, come detto in [http://www2.cs.uni-paderborn.de/cs/ag-madh/WWW/Teaching/2004SS/AlgInternet/Submissions/09-Epidemic-Algorithms.pdf], vi sono quello del corretto Replicated Database Maintenance, il 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 in quanto è fortemente 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 ad 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 delle l’elenco degli aggiornamenti e mai l’intero database. Ad 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 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.

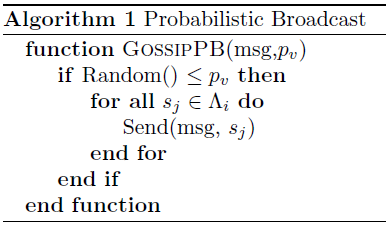
### Algoritmi di gossip per il P2P

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

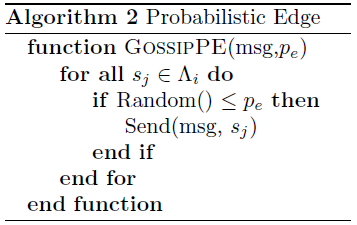
Il Probabilistic Broadcast, prevedere che chi ha l’informazione o la ricevuta, la diffonda con una certa probabilità *pv* hai nodi vicini. In figura 2.9 è riportato lo pseudo codice che descrive l’algoritmo.



*Figura 2.9: Pseudo codice Probabilistic Broadcast.*

### Probabilistic Edge

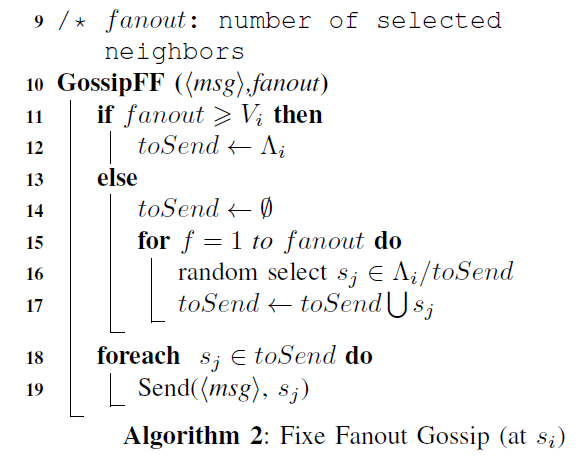
Il Probabilistic Broadcast, prevedere che chi ha l’informazione o l’ha ricevuta, la diffonda su ogni arco uscente con una certa probabilità *pe* per ogni arco. In figura 2.10 è riportato lo pseudo codice che descrive l’algoritmo.



*Figura 2.10: 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.

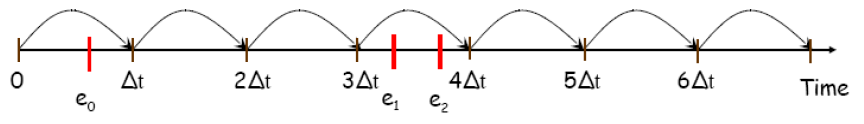


*Figura 2.11: Pseudo codice Fixed Fanout.*

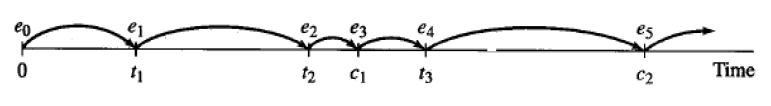
## Simulatori

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 anche su simulatori di protocolli che fossero sia basati su cicli sia su eventi. In generale simulare qualcosa prevede l’esecuzione di una sequenza di operazioni nel tempo e magari 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 simulazione molto realistica di una generica classe di nodi. Si adatta molto bene a simulazioni di operazioni batch e/o periodiche. Per poter avere più profondità nell’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 messo nessuna operazione da eseguire.



*Figura 2.12: Scheduling basato su cicli.*



*Figura 2.13: 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” ti 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.

Navigando sul web, abbiamo trovato tanti simulatori di reti come Mosquite [http://www.mesquite.com/] che è una libreria per CSIM, PADS [http://pads.cs.unibo.it/doku.php?id=start] 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 [http://peersim.sourceforge.net/], una libreria Java che fornisce gli strumenti di simulazione basata sia su cicli sia su eventi. Abbiamo infine trovato anche OMNeT++ [http://omnetpp.org/], un framework costruito sulla piattaforma Eclipse 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.

## 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 molto adatto a simulare reti che scalano molto facilmente. 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 venga copiato, o clonato, tante volte quanto 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 P2P. L’output deve essere creato dall’utente, rendendo quindi lo stream in output su console ristretto alle informazioni di simulazione più le informazioni volute dall’utente. 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 di 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 [http://www.gnuplot.info/] 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, 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 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 devono 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 simulare quindi è 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 [https://omnetpp.org/intro/what-is-omnet]. OMNeT++ è un software che offre un editor di sviluppo basato su Eclipse [https://eclipse.org/]. Sono disponibili molte una gran varietà di strutture base come reti wired e wireless, ma è possibile aggiungere estensioni che permettono di ampliare la gamma di reti supportate. INET è una delle estensioni del framework più corpose 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 NED che sta per Network Definition. Grazie a questa sua struttura modulare, è possibile riutilizzare facilmente i moduli, sia semplici sia più 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.

