Alma Mater Studiorum - Università di Bologna

**CAMPUS DI CESENA**

**DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA**

Corso di Laurea in Ingegneria e Scienze Informatiche

# STUDIO DEL FREE RIDING NELLA DISSEMINAZIONE PER MEZZO DI GOSSIP

Elaborato in
Programmazione di Reti

Relatore:
GABRIELE D'ANGELO

Presentata da:
FILIPPO SAVINI

Correlatore:
MORENO MARZOLLA

Anno Accademico
2017 - 2018

# Contenuti

# Indice degli Algoritmi

# Introduzione

I sistemi peer-to-peer (P2P) non strutturati costituiscono una buona pratica per realizzare applicazioni distribuite efficienti. Le architetture client-server presentano infatti un problema di costi che le rende spesso inadeguate a situazioni caratterizzate da un massiccio carico di messaggi scambiati ed un numero di clienti altamente dinamico, per le quali invece trova spesso un utilizzo una topologia di network decentralizzata. Le ragioni per questa preferenza sono particolarmente evidenti quando si considera che i peer che compongono tali network sono essi stessi altamente dinamici, nel senso che possono abbandonare o unirsi al network in ogni momento e si appoggiano solitamente ad infrastrutture di rete eterogenee, le quali sono spesso non affidabili limitate in termini di risorse in quanto a comunicazioni, capacità computazionale ed approvvigionamento energetico. [i] In tale caso infatti, l'uso di strategie agili di connessione per creare un *overlay network* (o "rete sovrapposta", cioè la rete composta dai collegamenti instaurati tra i nodi) in aggiunta all'uso di un semplice protocollo di disseminazione che permetta ai nodi di interagire offrono un modo semplice per gestire il sostrato di interazione, al di sopra del quale è possibile implementare applicazioni distribuite. In questo caso, i nodi creano collegamenti in base ad un processo di connessione che non dipende dal "tipo" dei nodi coinvolti. Quindi, per esempio, se ci troviamo ad avere a che fare con un sistema di gestione dei contenuti, i collegamenti tra peer non sono creati in base ai contenuti posseduti da questi ultimi ma seguendo altri criteri (anche arbitrariamente, se lo si desidera). [ii]

Si apre a questo punto la questione della disseminazione dei dati all'interno di tali network, che concerne la consegna di uno o più messaggi a tutti i nodi all'interno della rete. Che i messaggi siano generati da un singolo nodo o da molti nodi, l'importante è che la trasmissione sia caratterizzata da latenza (velocità di risposta) ridotta e quanto meno network overhead possibile (l'obiettivo è l'assenza di messaggi ridondanti). Un'interessante soluzione a questo problema è quella basata sul gossip. Tale strategia di disseminazione epidemica utilizza comunicazioni randomizzate che distribuiscono contenuti senza un piano di dirottamento preciso, né tantomeno basato sui contenuti presenti all'interno dei nodi. Il concetto chiave alla base del sistema è che ogni singolo nodo inoltra i messaggi ricevuti ai suoi vicini, cioè i nodi ai quali è collegato, sulla base di una data soglia di probabilità e di poche regole di base, quale la non-ritrasmissione di un messaggio al nodo dal quale è stato originariamente ricevuto. Un introduzione esaustiva a questo argomento è disponibile in [iii].

Il gossip è stato riconosciuto come un paradigma di comunicazione robusto e scalabile, da impiegare in ambiti distribuiti su vasta scala. [iv] [v] [vi] Difatti, per quanto abbia costi di comunicazione solitamente più alti di altre soluzioni ottimizzate, come i protocolli tree-based, un sistema di disseminazione basato sul gossip è intrinsecamente tollerante ai guasti, quali il *failure* di uno o più peer, e resistente ad attacchi malevoli.

Al giorno d'oggi, l'utilizzo della simulazione parallela e distribuita e l'avvento di processori multi-core permette agli analisti di scendere a notevoli livelli di dettaglio qualora si occupino di studiare il comportamento di entità simulate. È proprio grazie a questi strumenti fornitici dalle moderne tecnologie informatiche che siamo in grado di compiere le dettagliate analisi sul comportamento degli

algoritmi di gossip, senza le quali questa trattazione non avrebbe modo di esistere. I test sono stati tutti condotti tramite l'ambiente di simulazione sviluppato dal Parallel and Distributed Simulation Research Group del Dipartimento di Ingegneria e Scienze Informatiche dell'Università di Bologna, al cui interno è possibile trovare implementati e pronti all'uso diversi algoritmi di gossip, alcuni dei quali di recente formulazione.

L'oggetto di studio principale di questa tesi è un comportamento comune nei sistemi peer-to-peer: il free riding. In un network affetto da tale fenomeno, alcuni nodi beneficiano delle informazioni fornite dai loro peer senza offrire nulla in cambio alla rete. È evidente che gli effetti del free riding sulle performance dei protocolli possono essere importanti. [vii] Nel caso della disseminazione di dati, questo significa che alcuni nodi (cioè i free rider) generano nuovi messaggi perché siano consegnati in giro per la rete ma si rifiutano di inoltrare i messaggi che ricevono, generati da altri nodi. Nel dettaglio, in questa tesi si investigheranno gli effetti del free riding sulla base di determinate metriche (che saranno spiegate in seguito) in due casistiche principali:

1. in uno scenario dove, all'interno dell'ambiente di simulazione, il meccanismo di determinazione dei nodi free rider all'interno del grafo obbedisce ad un input fornito dall'utente;

2. in uno scenario dove i meccanismi adottati usano sempre come punto di partenza un input fornito dall'utente, ma dipendono anche da caratteristiche contingenti, non omogenee su tutto il grafo/rete e per tutta la durata della simulazione.

Fra questi due scenari, al secondo sarà riservata la porzione più vasta di questa trattazione. Il primo scenario è infatti utile soprattutto a fornire un metro di paragone per le performance di questi "meccanismi adattivi" di nuovo sviluppo rispetto al caso base più semplice. Per evitare che la trattazione diventasse troppo lunga e pesante, tutti i principali batch di simulazioni, i cui esiti saranno presentati man mano nei capitoli successivi a questa introduzione, sono stati condotti solo su una specifica configurazione di grafo/rete e utilizzando un singolo algoritmo di disseminazione.

La parte restante di questo documento è organizzata come segue: il capitolo 1 presenta le diverse topologie di grafi impiegate nella realizzazione di overlay network, le metriche per valutare le prestazioni di un protocollo di disseminazione basato sul gossip e tre di questi protocolli; il capitolo 2 descrive l'ambiente di simulazione ed il modo in cui si è scelto di rappresentare i risultati delle simulazioni ed illustra l'incidenza del fenomeno del free riding sul caso di studio a seconda della sua entità; il capitolo 3 analizza un primo meccanismo adattivo dove il processo per determinare se un nodo è o non è un free rider è dipendente dalle caratteristiche di un messaggio che è sul punto di essere inoltrato; il capitolo 4 analizza un secondo meccanismo adattivo dove lo stesso processo è dipendente dalla posizione di un nodo all'interno del grafo; il capitolo 5 prende i due meccanismi sviluppati ed elabora a partire da essi un terzo meccanismo ibrido che rende la determinazione dei free rider un processo dipendente sia dalle caratteristiche del messaggio che dalla posizione del nodo.

# Capitolo 1

# Gossip su Overlay Network

In questo primo capitolo della trattazione verranno fornite le nozioni riguardanti la disseminazione tramite gossip e le tipologie di grafi che servono da base per la realizzazione di network non strutturati. Esse sono di fondamentale importanza per comprendere gli argomenti trattati nei capitoli seguenti.

## 1.1 Topologie di network

In un network non strutturato le connessioni tra nodi sono stabilite arbitrariamente. I nodi gestiscono localmente le loro connessioni con altri nodi e tali collegamenti non dipendono dal processo di disseminazione dei contenuti. [viii] Lo specifico protocollo per la creazione di connessioni tra coppie di nodi permette di realizzare specifiche topologie di grafo e quindi di network. Queste determinate soluzioni sono particolarmente semplici da costruire e da gestire, con bassi costi di manutenzione. Il prezzo da pagare è un'organizzazione dell'overlay non ottimale.

Andremo ora a descrivere le caratteristiche generali di tali topologie, insieme ai modi a disposizione per realizzarle. Va detto però che solo la prima delle topologie ricoprirà un ruolo di primo piano all'interno di questa trattazione.

### 1.1.1 Reti a grafo aleatorio

Un grafo aleatorio corrisponde ad un network generale dove le connessioni tra nodi sono generate secondo un criterio randomico. Il modello Erdös-Rényi (ER) è impiegato per realizzare tali grafi aleatori. [ix] Secondo questo modello, la costruzione di un grafo avviene connettendo nodi in modo casuale. Ogni vertice viene incluso nel grafo con probabilità $p$, indipendentemente da qualunque altro vertice. Pertanto, $p$ determina il livello di connessione di un grafo.

### 1.1.2 Reti a grafo aleatorio

Una rete a invarianza di scala è un grafo che gode della seguente proprietà: se si considera la relazione tra il numero di nodi ed il numero delle connessioni tra di essi, si vede che il suo grafico è di tipo esponenziale negativo e quindi invariante per cambiamenti di scala, cioè aumenti del numero di nodi presenti nella rete. In pratica, paragonando il numero di due tipi di nodi, ad esempio quelli con 5 connessioni e quelli con 10, si vede che la proporzione è $e^{-a(Nb-Na)}$, dove Nb ed Na sono rispettivamente il numero di nodi del denominatore e del numeratore mentre *a* è un parametro del tipo di rete considerato. Questa legge è detta legge di potenza, di cui *a* è il parametro. [x] La conseguenza principale è che la maggioranza dei nodi possiede un numero di "vicini" (i nodi all'altro capo delle connessioni) relativamente basso, mentre una percentuale non trascurabile di nodi, noti in questo caso come "hub", ne possiede un numero sostanzialmente più elevato. [iv] La presenza di questi hub ha un'importante impatto sulla connettività della rete. Si potrebbe dire che la peculiarità di questi network è il loro diametro ridotto, che permette di propagare le informazioni con poche trasmissioni da un nodo all'altro, note in letteratura come "hop".

Per realizzare questo genere di network è impiegato il modello Barabási-Albert (BA). [xi] Il modello BA è basato su un sistema di generazione dei collegamenti preferenziale che mostra l'effetto rich-get-richer. In sostanza, all'arrivo di un nuovo nodo nella rete, esso crea nuove connessioni con altri nodi e in questo caso è più probabile che una connessione vada a collegare il nuovo nodo con nodi già ricchi di vicini.

### 1.1.3 Reti "small world" di Watts-Strogatz

Il modello Watts-Strogatz (WS) è un modello di generazione di grafo aleatorio che produce grafi con proprietà "small-world", ossia di "mondo piccolo". [xii] Esso arriva a questo risultato interpolando tra un grafo aleatorio ER ed un reticolo ad anello regolare. Una iniziale struttura reticolare a *d* dimensioni è utilizzata per generare un modello WS. Ciascun nodo nella rete è inizialmente collegato ai suoi $2d$ vicini più prossimi (per ogni dimensione un nodo ha 2 vicini, cioè il suo predecessore ed il suo successore lungo quella dimensione). Poi un parametro *p* viene utilizzato come probabilità di riconnessione. Ciascun vertice ha una probabilità *p* di essere riconnesso all'interno del grafo come vertice aleatorio.

### 1.1.4 Reti k-regolari

Le reti k-regolari dove tutti i nodi cominciano con lo stesso numero *k* di vicini. Questi network sono assai comuni in diversi sistemi P2P, dove il software in esecuzione sui peer è configurato per avere un dato numero di connessioni nell'overlay. Si tende ad agire in questo modo per questioni legate al load balancing. [xiii]

## 1.2 Fondamenti e metriche dei protocolli di disseminazione

Un protocollo di disseminazione lavora su reti non strutturate facendo sì che i messaggi siano trasmessi tra i nodi dell'overlay per mezzo delle connessioni tra essi. In questo modo, ogni volta che un nodo riceve un messaggio lo analizza (passandolo al modulo dell'applicazione, se necessario) e lo ritrasmette tramite algoritmo di disseminazione. Poiché le reti sono in genere assimilabili a grafi, è possibile che un nodo riceva il medesimo messaggio più volte. Per evitare che un messaggio sia ritrasmesso all'infinito e che la comunicazione fra i nodi sia congestionata da trasmissioni ridondanti, si sfruttano due pratici espedienti di implementazione: (1) ogni nodo inserisce in una cache al suo interno i codici identificatori di messaggi già processati e (2) i messaggi hanno un tempo di vita massimo all'interno della rete.

Per quanto riguarda la pratica del "caching", essa è in realtà una pratica assai comune in molti ambiti dell'informatica. In questo contesto i nodi sfruttano le loro cache per mantenere in memoria gli ultimi identificatori dei messaggi già processati. In questo modo, se un nodo riceve un messaggio il cui identificatore è contenuto nella sua cache, tale messaggio può essere "lasciato decadere" (eliminato dal sistema) al fine di evitare di spendere risorse su messaggi ridondanti. Un fattore chiave è la dimensione delle cache: la facilità di evitare ritrasmissioni ridondanti è tanto maggiore quanto più grande è questa dimensione. Il problema a quel punto sono i requisiti di memoria, che devono essere necessariamente più grandi a loro volta. [xiv]

Passando al secondo punto, ai messaggi viene associato un valore di Time-to-Live (TTL). Il time-to-live impedisce che i messaggi siano ritrasmessi senza fine lungo le connessioni della rete. Ogni volta che un messaggio è ritrasmesso sulla rete, il suo TTL viene decrementato e non appena questo valore scende a 0 il nodo destinatario dell'ultima ritrasmissione non lo ritrasmette più ai suoi vicini e lo elimina dal sistema. La ricerca del TTL ideale è una questione importante per le performance del protocollo di disseminazione. Esso dovrebbe essere sufficientemente grande da garantire che il messaggio possa essere disseminato per tutta la rete. Quando si impiega un protocollo di puro *flooding*, il dato del diametro della rete può essere impostato come time-to-live e, in quel caso, è necessario compiere un'operazione di stima di tale valore. [xv] Tuttavia, quando consideriamo il diametro di un network ci basiamo sui cammini più brevi tra i nodi e in generale questo non significa che la lunghezza media di un cammino sia in scala con il valore del diametro. Quando il protocollo di disseminazione è invece basato sul gossip, non c'è garanzia che un messaggio proveneniente da una sorgente raggiunga un dato nodo tramite il percorso più breve. [xiv]

Una volta che si decide di impiegare un protocollo di disseminazione per diffondere informazioni, è importante definire le metriche atte a valutarlo.

Una proprietà desiderabile in un protocolo di disseminazione è quella di essere in grado di far giungere un messaggio a tutti i nodi e possibilmente nel modo più veloce. Pertanto si usa un parametro chiamato **copertura** (*coverage*) che denota la frazione di nodi che hanno effettivamente ricevuto i messaggi. Idealmente, si spera di ottenere un rapporto di copertura del 100%, che significa che tutti i nodi hanno ricevuto tutti i messaggi generati. È ovviamente difficile stimare tale valore in modo teorico, in special modo se consideriamo l'introduzione di caching e TTL. Questo parametro viene misurato tramite simulazioni di disseminazione di dati.