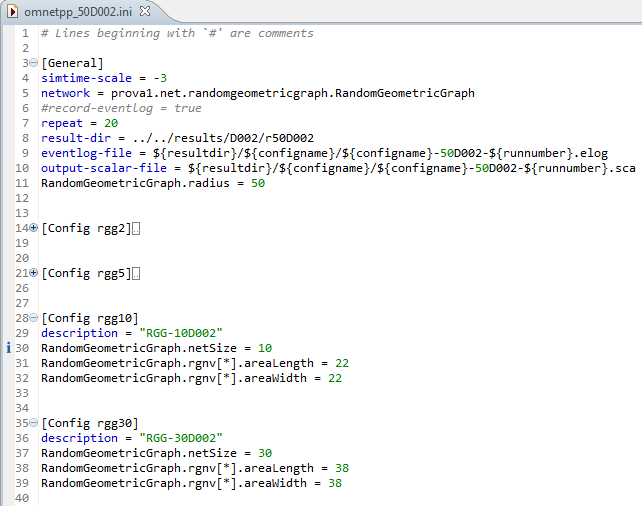
I parametri specificati possono essere letti dal file di Network Definition e usati per creare e impostare la rete secondo i parametri specificati dall’utente nel file di inizializzazione. Tramite il file di Network Definition si può propagare la lettura dei parametri fino ai file C++ che implementano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

OMNeT++ offre anche un sistema di raccolta dati, col 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 fare una definizione dei parametri organizzata 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 ridefiniti, dello scenario padre da cui discente. 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 di lancio.

Nella figura2.14 abbiamo un esempio di file d’inizializzazione. Vediamo come sotto la sezione “General” vengano specificati tutti quei parametri comuni a tutte le configurazioni e di seguito vengono specificate tutte le altre configurazioni coi parametri specifici per quelle simulazioni.

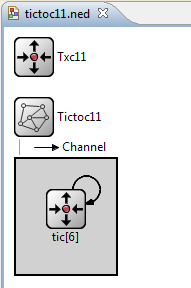


*Figura 2.14: Esempio file di inizializzazione.*

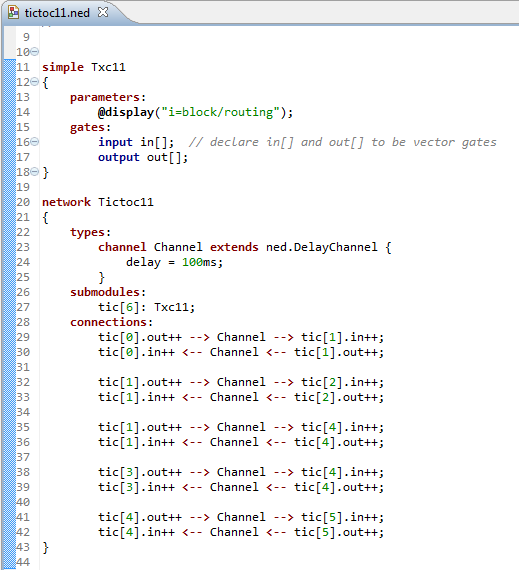
In generale ogni file di inizializzazione caratterizza un set di scenari da simulare, quindi si tende a mettere nella sezione “General” anche tutti quei parametri che servono a impostare correttamente la simulazione o la gestione del framework quali per esempio l’unità di misura con a quale viene scandito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

### 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 ci fornisce 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 fatto 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 che sta dietro al 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”, definito dentro la rete.



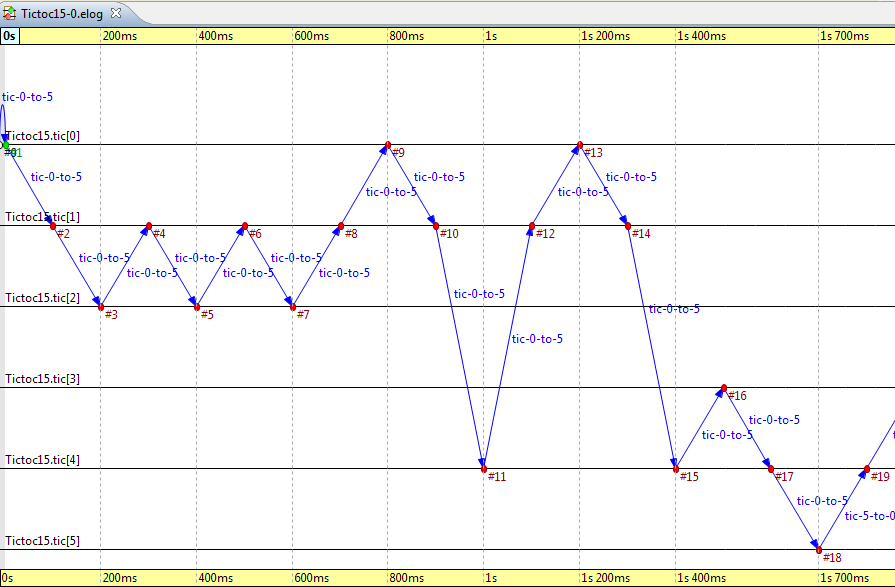
*Figura 2.15: Esempio di NED file, vista Design.*