È inoltre importante identificare appropriate metriche di costo, cosicché si possa effettuare un confronto nelle stesse condizioni fra tutti i protocolli di disseminazione. Per questo fine, una unità di misura utile è l'**overhead ratio** (a volte indicata con la lettera greca ρ) che è calcolata come segue [xvi]:

$$\rho = \frac{Messaggi\ consegnati}{Limite\ inferiore}$$

dove "messaggi consegnati" rappresenta il numero totale di messaggi che vengono consegnati da uno specifico protocollo di disseminazione ed il "limite inferiore" è il numero minimo di messaggi (in ciascun grafo) che è necessario per ottenere una copertura completa. Il limite inferiore rappresenta il numero di messaggi inviati da un protocollo di broadcast che consegnano eventi lungo i vertici di uno spanning tree senza inviare mai duplicati. Il limite inferiore dipende dal grafo ed è indipendente dal protocollo di disseminazione che sarà usato. [xiv] Per esempio, in un grafo composto di $n$ nodi e nel quale $m$ eventi diversi sono generati, il limite inferiore al numero di messaggi consegnati è $\Omega(nm)$.

## 1.3 Protocolli di disseminazione

Passiamo ora a descrivere i tre protocolli di disseminazione che verranno utilizzati o comunque menzionati all'interno di questa trattazione. Siccome questo capitolo ripropone l'analisi del medesimo argomento riportata alle pagg. 9-11 di [xiv], si rimanda il lettore al testo originale per una trattazione più approfondita.

Questi protocolli sono tutti approcci basati sul gossip di tipo "push", ovvero dove il fondamento della strategia di disseminazione è l'inoltro ricorsivo di messaggi tra i peer. Un nodo che riceva un messaggio lo trasmette attivamente ad alcuni altri nodi, che ricorsivamente ripetono tale azione fino a quando una condizione di terminazione (quale l'azzeramento del time-to-live del messaggio) non è confermata. [xvii] Ai fini del nostro studio, specifichiamo qui che il modello di riferimento è che tutti i nodi siano in grado di generare un nuovo messaggio da disseminare nella rete. Quando la procedura di generazione è invocata in un dato nodo, un singolo messaggio viene creato con una certa probabilità, come descritto nell'algoritmo 1. La generazione di un messaggio simula l'avvenire di un nuovo evento prodotto presso un dato nodo che deve essere propagato. Se il messaggio viene effettivamente creato, allora viene spedito attraverso la rete, usando la procedura Disseminate() in riga 6. Il messaggio è inoltre inserito nella cache in riga 5.

**Algoritmo 1** Generazione di un messaggio

```
1:  function Generate()
2:  t ← Generation_Threshold()
3:  if Generate_Random_Double() < t then
4:      msg ← Create_Message()
5:      Cache(msg)
6:      Disseminate(msg)
7:  end if
```

Alla ricezione di un dato messaggio (vedi algoritmo 2), il nodo ricevente inoltra il messaggio ai suoi vicini chiamando la funzione Disseminate(), visibile alla riga 5 dell'algoritmo. Questa operazione viene svolta solo se il messaggio non è già presente nella cache del nodo. Difatti, se l'identificatore del messaggio è presente nella cache, significa che è già stato dis-

seminato e pertanto il nodo non deve fare nulla con msg (riga 2). In caso contrario e se il TTL non è già azzerato, msg viene trasmesso e messo in cache (riga 3). Come già spiegato nel capitolo 1.2, la cache è limitata in quanto a dimensione a quella impostata dal programmatore.

**Algoritmo 2** Ricezione di un messaggio

```
1:  function Receive(msg)
2:  if (NotCached(msg) and msg.ttl > 0) then
3:      Cache(msg)
4:      msg.ttl ← msg.ttl - 1
5:      Disseminate(msg)
6:  end if
```

### 1.3.1 Gossip a probabilità fissa

Analizziamo questo primo protocollo, rappresentato in pseudocodice nell'algoritmo 3. Esso prevede che un nodo $n_i$ selezioni in modo casuale i vertici tramite i quali il messaggio *msg* sarà propagato. [xviii] [xix] Nello specifico, tutti i vicini di $n_i$ (cioè $\prod_i$) sono presi in considerazione e si applica un valore soglia $\gamma \leq 1$ così da determinare la probabilità che *msg* venga trasmesso tramite gossip al vicino. Quando $\gamma = 1$ otteniamo un algoritmo di flooding. Ad ogni step il messaggio è propagato da $n_i$ a $\gamma|\prod_i|$ altri nodi. Ne consegue che maggiore è il numero dei vicini, maggiore è il carico di lavoro.

**Algoritmo 3** Gossip con probabilità fissa di disseminazione

```
1:  function Initialization()
2:  γ ← ChooseProbability()
3:
4:  function Disseminate(msg)
5:  for all n_j ∈ ∏_i do
6:      if Generate_Random_Double() < γ then
7:          Send(msg, n_j);
8:      end if
9:  end for
```

### 1.3.2 Broadcasting probabilistico

Il secondo protocollo di disseminazione che prenderemo in considerazione è un metodo di broadcast probabilistico (vedi algoritmo 4). Una volta che la procedura Disseminate() è stata chiamata, se il messaggio è stato generato localmente presso il nodo e *msg* deve ancora essere propagato sulla rete (si immagini che First_Transmission() in riga 5 assolva questo compito), *msg* è inviato a tutti i vicini del nodo (righe 6-8). Per contro, se *msg* è stato ricevuto da qualcun altro, il nodo decide con una certa probabilità $\beta$, definita all'inizio del protocollo, di inoltrare *msg* (riga 5). In caso positivo, il messaggio è inviato a tutti i vicini del nodo.

**Algoritmo 4** Broadcast probabilistico

```
1:  function Initialization();
2:  β ← Probability_Broadcast();
3:
4:  function Disseminate(msg);
5:  if (Generate_Random_Double() < β or First_Transmission()) then
6:      for all nj ∈ ∏i do
7:          Send(msg, nj);
8:      end for
9:  end if
```

### 1.3.3 Gossip dipendente dal degree

In questo metodo, un nodo decide di trasmettere il messaggio in base ad una specifica caratteristica dei suoi vicini: il numero di nodi ai quali essi sono a loro volta connessi, ossia il suo *degree*. La probabilità di gossip utilizzata non è perciò la stessa per tutti i nodi, ma dipende dal numero di vicini del nodo. Se un nodo $n$ possiede degree $i$, riceverà un messaggio da un vicino con una probabilità $\gamma(i)$, che è una funzione di tale caratteristica. L'algoritmo relativo a questa famiglia di metodi può essere ottenuto prendendo l'algoritmo 3 e sostituendo $\gamma$ con una funzione $\gamma(i)$, se $i$ è il degree del nodo considerato. Esiste un set molto vasto di possibili funzioni che possono essere impiegate come $\gamma(i)$ in questo protocollo di disseminazione.

Il ragionamento dietro a questa strategia è che i nodi con basso degree potrebbero "compensare" la loro piccola quantità di connessioni con una probabilità di ricezione più alta, nel senso di fare in modo che i loro vicini incrementino la loro probabilità di disseminazione, laddove nodi dotati di degree alto hanno una probabilità maggiore di ricevere un messaggio da uno dei loro vicini e perciò la loro probabilità di ricezione può essere ridotta perlopiù senza rischi. Si prende quest'ultima contromisura per evitare un'invasione di messaggi ridondanti.

# Capitolo 2

# Algoritmi di gossip: il banco di prova

Questa sezione presenta i dettagli del simulatore che è stato utilizzato per valutare gli impatti del fenomeno noto come free riding sulle metriche prese in considerazione da questa tesi di laurea (vedi capitolo 1.2).

## 2.1 L'ambiente di simulazione

Tutte le simulazioni compiute al fine di raccogliere i dati necessari allo studio che ci siamo proposti di condurre sono state eseguite in un ambiente di simulazione a tre livelli: l'elemento cardine della simulazione è l'Advanced RTI System (ARTÌS) [xx], un middleware di simulazione parallelo e distribuito, ispirato dall'High Level Architecture Standard. [xxi] ARTÌS è stato integrato con la Generic Adaptive Interaction Architecture (GAIA), un framework che si occupa di far migrare gli elementi della simulazione per migliorare le performance. Sul livello più alto di questa complessa architettura è posto il Large Unstructured NEtwork Simulator (LUNES) [xxii], il quale fa uso dei servizi forniti da ARTÌS e GAIA per simulare protocolli complessi al di sopra di grafi di grandi dimensioni di qualsivoglia tipologia.

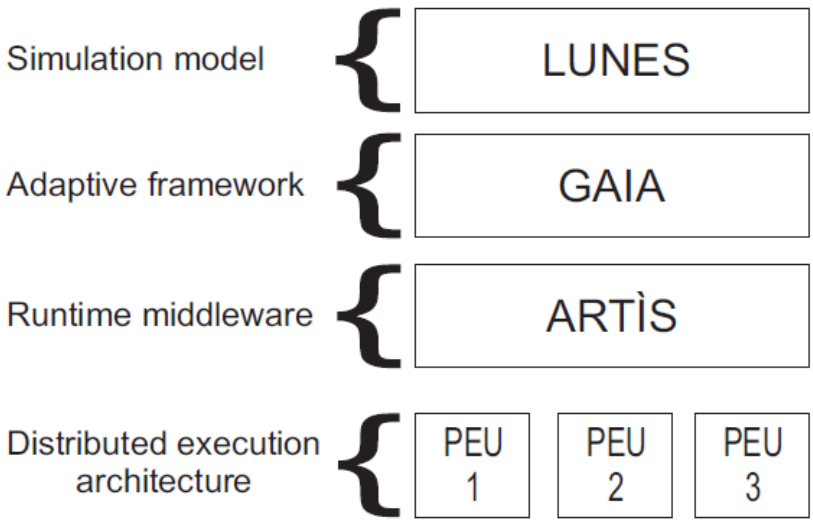



Immagine 2.1 Stack dell'ambiente di simulazione (ripreso da [xxiii] e poi modificato).

PADS (Parallel And Distributed Simulation) è l'acronimo impiegato per riferirsi all'esecuzione di processi concorrenti di simulazione su architetture computazionali fortemente accoppiate o debolmente accoppiate, rispettivamente. [xx] In altre parole, un ambiente di simulazione parallelo e distribuito è composto da un insieme di Unità Fisiche di Esezione (PEUs) come host, CPU e CPU-core, tutte connesse ad una rete comune, che può essere memoria condivisa, LAN o internet.

LUNES, cioè il livello dello stack con il quale l'utente può interagire tramite appositi script pronti all'uso, è un'applicazione modulare e separa in differenti componenti software i compiti di: (1) creazione dei network, (2) implementazione dei protocolli e (3) analisi dei risultati. L'uso di un approccio modulare offre il vantaggio di permettere l'uso, il riuso e l'integrazione di strumenti software esistenti e facilita l'aggiornamento e l'estensibilità dello strumento. [xxiv]

È degno di nota il fatto che tutti gli strumenti sono stati progettati ed implementati per lavorare in parallelo sono in grado di sfruttare tutte le risorse computazionali fornite da architetture parallele (multi-processore o multi-core) o distribuite (come può essere un cluster di PC). In altre parole, mentre, per esempio, una rete (generata dal modulo addetto alla creazione di topologie di network) è usata dal simulatore di protocollo, il suddetto modulo addetto alla creazione di topologie di network può essere attivo ed all'opera nella generazione di un altra rete. Allo stesso modo, mentre il simulatore di protocollo è in esecuzione, i risultati da esso prodotti in precedenti esecuzioni possono essere analizzati dal modulo addetto all'analisi delle tracce. I risultati provenienti da un dato modulo sono sfruttati da un altro tramite semplici file template (come ad esempio quelli scritti in DOT, linguaggio per la descrizione dei grafi).

## 2.2 Risultato di una simulazione

LUNES può condurre simulazioni ed analizzare i dati prodotti durante il loro svolgimento di tutti i protocolli di disseminazione presentati nel capitolo 1.3 su tutte le topologie di network presentate nel capitolo 1.1. Gli script la cui esecuzione è chiamata dall'utente tramite linea di comando ed il file di configurazione utilizzato dagli script stessi danno la possibilità al suddetto utente di impostare una serie di parametri tra cui i seguenti, assolutamente indispensabili perché l'applicazione possa svolgere la simulazione esattamente secondo le linee volute dall'utente:

- indirizzo della directory nella quale reperire il corpus di grafi da utilizzare nell'esecuzione delle simulazioni, la directory specifica nella sua denominazione nell'ordine la topologia del grafo, il numero di nodi, il numero di vertici, il diametro ed il numero di grafi contenuti al suo interno;
- numero di processi logici da utilizzare (simulazione sequenziale se uguale a 1, parallela se maggiore di 1);
- numero totale di esecuzioni per ciascuna configurazione;
- il time-to-live massimo dei messaggi;
- la probabilità di disseminazione (può essere un solo numero o un array di numeri, sempre compreso tra 1 e 100);
- la soglia di probabilità che un nodo sia un free rider;
- la dimensione della cache di ogni nodo.

Per ogni valore di probabilità di disseminazione saranno svolte tante simulazioni quante sono state impostate dall'utente ed i file di output prodotti a seguito dell'analisi dei dati raccolti conterran-

no i valori di media aritmetica rispetto a tutti i dati raccolti. Il contenuto di tali file di output consisterà di una singola riga di testo nella quale, per quella precisa probabilità di disseminazione e determinato protocollo di gossip, su quella particolare topologia di grafo costituita da quell'esatto numero di nodi e vertici, il simulatore ci illustra nell'ordine:

1. il numero di nodi nel grafo;
2. il rapporto di copertura medio;
3. il delay medio, cioè la media del numero di hop calcolata su tutti i nodi che hanno ricevuto un determinato messaggio e su tutti i messaggi inviati durante una singola esecuzione;
4. il numero totale di messaggi veicolati dal sistema;
5. l'overhead ratio;
6. la dimensione della cache di un peer;
7. il time-to-live massimo di un messaggio.

Per ottenere un'analisi completa delle prestazioni di un protocollo di simulazione su una determinata topologia di network è ovviamente necessario considerare tutti possibili valori della probabilità di disseminazione, nel senso di considerare tutti i numeri interi compresi tra 1 (probabilità minima) e 100 (probabilità massima). Dal momento che i parametri 2 e 5 della lista poco sopra costituiscono i valori che più ci interessano tra quelli analizzati da LUNES, viene naturale immaginare di rappresentare i dati ottenuti a seguito di una simulazione sotto forma di punti in un grafico cartesiano, in cui i due assi sono rappresentazione di copertura e overhead ratio. In tutte le successive rappresentazioni, il valore del rapporto di copertura (in percentuale) sarà quindi associato all'asse delle ordinate e l'overhead ratio (numero puro) a quello delle ascisse.

A titolo dimostrativo di quanto esposto finora, vediamo un diagramma che illustra i risultati ottenuti a seguito di una simulazione condotta utilizzando il protocollo di broadcast probabilistico su una rete ad invarianza di scala composta da 500 nodi e 1494 vertici, avente diametro 5. Durante una simulazione è stato utilizzato 1 processo logico, sono state lanciate 3 esecuzioni per ogni configurazione, il TTL massimo dei messaggi è stato impostato a 16, sono stati esaminati tutti i valori interi di probabilità tra 1 e 100, la probabiltà che un nodo sia un free rider è nulla ogni nodo ha spazio per 256 elementi in cache. Ognuno dei 100 punti contrassegnati nel grafico con un asterisco è una rappresentazione grafica di uno dei 100 file di output prodotti da LUNES usando, come già evidenziato, i valori di rapporto di copertura e overhead ratio come coordinate cartesiane.

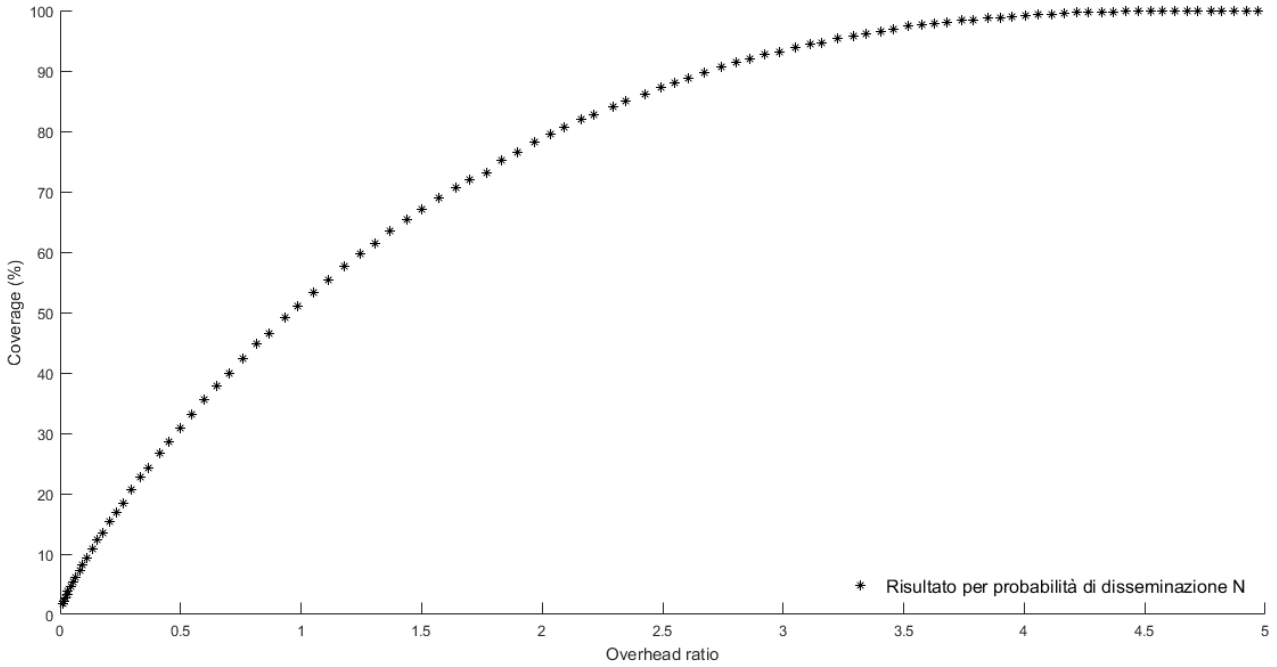



Immagine 2.2 Visualizzazione dei risultati della simulazione dimostrativa. Algoritmo: broadcast probabilistico; topologia: rete ad invarianza di scala con 500 nodi e 1494 vertici; $1 \leq N \leq 100$; TTL=16; cache=256.

Il protocollo consente di raggiungere la copertura completa dell'overlay network con il 98% di probablità di disseminazione e il valore massimo di overhead ratio si assesta su [ 4,97 ]. Il dettaglio più importante tuttavia è la disposizione ordinata dei risultati sul grafico. Il trend di crescita del dato dell'overhead ratio, unito al corrispettivo relativo al rapporto di copertura (al netto di trascurabili oscillazioni nella fase in cui il dato si approssima al massimo raggiungibile) fa sì che l'andamento generale approssimi molto strettamente quello di una curva logaritmica. Questo sarà un dettaglio importante in future simulazioni, dal momento che una distribuzione disordinata dei risultati potrebbe essere sintomo di errori di qualche tipo nell'implementazione di un protocollo.

Mettendo da parte questo primo esempio e riprendendo quanto anticipato nell'introduzione, i batch di simulazioni relative ai meccanismo adattivi sviluppati ai fini di questa tesi sono stati tutti condotti seguendo una medesima configurazione, che ora andremo ad illustrare.

Nella figura 2.3, possiamo vedere i risultati dei test condotti utilizzando il protocollo di disseminazione a probabilità fissa su una rete a grafo aleatorio composta da 500 nodi e 1500 vertici, avente diametro 7. Durante una simulazione è stato utilizzato 1 processo logico, sono state lanciate 3 esecuzioni per ogni configurazione, il TTL massimo dei messaggi è stato impostato a 10, sono stati esaminati tutti i valori interi di probabilità tra 1 e 100 e ogni nodo ha spazio per 256 elementi in cache. Per questa prima introduzione al caso di studio, la probabilità che un nodo sia un free rider è nulla.

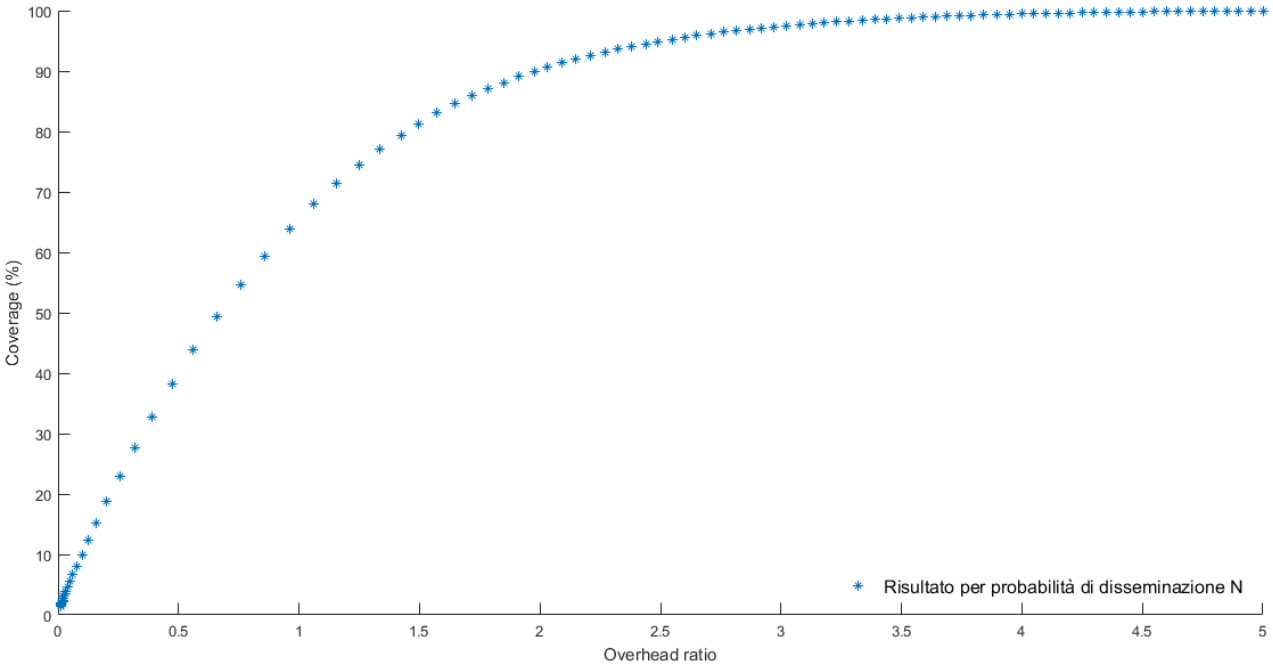



Immagine 2.3 Visualizzazione delle performance del protocollo di disseminazione a probabilità fissa su una rete a grafo aleatorio (500 nodi, 1500 vertici) in assenza di free riders. $1 \leq N \leq 100$.

In figura 2.3 la corpertura completa dell'overlay network è raggiunta con probabilità di disseminazione pari al 100% e l'overhead ratio raggiunge un valore massimo pari a [ 5 ]. Anche in questo caso è possibile notare la regolarità con la quale i risultati si dispongono sul piano cartesiano a formare un segmento di curva con andamento simil-logaritmico, nonostante il grafo sul quale sono stati condotti i test abbia una topologia differente.

## 2.3 Il meccanismo a soglia probabilistica stabile

Il meccanismo a soglia probabilistica stabile è il sistema implementato di default in LUNES tramite il quale il simulatore determina quali nodi del grafo si comporteranno come free rider durante la simulazione e quali no. Come spiegato in 2.2, l'utente può fornire all'applicazione una soglia di probabilità che un nodo sia un free rider semplicemente assegnando un valore compreso tra 0 e 1 ad una variabile d'ambiente (FREERIDER) presente in tutti gli script che si occupano di avviare le simulazioni e gestire i dati in output.

L'algoritmo 5 illustra il meccanismo con cui, al momento, LUNES individua i nodi all'interno della rete/grafo ai quali assegnare un comportamento da free rider. Come si può ben vedere dallo pseudocodice, si tratta di un algoritmo incredibilmente semplice: il valore della variabile d'ambiente FREERIDER viene preso ed usato "as it is" come soglia di probabilità. Un numero casuale compreso tra 0 e 1 viene generato (riga 2) da una funzione implementata nelle librerie dell'applicazione e, a seconda che tale numero sia minore o maggiore del valore soglia, il nodo sotto esame è definito o meno come free rider. Questa operazione di definizione nel codice vero e proprio avviene semplicemente settando ad 1 (free rider) oppure a 0 (non free rider) una variabile all'interno della struttura che costituisce l'implementazione dell'entità simulata nodo.

**Algoritmo 5** Meccanismo di determinazione dei free rider a soglia di probabilità stabile

**Elementi:** il nodo (*node*) del grafo passato come argomento a lunes_user_register_event_handler() e la variabile di ambiente *freerider_threshold* che esprime la probabilità che un nodo sia un free rider.

```
if freerider_threshold > 0 then
        freerider ← Generate_random_double(0, 1)
        if freerider <= freerider_threshold then
                node è un freerider
        else
                node non è un freerider
        end if
end if
```

In questa sezione si investigheranno gli effetti del free riding sulle prestazioni del caso di studio, con particolare attenzione al dato di copertura massima raggiungibile con il 100% di probabilità di disseminazione. Gli effetti del free riding sulle performance dei protocolli possono infatti essere molto severi [xxv] e difficilmente si può immaginare in questo contesto una conseguenza negativa più importante dell'impedire ad un protocollo di gossip di raggiungere la piena disseminazione.

Suddivisi tra le figure 2.4, 2.6 e 2.8 vediamo quindi la visualizzazione grafica degli insiemi di dati ottenuti da test su percentuali crescenti di free rider all'interno della rete. In pratica, il valore di FREERIDER viene impostato di volta in volta al valore utilizzato per la simulazione preedente aumentato di [ 0,1 ]. Le linee continue blu servono come riferimento in modo da non perdere di vista la progressiva degradazione delle performance da un'immagine all'altra. Corrispondono al tracciato seguito dall'insieme di dati ottenuto dal valore di FREERIDER più alto nell'immagine precedente, tranne nell'immagine 2.4 dove essa rappresenta invece l'andamento delle prestazioni in caso di assenza totale di free rider. Le figure 2.5, 2.7 e 2.9 mostrano delle vedute di dettaglio per rendere più chiare zone dove la quantità di dati rappresentati nei grafici è tale da inficiare la leggibilità.

È chiaramente visibile in questi diagrammi che è sufficiente anche una bassa percentuale di free rider (10% in figura 2.4) per impedire al protocollo a probabilità fissa di raggiungere la piena disseminazione sul grafo, segno che alcuni messaggi vengono scartati dai free rider e questo impedisce ad altri nodi, dotati di pochi collegamenti con il resto del grafo, di ottenere tali messaggi per altre vie. Ovviamente, questo effetto sulle performance di copertura è solo amplificato da percentuali maggiori di free rider, al punto che per il massimo valore di FREERIDER (figure 2.8 e 2.9) analizzato in questa trattazione la disseminazione dei messagi è quasi completamente bloccata: con il 90% di free rider in un grafo, un qualunque messaggio ha al massimo la possibilità di raggiungere appena il [ 2,36% ] dei nodi.

Il dettaglio più interessante che è possibile notare in questo insieme di diagrammi è che il calo della copertura massima non segue un andamento direttamente proporzionale all'incremento della percentuale di free rider. Fino ad un 40% di free rider nel network il differenziale con lo scenario senza free riding rimane assai contenuto [ -0,254 -1,264 -2,957 -5,038 ]. Al contrario, si assiste ad un abbattimento incredibilmente repentino per quanto riguarda gli scenari riportati nell'immagine 2.8, dove il dato per lo scenario con il 90% di free rider è poco più di un sesto di quello con l'80%, che a sua volta corrisponde a meno di un quarto di quello con il 70%.

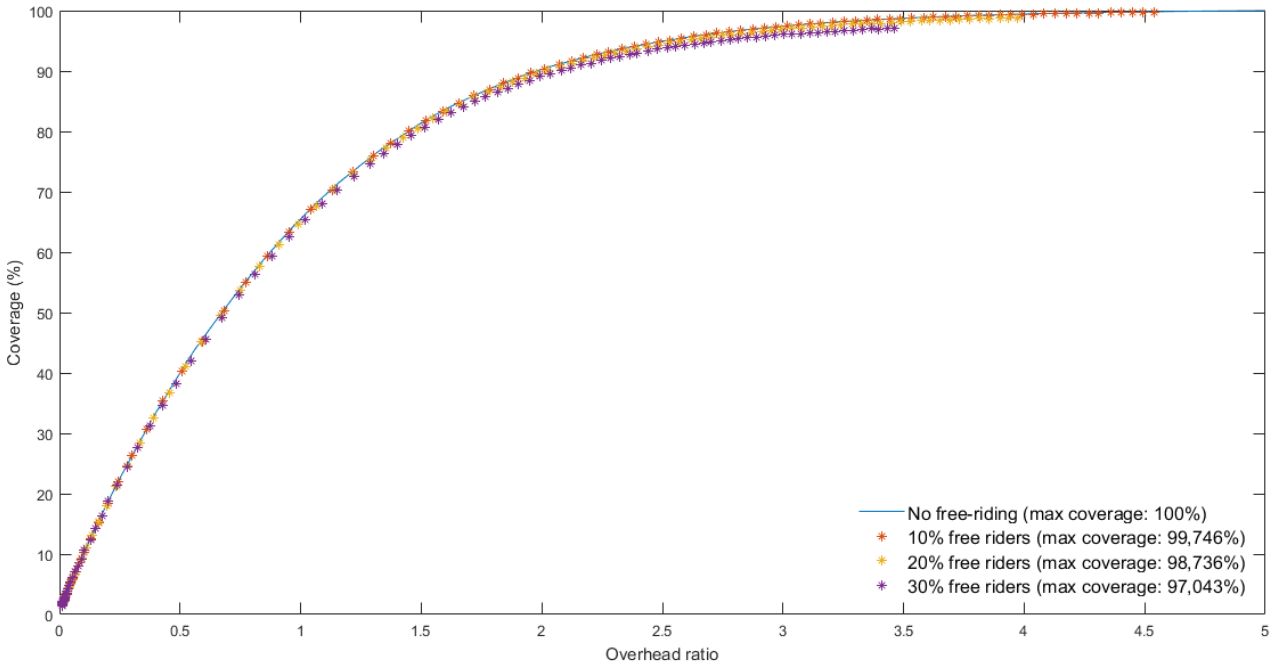



Immagine 2.4 Studio delle prestazioni del protocollo a probabilità fissa su rete a grafo aleatorio per valori di FREERIDER pari a [ 0,1 0,2 0,3 ] impiegando il meccanismo stabile per la determinazione dei free rider.