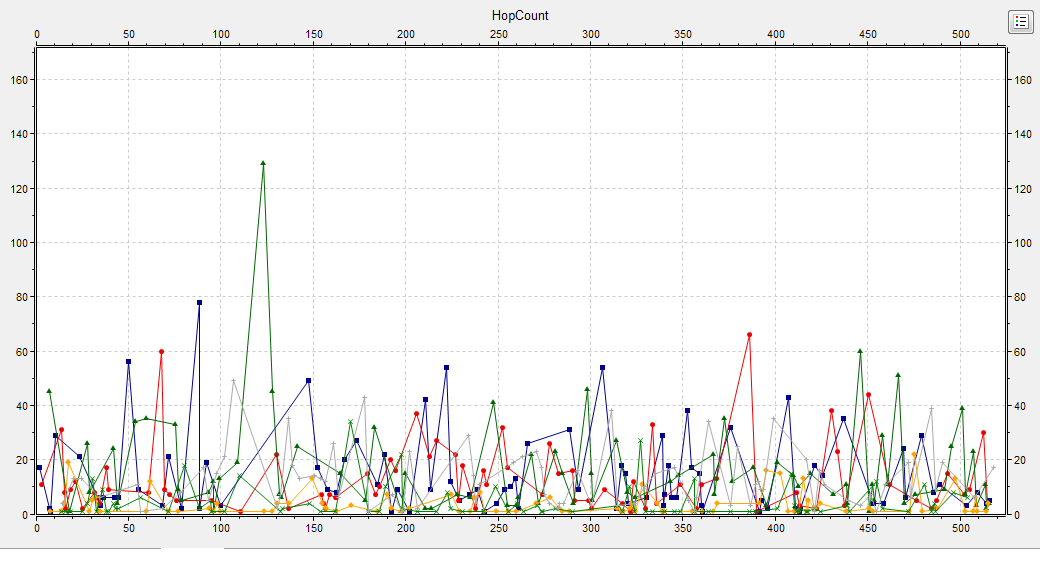
*Figura 2.16: Esemprio di NED file, vista Source.*

### Raccolta Dati

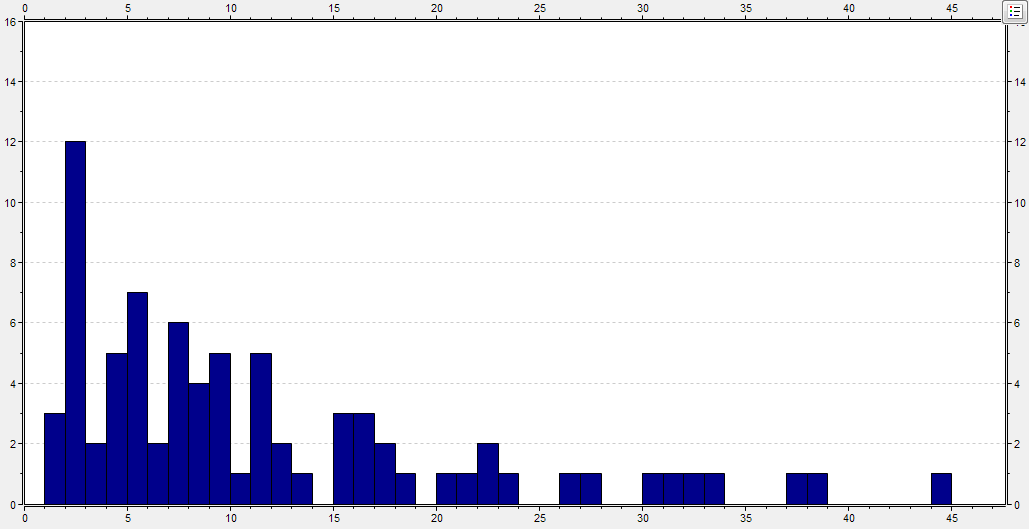
OMNeT++ fornisce metodi per la raccolta dati e di statistiche e la possibilità di analizzarli al termine della simulazione. Si possono raccogliere due categorie di dati: i log temporali, chiamati anche event-log e i dati relativi a statistiche. I log temporali non sono altro che rappresentazioni grafiche della successione degli eventi tra i nodi della rete. Molto utile per analizzare la sequenza temporale degli eventi e la loro durata, 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 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 2.17: Esempio di event-log.*



*Figura 2.18: Esempio di grafico di vettori temporali.*



*Figura 2.19: Esemprio di grafico di statistiche.*