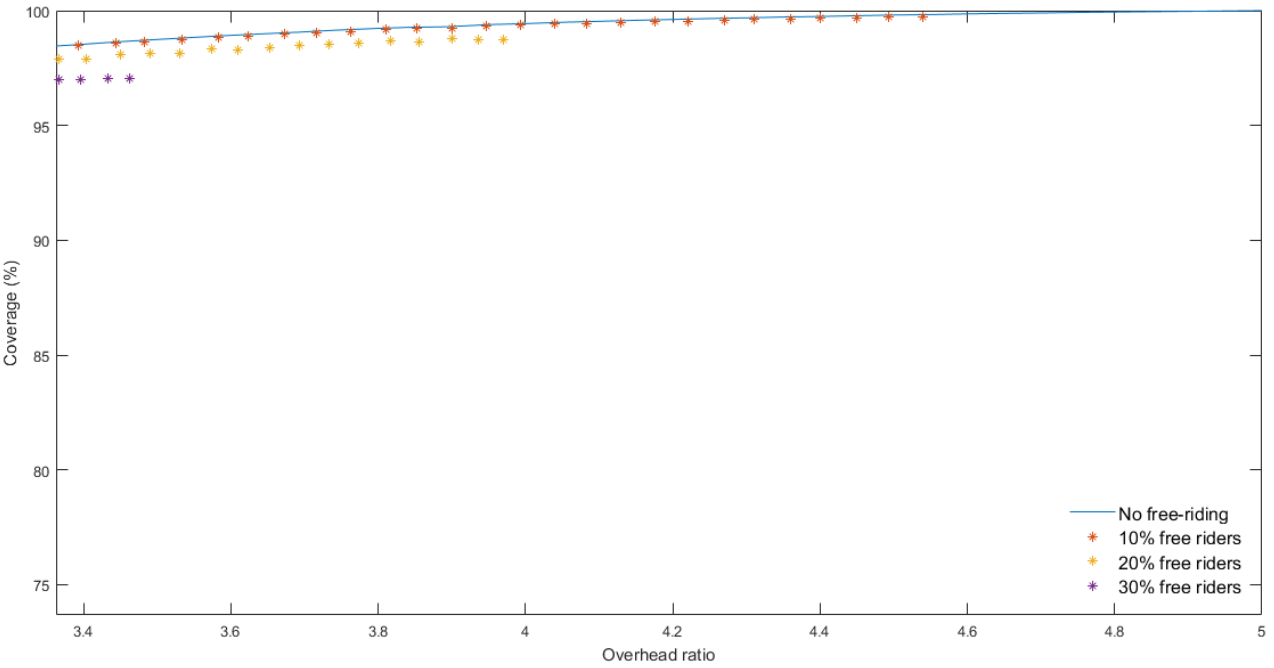



Immagine 2.5 Veduta di dettaglio dell'immagine 2.4 per valori di overhead ratio superiori a 3,4.

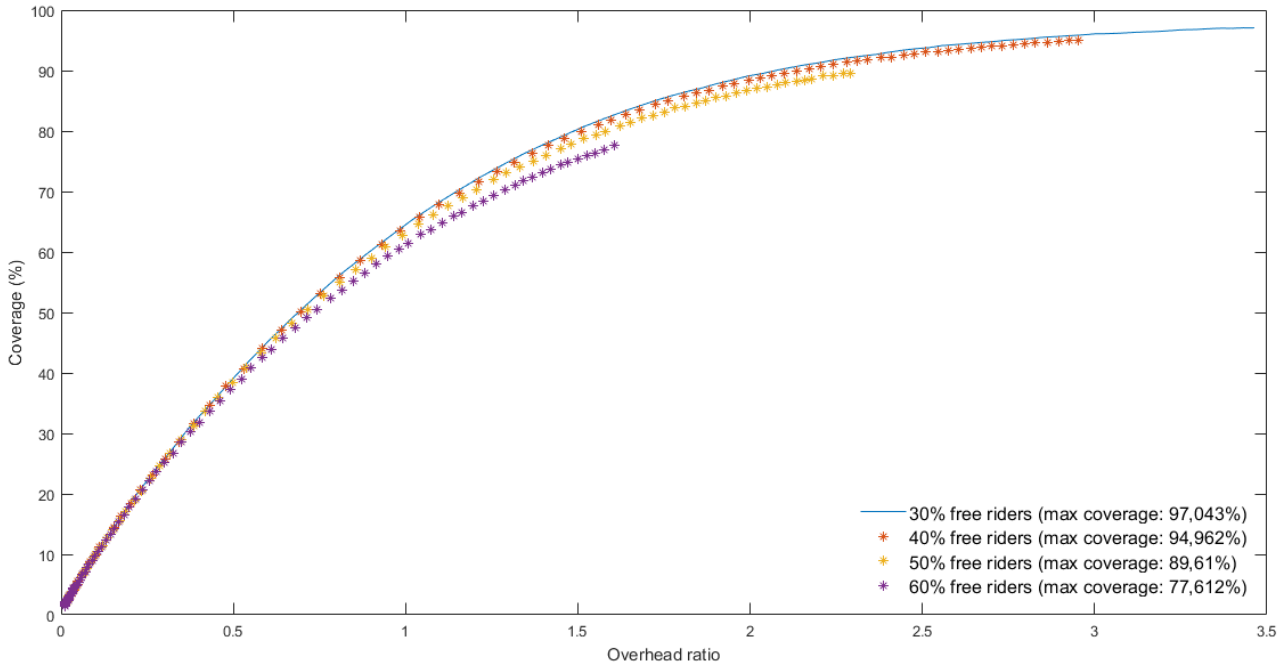



Immagine 2.6 Studio delle prestazioni del protocollo a probabilità fissa su rete a grafo aleatorio per valori di FREERIDER pari a [ 0,4 0,5 0,6 ] impiegando il meccanismo stabile per la determinazione dei free rider.

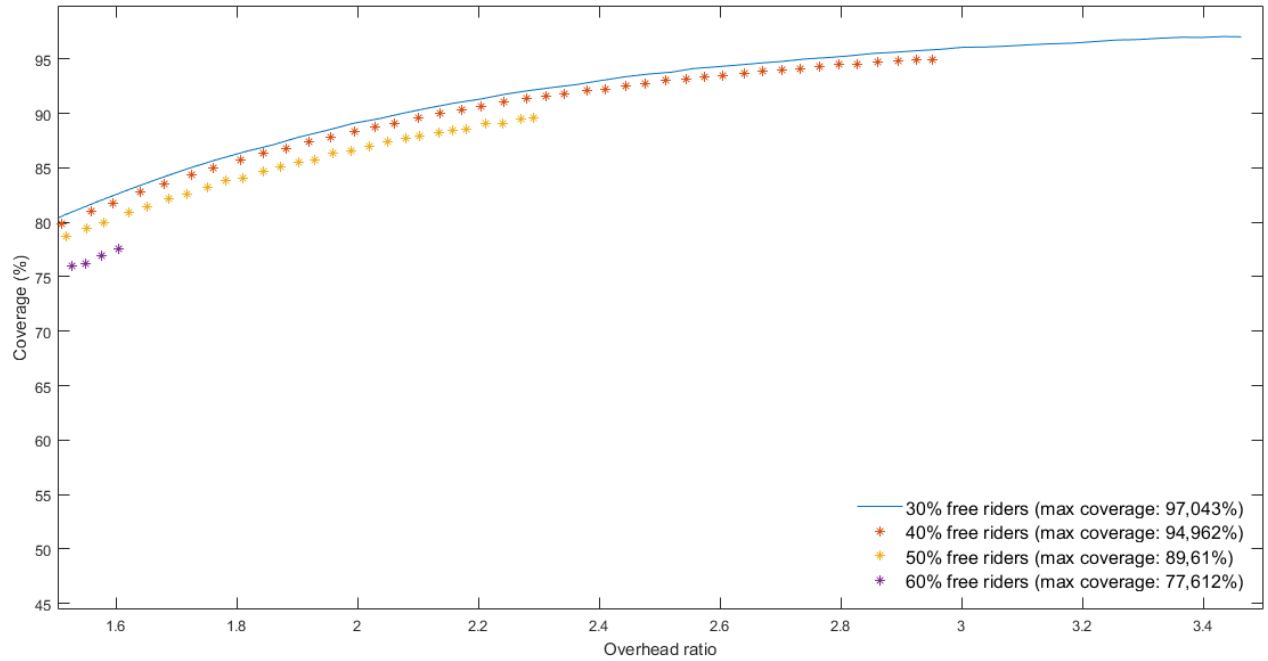



Immagine 2.7 Veduta di dettaglio dell'immagine 2.6 per valori di overhead ratio superiori a 1,5.

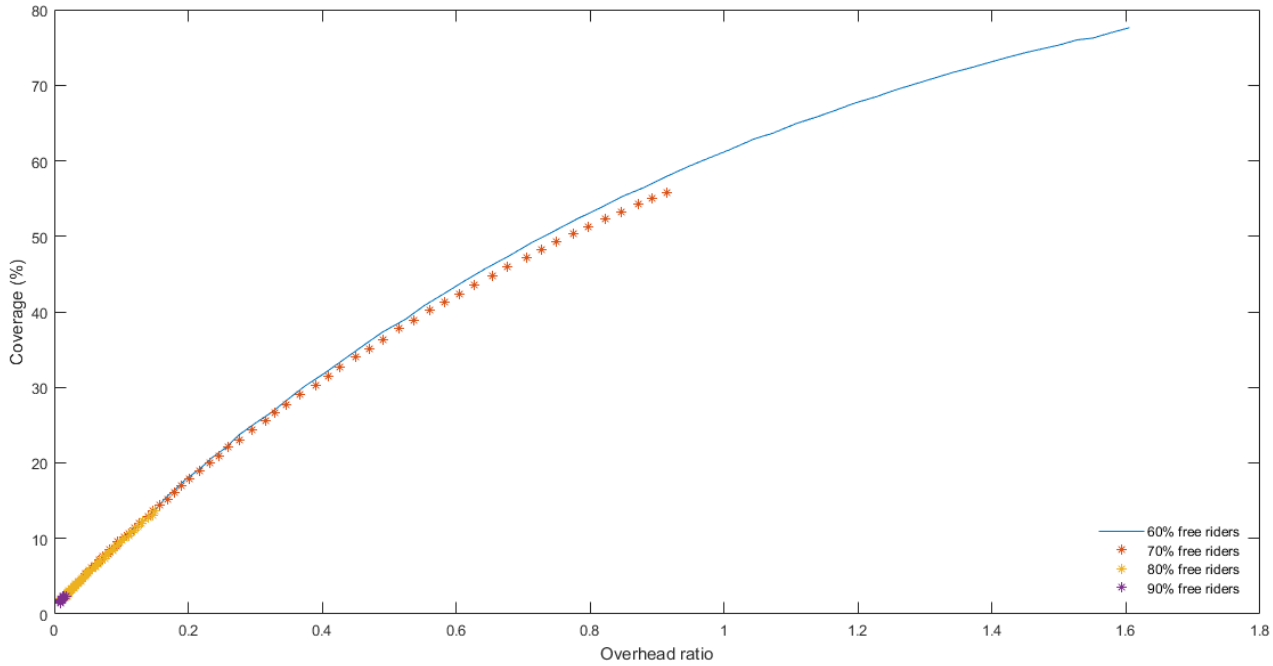



Immagine 2.8 Studio delle prestazioni del protocollo a probabilità fissa su rete a grafo aleatorio per valori di FREERIDER pari a [ 0,7 0,8 0,9 ] impiegando il meccanismo stabile per la determinazione dei free rider.

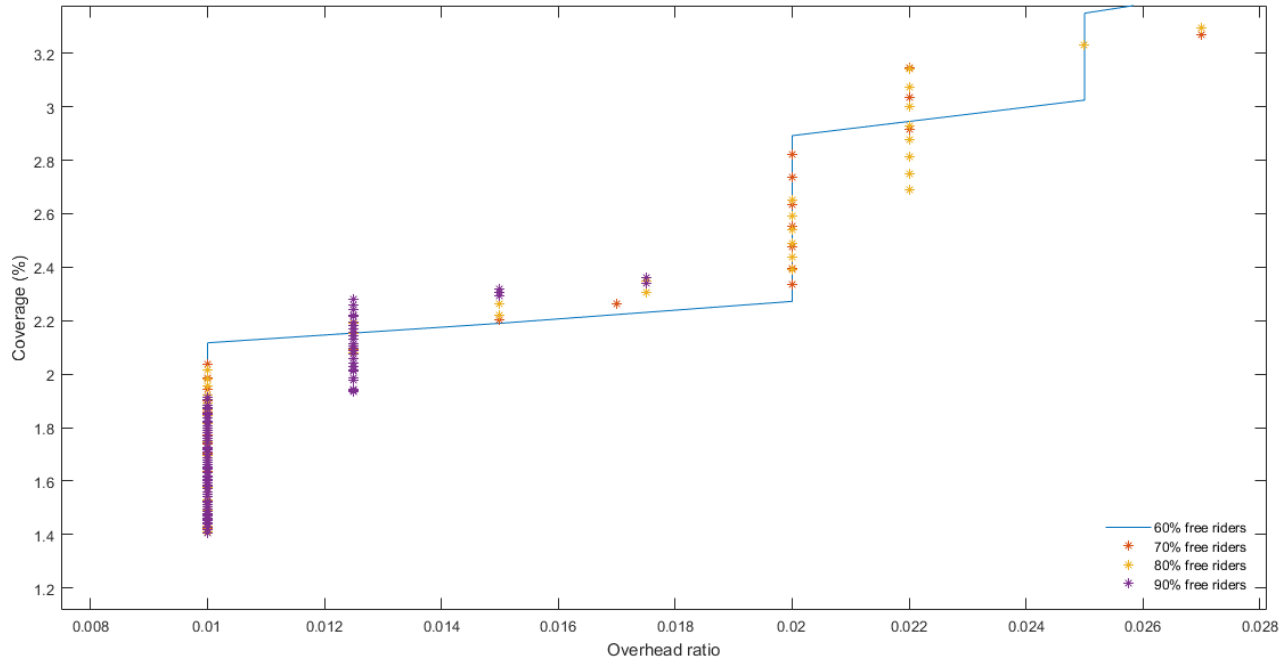
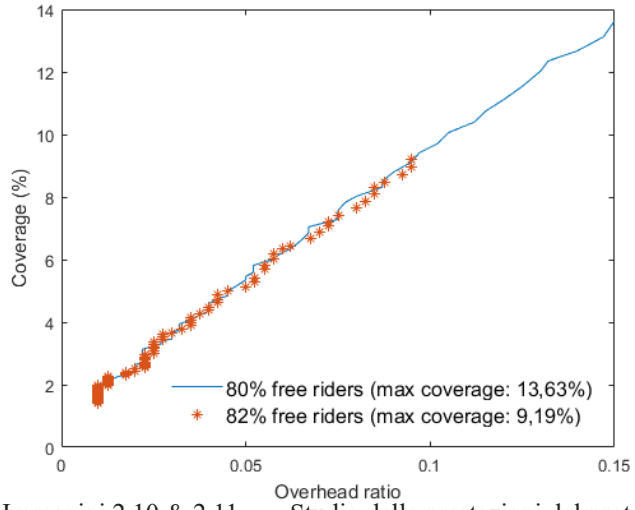
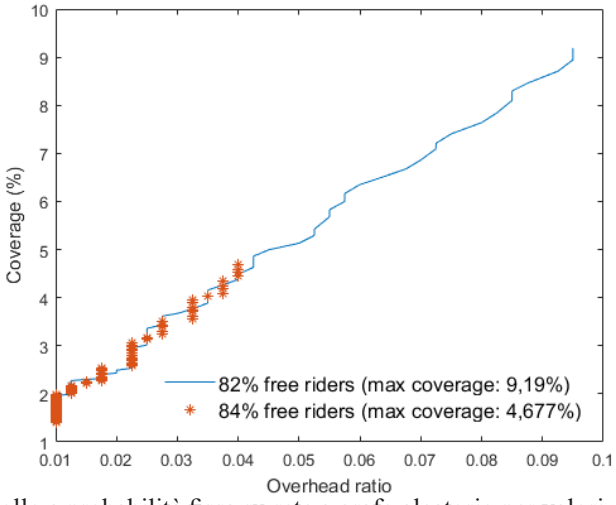



Immagine 2.9 Veduta di dettaglio dell'immagine 2.8 per valori di overhead ratio minori di 0,028.

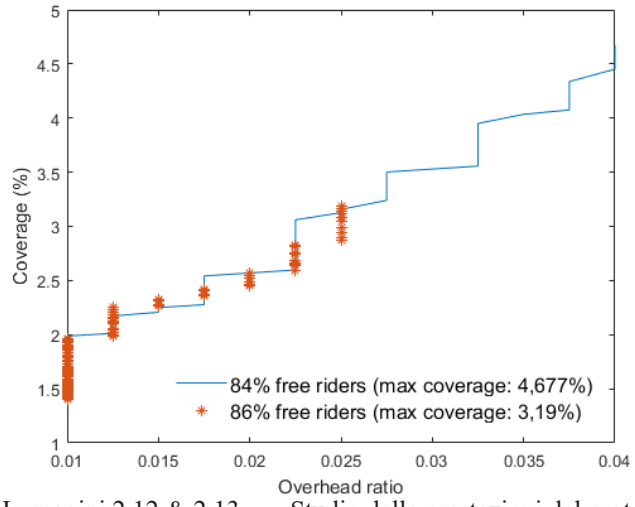
Il precipitoso calo delle performance per valori di FREERIDER tendenti a 1 è un fenomeno talmente drastico che merita un qualche tipo di approfondimento. Nelle figure 2.10, 2.11, 2.12 e 2.13 si analizza con maggiore precisione l'andamento dei valori di copertura e overhead ratio fra i due scenari con 80% e 90% di free rider, rispettivamente, tramite incrementi successivi di [ 0,02 ] della variabile d'ambiente FREERIDER.

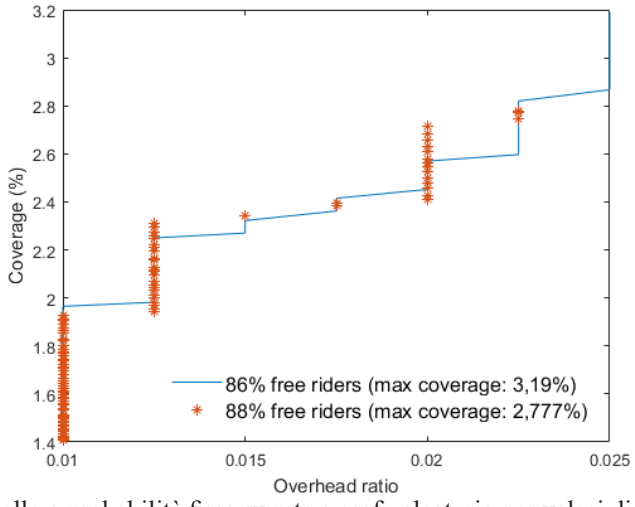
Questo nuovo batch di simulazioni rivela che il rapporto massimo di copertura continua a scendere rapidamente in figura 2.10, figura 2.11 e figura 2.12, con decrementi percentuali nell'ordine di [ -33% -49% -32% ] rispettivamente. Anche in figura 2.13 si assiste ad un decremento [ -13% ] tra i valori di ordinata massima dell'insieme di dati della simulazione corrente e quelli della precedente, ma di entità molto più contenuta. La spiegazione potrebbe essere che arrivati ad un valore di soglia per la determinazione dei free rider pari a 0,86 (o comunque un valore molto vicino a questo numero) l'overlay network è talmente pieno di "ostacoli" alla disseminazione di dati che il valore di copertura è sceso a livelli talmente infimi che, pure aumentando la percentuale di free rider nel network, il rapporto di copertura non scende più tanto drasticamente quanto nei passaggi precedenti.




Immagini 2.10 & 2.11 Studio delle prestazioni del protocollo a probabilità fissa su rete a grafo aleatorio per valori di FREERIDER pari a [ 0,82 0,84 ] impiegando il meccanismo stabile per la determinazione dei free rider.




Immagini 2.12 & 2.13 Studio delle prestazioni del protocollo a probabilità fissa su rete a grafo aleatorio per valori di FREERIDER pari a [ 0,86 0,88 ] impiegando il meccanismo stabile per la determinazione dei free rider.

# Capitolo 3

# Free Riding e Time-to-Live

Le simulazioni riportate nel precedente capitolo sono tutte basate su tre presupposti chiave: *in primis*, che esista una probabilità omogenea, in ogni punto della rete ed in ogni momento della durata della simulazione, che un nodo si comporti come un free rider; *in secundis*, che in base a questo valore di probabilità si determinino quali nodi sono free rider e quali no nel momento stesso in cui il grafo/rete viene generato; *in tertiis*, che un free rider si comporti sempre come tale una volta che il processo di creazione del grafo/rete è stato completato. Ma cosa succede se neghiamo tutti questi presupposti?

Da questo momento in poi verrà affrontato l'argomento cardine dell'intera tesi, cioè la definizione di meccanismi adattivi per la scelta di free rider in una rete peer-to-peer. Un nodo non sarà più impostato come free rider solo in base ad una soglia di probabilità fissata a priori, ma entreranno in gioco fattori che variano a seconda del tempo e della posizione. Per cominciare, in questo capitolo verrà trattato un meccanismo che lega la condizione di free rider di un nodo destinatario al time-to-live (TTL) di ogni messaggio ricevuto.

## 3.1 Definizione del meccanismo adattivo

Come ampiamente visto nel capitolo 1, LUNES dà agli utenti la possibilità di osservare gli effetti del fenomeno noto come free riding su una rete peer-to-peer simulata semplicemente modificando una variabile di ambiente negli script di attivazione. Ma qual è il meccanismo con cui l'applicazione imposta una certa percentuale di nodi del grafo come free rider? Osservando il diagramma di sequenza riportato nell'immagine 3.1, è possibile notare che si tratta di un processo assai semplice, che ora si procederà ad illustrare nel dettaglio.

Ogni nuova entità simulata viene inizializzata e registrata nell'ambito di una parte del livello di simulazione nota come `mig-agents` (fasi non riportate nel diagramma). A seguito di questi passaggi iniziali, l'applicazione procede a chiamare l'Event Handler che inizializza le strutture dati del nodo. Per completare questa fase però è necessario stabilire una comunicazione con il livello utente (`Lunes`) tramite la funzione lunes_user_register_event_handler(), la quale completa il processo di inizializzazione dell'entità simulata appena creata definendo tra l'altro se il nodo è un free rider o meno. Siamo quindi arrivati al punto più interessante ai fini di questa trattazione. Ricapitolando, nel meccanismo implementato di default in LUNES, i nodi che vengono designati come free rider assumono tale ruolo immediatamente dopo essere stati creati e registrati. A questo punto rimane solo da

vedere in quali passi si articola questo processo.

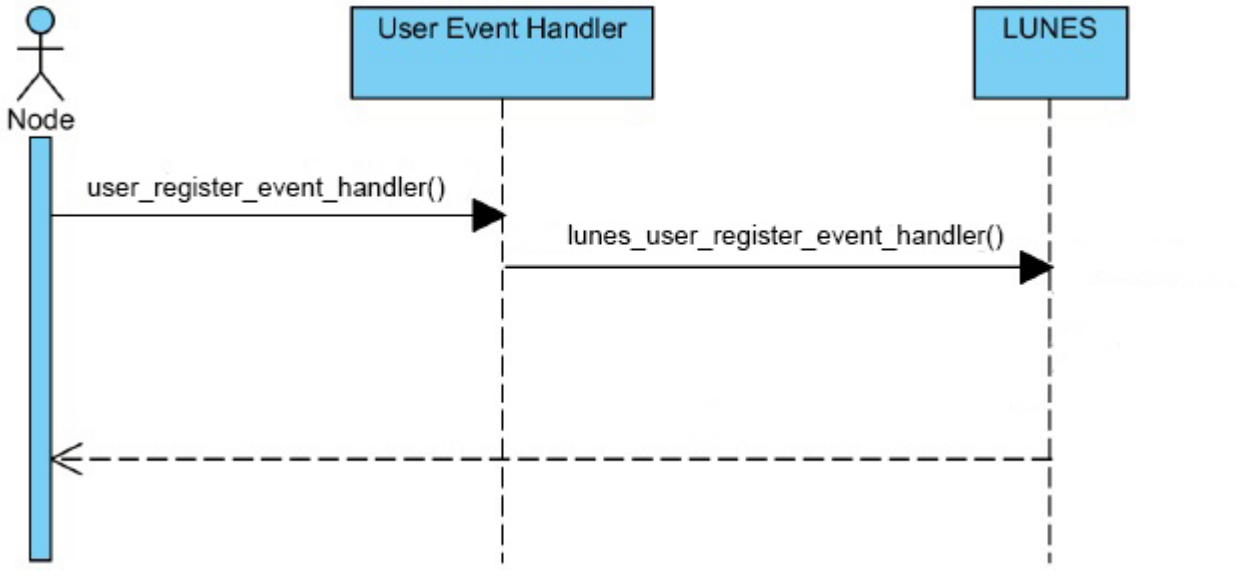



Immagine 3.1 Diagramma di sequenza UML delle chiamate a funzione a seguito dell'inizializzazione di un nodo.

Con l'introduzione di un nuovo meccanismo adattivo, sarà necessario apportare un cambiamento piccolo ma significativo alla sequenza di azioni descritta nell'immagine 3.1. Dal momento che la probabilità che un nodo si comporti da free rider è dipendente anche dal TTL rimanente di ogni messaggio ricevuto, questa parte dell'inizializzazione delle strutture dati dell'entità simulata (o SE, *simulated entity*) dovrà per forza essere scorporata da lunes_user_register_event_handler(). Tale funzione, nel sistema corrente, è pensata per essere attivata solo nel momento in cui una SE viene registrata e rimane invece inattiva durante l'esecuzione della simulazione vera e propria. È quindi più conveniente introdurre una nuova funzione all'interno di `Lunes` che si occupi esclusivamente di risettare lo status di freeriding del nodo ricevente ogni qual volta venga viene inviato un messaggio, quindi anche decine di milioni di volte a simulazione. Il nome di questa nuova funzione sarà lunes_set_freeriding() e logica vorrebbe che come argomenti le vengano passati: (1) il nodo ricevente e (2) il TTL rimanente del messaggio.

Ma in quale punto dell'applicazione è ora opportuno effettuare la chiamata a lunes_set_freeriding()? Per rispondere a questa domanda, è opportuno esaminare il sistema che gestisce la ricezione e successivo inoltro dei messaggi fra entità simulate.

Nell'immagine 3.2 è riportato il diagramma di sequenza UML delle chiamate a funzione dopo che un nodo del grafo (l'attore) ha ricevuto un messaggio ping da uno dei suoi vicini. Il livello di simulazione chiama il corretto Event Handler, il quale comunica a sua volta con il livello utente tramite le primitive di `Lunes`. Le funzioni lunes_forward_to_neighbors() e lunes_real_forward() racchiudono il nucleo dei protocolli di disseminazione, poiché implementano le regole di disseminazione di tali protocolli. Data l'importanza del time-to-live del ping ai fini dell'implementazione del sistema adattivo, è inoltre importante sottolineare che le funzioni lunes_user_ping_event_handler() e execute_ping() sono le responsabili dell'aggiornamento e della gestione del TTL. Nel dettaglio, la prima controlla se è tempo di lasciar decadere il messaggio nel caso che il TTL sia arrivato a 0 e, se tale condizione non è verificata, inizializza una nuova variabile con il time-to-live del messaggio

ricevuto decrementato di 1; la seconda crea ed invia (tramite le funzioni che implementano GAIA) nuovi messaggi ping aventi time-to-live pari al valore della variabile inizializzata in lunes_user_ping_event_handler().

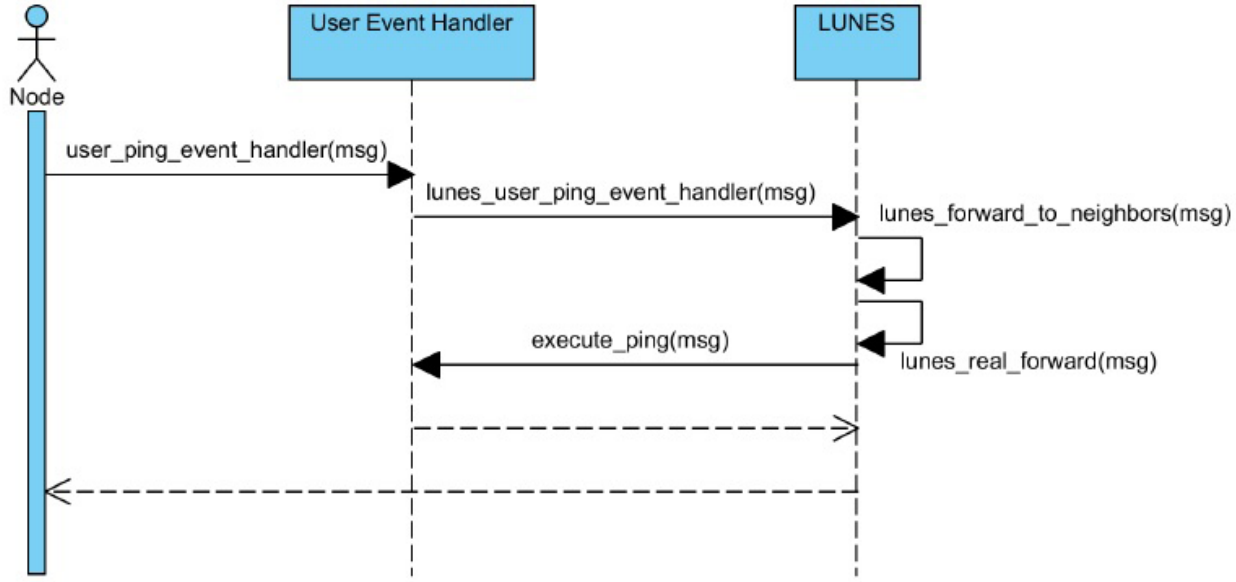



Immagine 3.2 Diagramma di sequenza UML delle chiamate a funzione ad ogni ricezione di un messaggio ping.

Non è a questo punto difficile capire che il meccanismo adattivo in esame dovrà essere implementato per funzionare *dopo* che lunes_user_ping_event_handler() abbia aggiornato il valore di TTL del messaggio ricevuto e *prima* che execute_ping() crei e spedisca i nuovi con TTL aggiornato. La funzione lunes_set_freeriding() dovrà quindi essere chiamata all'interno di lunes_real_forward(), dove avviene l'analisi dei vicini del nodo e, in seguito alla valutazione probabilistica, i messaggi vengono inviati ad alcuni di loro. In particolare, la chiamata a lunes_set_freeriding() avverrà immediatamente prima di quella ad execute_ping(), quando il livello di simulazione avrà individuato esattamente a quali vicini inoltrare i ping. È proprio in quel momento del resto che l'applicazione ha a disposizione sia il nodo destinatario sia il valore di TTL, ovvero gli argomenti da passare alla funzione che deciderà se il ricevente sarà o meno un free rider.

Ora che sappiamo dove innestare il codice dal quale dipende il nuovo meccanismo adattivo, rimane solo da capire come implementarlo.

Poiché la soglia di probabilità di incontrare un free rider deve necessariamente rimanere compresa nell'intervallo tra 0 e 1, ha senso utilizzare il time-to-live corrente per ottenere un fattore moltiplicativo che non le faccia sforare in nessun caso i limiti di tale intorno. In altre parole, anche questo fattore moltiplicativo dovrà sempre essere compreso tra 0 e 1. Per ottenere tale risultato, la soluzione adottata, illustrata nell'algoritmo 2, è quella di calcolare prima di tutto il "peso" o "delta" del TTL rimanente rispetto al time-to-live massimo per i messaggi di una simulazione (riga 3 dell'algoritmo). Quest'ultimo è molto semplice da ottenere poiché tale valore è contenuto in una variabile d'ambiente che può essere inizializzata dall'utente stesso al momento di lanciare gli script di esecuzione. Il fattore moltiplicativo vero e proprio sarà poi ottenuto con una banale operazione di sottrazione riga 4) e poi moltiplicato per la soglia di probabilità (riga 5).

**Algoritmo 6** Meccanismo di determinazione dei free rider a soglia di probabilità adattiva rispetto al time-to-live del messaggio da inviare

**Elementi:** il nodo (*node*) del grafo passato come argomento a lunes_set_freeriding(), la variabile d'ambiente *environment_threshold* che esprime la probabilità impostata dall'utente che un nodo sia un free rider, il *ttl* del messaggio che sta per essere ricevuto da *node* ed il valore massimo del TTL di un generico messaggio (*max_ttl*).

```
1:    if environment_threshold > 0 then
2:        freerider ← Generate_random_double(0, 1)
3:        delta ← ( max_ttl - ttl ) / max_ttl
4:        factor ← 1 - delta
5:        adaptive_threshold ← environment_threshold * factor
6:        if freerider <= adaptive_threshold then
7:            node è un freerider
8:        else
9:            node non è un freerider
10:       end if
11:   end if
```

È molto intuitivo capire che il "caso limite" dell'algoritmo si verificherà quando il messaggio è stato creato senza passare da una procedura di ritrasmissione e quindi il il TTL sarà uguale al TTL massimo, dando origine ad un coefficiente moltiplicativo uguale a 1. L'altro caso limite in teoria sarebbe quando un messaggio ha compiuto tutte le ritrasmissioni possibili e quindi il suo TTL si è azzerato, così come il fattore moltiplicativo che ne deriva, ma in quelle circostanze l'applicazione è già programmata per lasciar decadere il messaggio dopo aver effettuato l'ultima ritrasmissione.

Ora che l'implementazione di questo primo sistema adattivo è completa, possiamo cominciare la fase di test. Per quanto sarebbe interessante osservare il comportamento della versione modificata di LUNES in vari scenari, ognuno dei quali relativo ad un diverso algoritmo di gossip, in questa trattazione le simulazioni saranno tutte effettuate in un contesto dove la disseminazione di messaggi avviene seguendo l'algoritmo definito come "a probabilità fissa".

## 3.2 Fase di simulazione

Prima di entrare nel dettaglio di questa sezione della tesi, riassumiamo il caso di studio: tutte le simulazioni qui illustrate sono state realizzate su reti a grafo aleatorio, dove cioè le connessioni tra i nodi sono generate randomicamente. Tutti i grafi che vedremo in questo e nel prossimo capitolo contano al loro interno 500 nodi e 1500 connessioni. Il diametro massimo del grafo è 7. Il time-to-live massimo di un messaggio è fissato a 10 *hop*. La cache di ogni singolo nodo è fissata a 256 elementi. Ogni grafico riportato di seguito è del tipo già utilizzato nel capitolo 2, con la misurazione del rapporto di copertura (in percentuale) sull'asse delle ordinate e l'overhead ratio (che funge anche da misura indiretta del numero di messaggi veicolati dal sistema) sull'asse delle ascisse. L'algoritmo di disseminazione dei messaggi è sempre quello a probabilità fissa.

L'approccio al problema di mostrare le differenze tra il meccanismo di gestione del free riding implementato di default in LUNES, che chiameremo anche "meccanismo stabile" da qui in

avanti, e quello dipendente dal TTL dei singoli messaggi è stato affrontato in maniera molto diretta. Dal momento che il nuovo meccanismo risponde a criteri adattivi ma si appoggia comunque alla stessa variabile (FREERIDER) necessaria al funzionamento della sua controparte stabile, ogni grafico presenta due insiemi disgiunti di dati: quelli raccolti a seguito di un'insieme di simulazioni che sfruttano il meccanismo stabile e quelli raccolti a seguito di un'insieme di simulazioni che sfruttano il meccanismo dipendente dal TTL, entrambi con riferimento al medesimo valore di FREERIDER. Ogni nuova batch di dati raccolti è riferita ad un valore di FREERIDER incrementato di 0,1 rispetto al precedente. I casi limite (tutti i nodi sono free rider, nessun nodo è un free rider) sono ovviamente esclusi da questa fase.

Come si può osservare già a partire dalla figura 3.3, anche in caso di meccanismo adattivo dipendente dal time-to-live, la presenza di una percentuale di free rider, ancorché piccola, è sufficiente per impedire alla disseminazione a probabilità fissa di raggiungere la piena copertura del network. È osservabile un interessante scostamento tra le "curve" formate dai due insiemi di dati nel loro percorso verso il rapporto massimo di copertura da loro raggiungibile, uno scostamento che diventa sempre più importante man mano che il valore di FREERIDER aumenta.

Le "curve" di colore rosso, quelle cioè generate da simulazioni in cui è stato adoperato il sistema adattivo, crescono in modo percepibilmente più lento di quelle di colore blu e tuttavia raggiungono in ogni caso rapporti di copertura più alti. Questa differenza tra massimi è dapprima quasi trascurabile, con un delta di appena [ +0,076 ] in figura 3.3, per FREERIDER=0,1. Questo delta però aumenta di quasi dieci volte [ +0,694 ] in figura 3.4, per FREERIDER=0,2 e continua a crescere [ +1,613 +2,344 +4,72 +10,268 +15,18 +30,122 ] fino ad un valore di FREERIDER pari a 0,8 (figura 3.10). Nell'ultimo scenario (figura 3.11), a causa del crollo delle performance già osservato per il meccanismo stabile e che interessa anche il nuovo meccanismo adeguato al TTL, questo delta si riduce ad appena [ +0,832 ]. Bisogna comunque ricordare che, date le cifre in gioco, si tratta comunque di un notevole miglioramento del rapporto di copertura (per la precisione, il valore massimo raggiunto dalla curva rossa è il [ 140% ] di quello raggiunto dalla curva blu).

Per concludere, è possibile notare come le curve generate dal meccanismo adattivo raggiungono, nel loro cammino quasi asintotico (ben visibile nelle figure da 3.3 a 3.8), valori di overhead ratio anche molto superiori a quelli delle curve generate dal meccanismo stabile, indice di una variazione, pure consistente, nel traffico di messaggi nel corso delle simulazioni. Questa differenza raggiunge presto un'entità molto consistente. Come è visibile già in figura 3.6, per un valore di FREERIDER pari a 0,4 l'overhead ratio del meccanismo dipendente dal TTL con probabilità di disseminazione del 100% è pari quasi al doppio di quello del meccanismo stabile con medesima probabilità di disseminazione. Per FREERIDER=0,7 (figura 3.9), il valore massimo di ascissa raggiunto dalla curva rossa è circa il triplo del corrispettivo per la curva blu. Per FREERIDER=0,8 (figura 3.10), lo stesso valore è nove volte superiore. Con l'abbattimento delle performance che avviene superata questa soglia, anche l'overhead ratio massimo della "curva" rossa in figura 3.11 crolla ad appena [ 0,03 ], registrando comunque un incremento dell'[ 80% ] rispetto all'equivalente della "curva" blu.

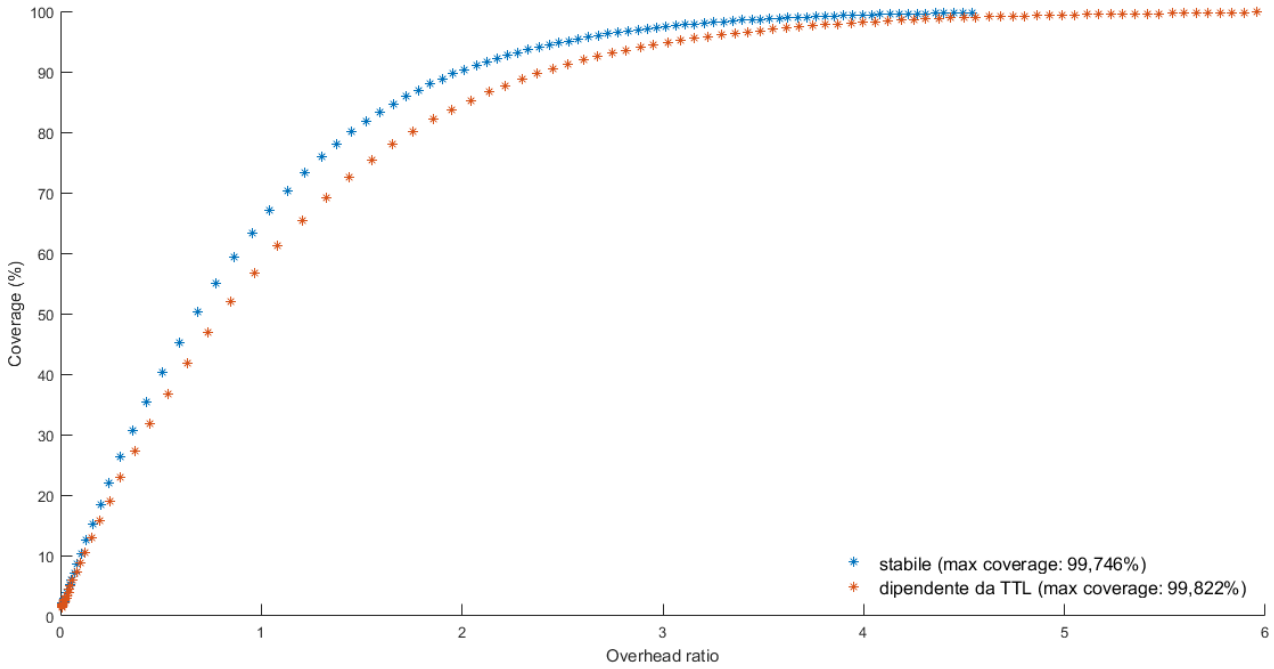



Immagine 3.3 Confronto tra meccanismo stabile e meccanismo dipendente da TTL, FREERIDER=0,1.

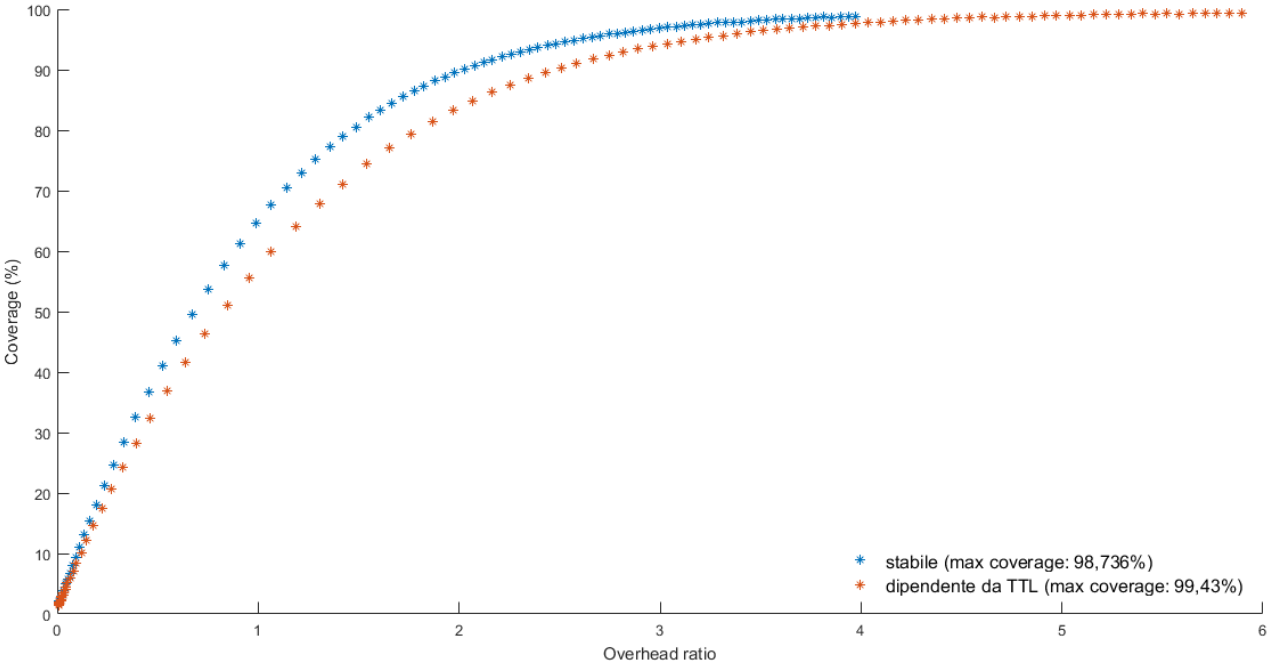



Immagine 3.4 Confronto tra meccanismo stabile e meccanismo dipendente da TTL, FREERIDER=0,2.

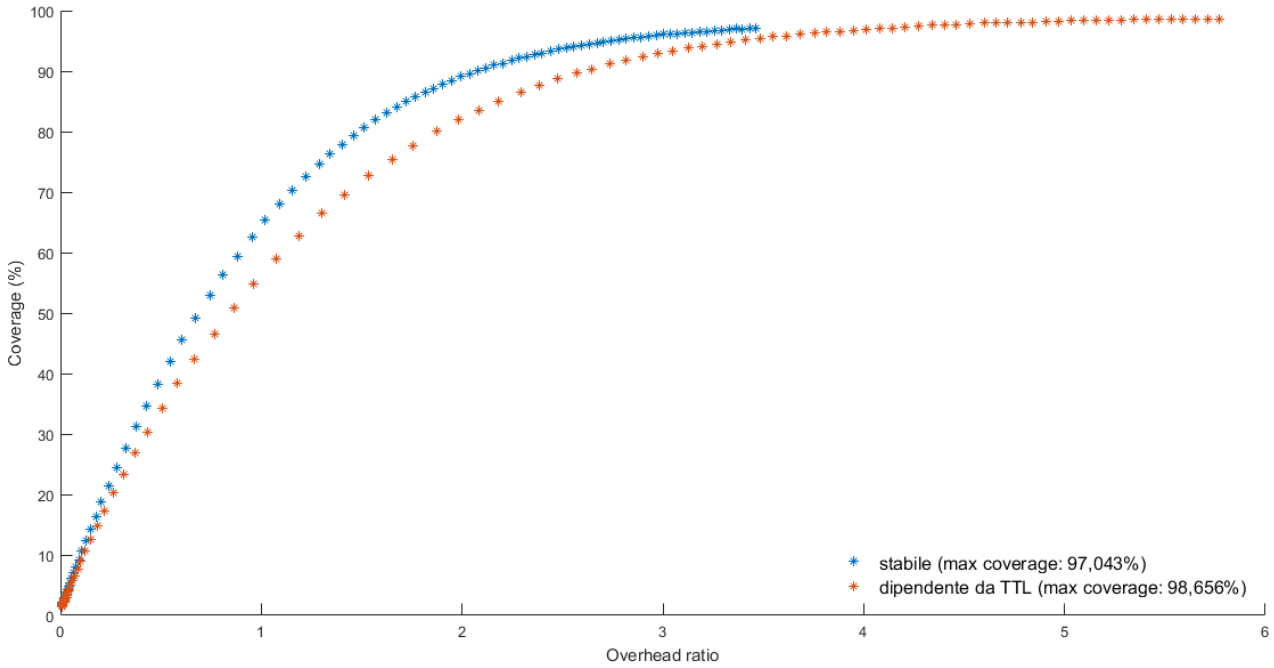



Immagine 3.5 Confronto tra meccanismo stabile e meccanismo dipendente da TTL, FREERIDER=0,3.

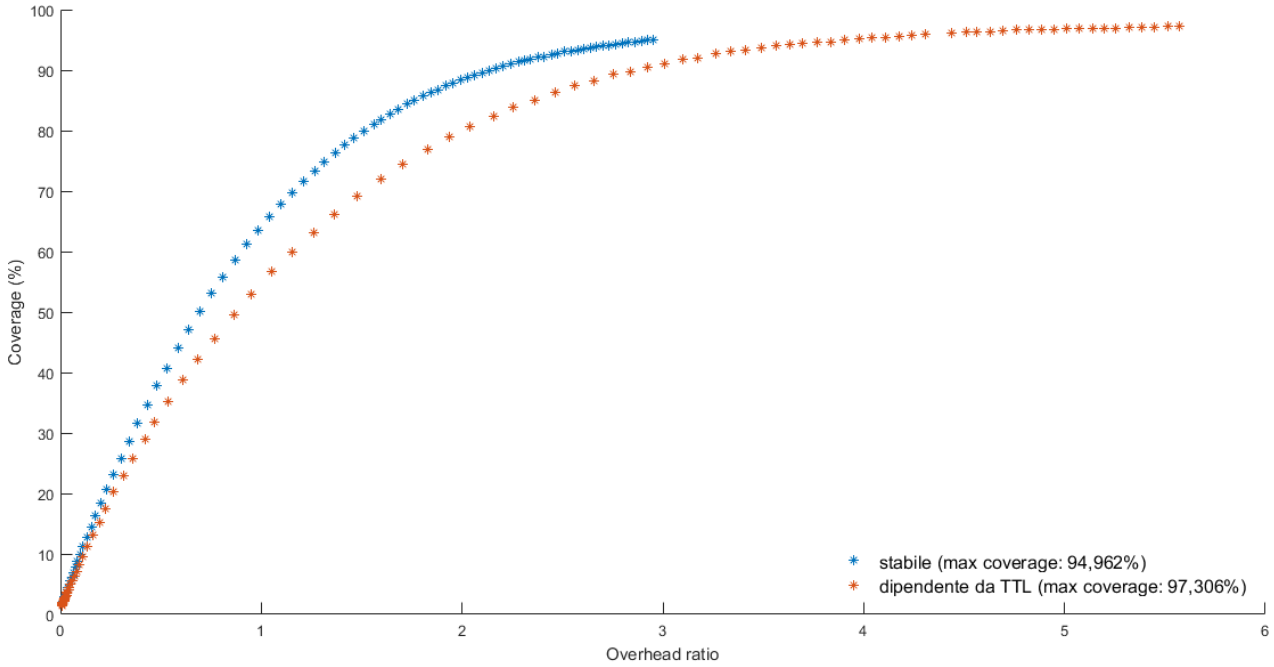



Immagine 3.6 Confronto tra meccanismo stabile e meccanismo dipendente da TTL, FREERIDER=0,4.

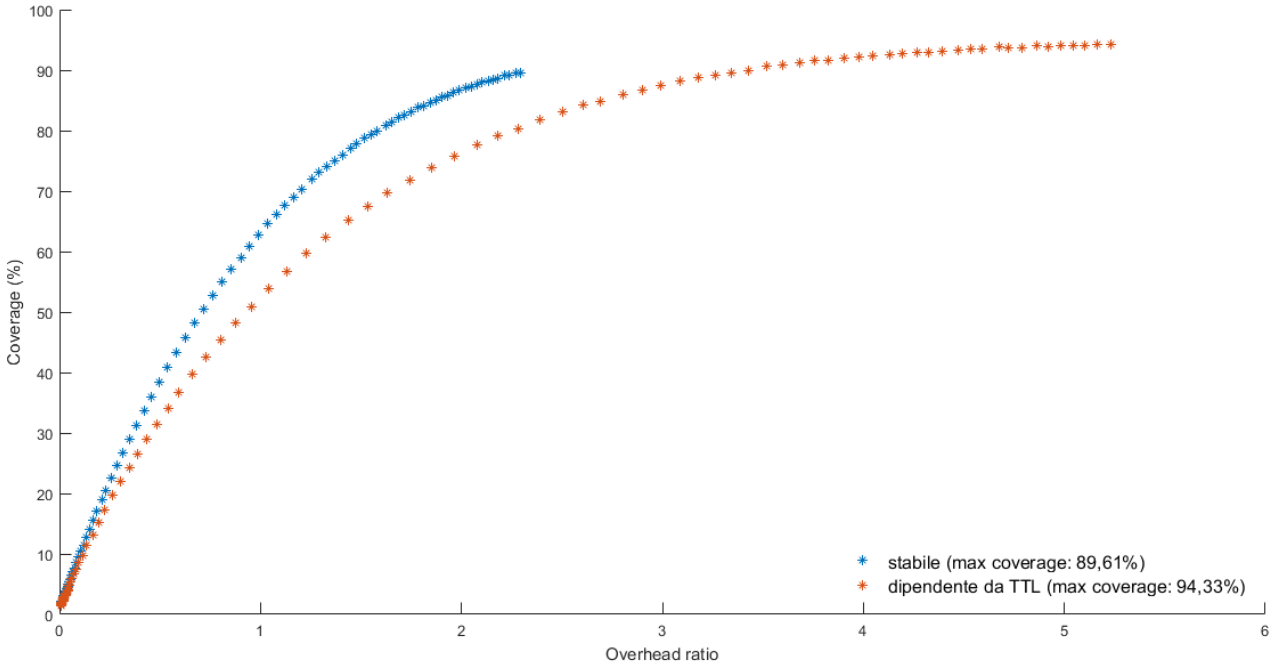



Immagine 3.7 Confronto tra meccanismo stabile e meccanismo dipendente da TTL, FREERIDER=0,5.

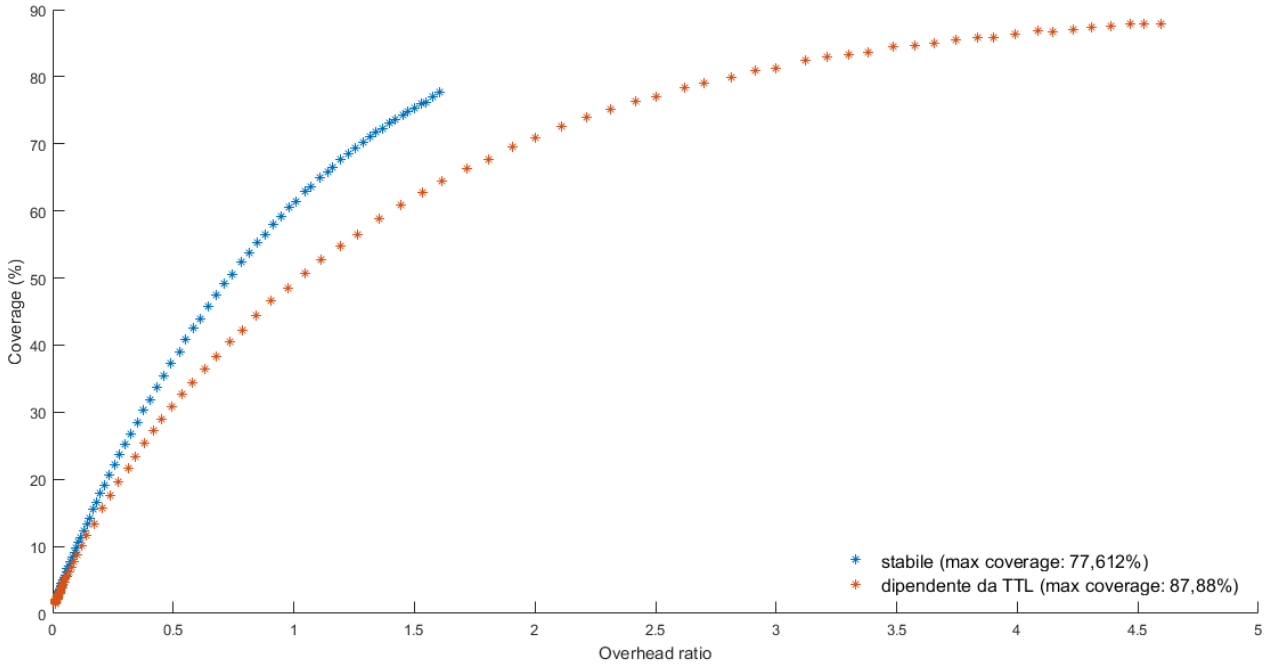



Immagine 3.8 Confronto tra meccanismo stabile e meccanismo dipendente da TTL, FREERIDER=0,6.

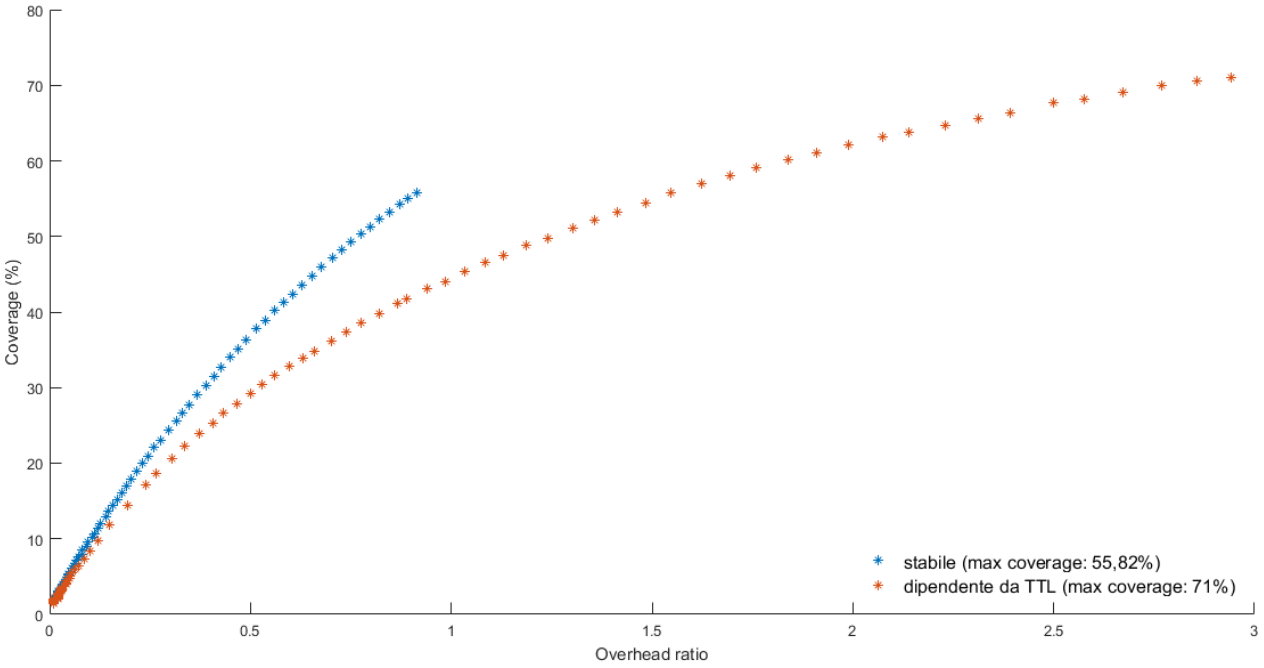



Immagine 3.9 Confronto tra meccanismo stabile e meccanismo dipendente da TTL, FREERIDER=0,7.

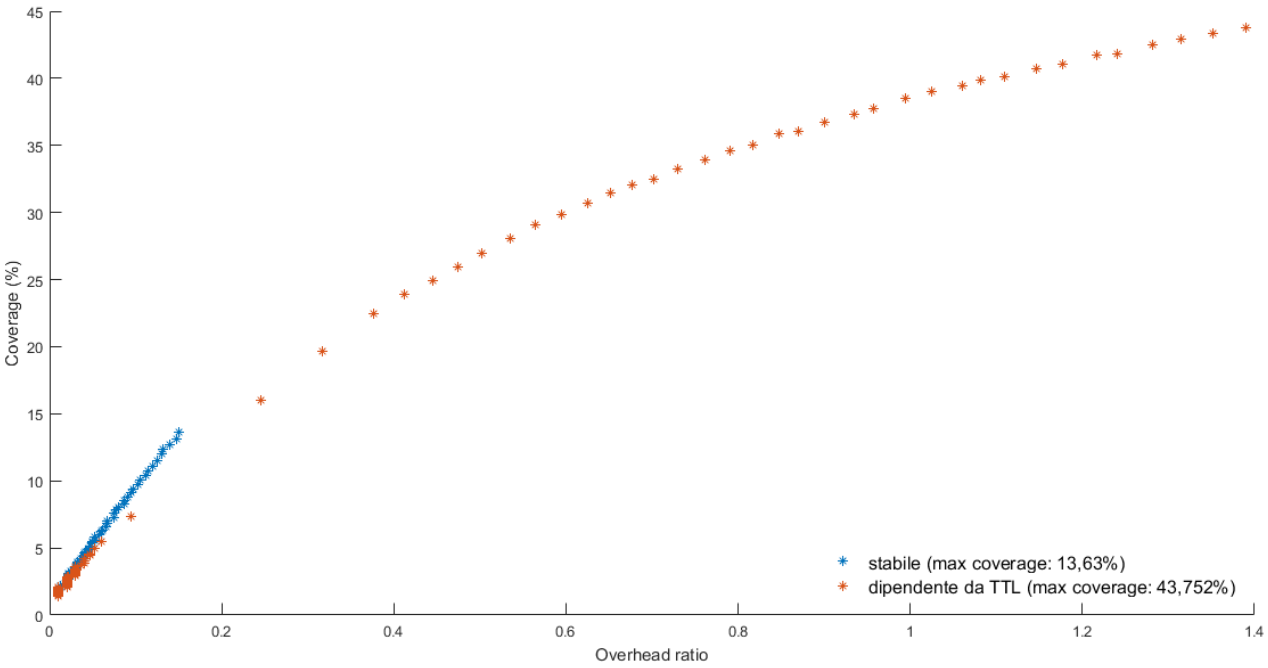



Immagine 3.10 Confronto tra meccanismo stabile e meccanismo dipendente da TTL, FREERIDER=0,8.

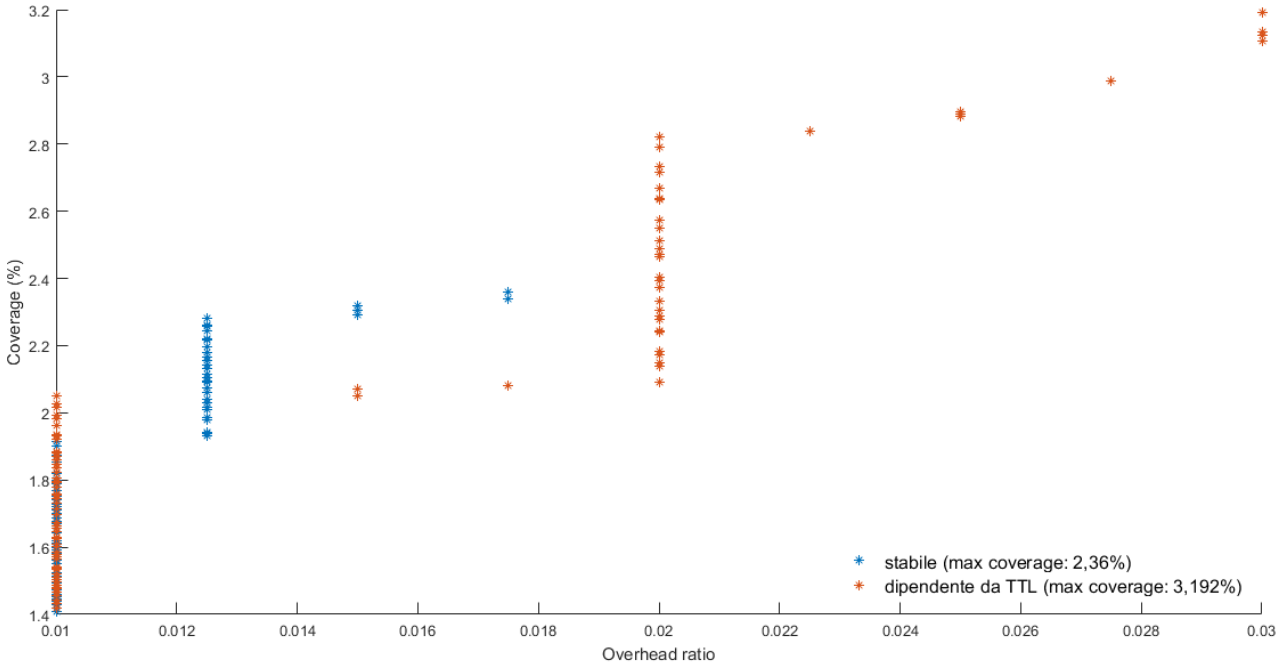



Immagine 3.11 Confronto tra meccanismo stabile e meccanismo dipendente da TTL, FREERIDER=0,9.

## 3.3 Interpretazione dei risultati

La prima domanda che è opportuno porsi a riguardo dei risultati dei test svolti è se questi siano o meno sensati. Una risposta negativa a questo quesito infatti sarebbe un chiaro invito a ripensare l'algoritmo e/o la sua implementazione e ricominciare da capo la fase di testing. Tuttavia, anche una semplice analisi visiva dei grafici prodotti ci conferma che non si riscontrano comportamenti bizzarri o fortemente controintuitivi (come potrebbe essere, ad esempio, un rapporto di copertura più alto in corrispondenza di minori probabilità di disseminazione) nei trend individuabili a fine test, ovvero le "curve" rosse. Focalizzandoci poi sui singoli dati, nessun risultato ottenuto durante le simulazioni rappresenta una decisa anomalia rispetto a tali trend. A meno di qualche minima oscillazione nelle fasi dove le curve assumono un comportamento quasi asintotico, nessun dato è localizzabile sul piano cartesiano in posizione molto distante rispetto all'andamento generale della sua curva. Per quanto approssimativa e fortemente qualitativa sia questa prima analisi, resta il fatto che non ci dovrebbe essere ragione di dubitare della sensatezza dei risultati ottenuti.

Una volta chiarito questo primo punto, possiamo ora affrontare la questione di dare una spiegazione a questi risultati, cominciando da quello più evidente alla vista in ogni singolo grafico: il consistente incremento del valore massimo di overhead ratio raggiungibile nel corso delle simulazioni. La causa dell'incremento di tale parametro, che ricordiamo essere un rapporto tra numero totale di messaggi veicolati dal sistema ed il numero minimo di messaggi necessari ad ottenere una copertura completa, può essere o un aumento del dividendo (numero totale di messaggi) o una diminuzione del divisore (limite minimo per copertura completa). Le modifiche apportate a `Lunes` non vanno in alcun modo a modificare le caratteristiche dei grafi aleatori sui quali vengono svolte le simulazioni e pertanto non c'è motivo di ritenere che avvenga una qualsivoglia alterazione del limite inferiore. La

spiegazione va perciò ricercata nel volume di traffico veicolato dal sistema. E in effetti, se si va ad osservare il dato del numero medio di messaggi scambiati nel corso delle simulazioni per la massima probabilità di disseminazione (figura 3.12), le differenze tra scenario con meccanismo stabile e scenario con meccanismo adattivo dipendente da TTL sono immediatamente visibili.

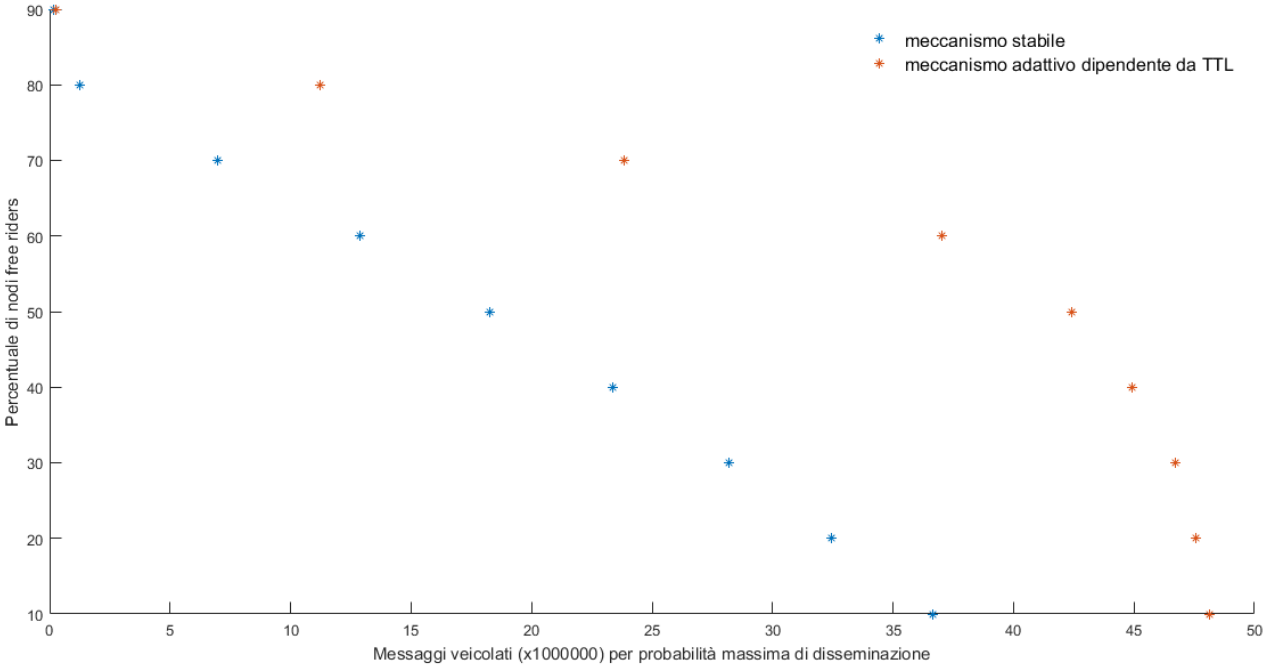



Immagine 3.12 Confronto tra numeri totali di messaggi spediti per probabilità di disseminazione pari a 100%.

Nell'insieme di dati relativo al meccanismo stabile, il numero totale di messaggi cresce in modo pressoché lineare (con l'eccezione del tratto tra FREERIDER=0,9 e FREERIDER=0,8, dove la "pendenza" subisce un'impennata) per valori via via minori di FREERIDER. Lo stesso non si può dire dei dati relativi al nuovo meccanismo adattivo. Il numero totale per il valore massimo di probabilità (tra quelle su cui si sono condotti i test, almeno) che un nodo si comporti come un free rider è sì molto vicino a quello individuato nel meccanismo stabile, ma l'andamento successivo sembra formare un ramo di iperbole e non una linea retta. Se andiamo adesso a confrontare i dati riportati in quest'ultimo diagramma con l'analisi parziale dei valori massimi di overhead ratio condotta nel capitolo 3.2, possiamo riscontrare una corrispondenza quasi perfetta tra i due insiemi di variazioni. L'ipotesi per cui i valori di overhead ratio riscontrati sono dovuti ad un maggiore traffico di messaggi appare quindi plausibile.

Ma a cosa è dovuto questo notevole aumento del numero totale di messaggi consegnati durante le simulazioni con il sistema dipendente dal TTL? La spiegazione più semplice è che con il nuovo meccanismo adattivo la possibilità che un messaggio vada perso a causa di ricezione da parte di un free rider aumenta ad ogni *hop* ma rimane sempre e comunque minore della soglia impostata dall'utente prima dell'inizio della simulazione. Nel sistema con meccanismo di gestione del free riding tale soglia agisce come unica discriminante e perciò un ping appena generato, con time-to-live ancora pari al massimo consentito, ha la stessa probabilità di un ping giunto al suo ultimo *hop* di essere recepito da un nodo free rider. Nel sistema dotato di meccanismo adattivo invece il primo evento è semplicemente meno probabile del secondo e lo stesso discorso vale per tutti i valori di TTL compresi fra questi due estremi. Una conseguenza logica di questa differenza è che nel nuovo scenario la quantità di messaggi che vengono correttamente recapitati a destinazione e ri-inoltrati ad altri nodi vicini raggiunge cifre superiori a quelle collegate allo scenario precedente; l'overhead ratio tenderà quindi a

crescere più velocemente all'aumentare della probabilità di disseminazione e a raggiungere valori più alti.

La spiegazione per questa prima differenza tra le performance del meccanismo stabile e quelle del meccanismo adattivo dipendente dal TTL può aiutarci anche a dare una spiegazione all'altra sostanziale differenza: il maggior indice di copertura della rete che viene raggiunto per ogni soglia di probabilità di free riding sulle quali siano stati condotti test. La spiegazione a questo fenomeno è del resto assai intuitiva una volta fissati tutti i concetti relativi al nuovo meccanismo. In uno scenario nel quale una quantità maggiore di messaggi viene ricevuta e ri-inoltrata all'interno dell'overlay network simulato e i free rider non sono punti fissi all'interno del grafo (un nodo si comporta da free rider solo in relazione ad un singolo messaggio ricevuto, indipendentemente dalle sue condizioni prima e dopo averlo processato) è infatti ragionevole assumere che tali messaggi possono raggiungere punti della rete che non erano invece raggiungibili in un contesto in cui i nodi free rider sono determinati subito dopo la fase di generazione il grafo aleatorio e tali rimangono per tutta la durata della simulazione.

Nel capitolo 3.2 le "curve blu" rappresentavano gli esiti delle simulazioni condotte nello scenario a free rider fissi, dove questi ultimi risultano essere delle vere e proprie barriere nell'ambito della circolazione di informazioni con ripercussioni pesanti sulle performance di copertura all'aumentare del valore di FREERIDER. Con lo sviluppo del meccanismo adattivo dipendente dal time-to-live, questi "ostacoli" non sono più fissi ed inamovibili ma contingenti ad una caratteristica dei singoli messaggi che vengono loro inoltrati e non è controintuitivo aspettarsi che le percentuali di copertura si attestino su valori maggiori. Scendendo un po' più nel dettaglio, si può inoltre notare che il miglioramento del rapporto di copertura è in un qualche modo proporzionato all' incremento del numero di messaggi veicolati dal sistema; si noti, ad esempio, che per un valore di FREERIDER pari a 0,8 si ha il massimo miglioramento della copertura (figura 3.10, max coverage del meccanismo stabile è meno di un terzo di max coverage del meccanismo dipendente dal TTL) e contemporaneamente il massimo incremento del numero medio di messaggi veicolati (figura 3.12, [ 1 271 573,75 ] contro [ 11 245 456,25 ], ossia circa un nono).

# Capitolo 4

# Free Riding e Degree del Nodo

Visto il successo riscontrato nella definizione di un meccanismo adattivo che leghi la condizione di free rider di un nodo destinatario al time-to-live (TTL) di ogni messaggio ricevuto, in questo nuovo capitolo tratteremo la definizione di un altro sistema per la scelta di free rider in una rete peer-to-peer non governato unicamente da una costante fissata a priori dall'utilizzatore del simulatore. Tuttavia in questo caso il fattore che concorrerà a definire la nuova soglia di probabilità non sarà relativa al messaggio da recapitare, bensì al nodo destinatario stesso. Tale fattore è il *degree* di un nodo all'interno del grafo.

## 4.1 Definizione del meccanismo adattivo

Come già menzionato nel capitolo 2, il degree di un nodo in un grafo non direzionato è definito come il numero dei suoi vicini, cioè i nodi in grado di comunicare direttamente con esso tramite vertici. Nel caso di grafi non direzionati è possibile distinguere fra out-degree ed in-degree, ovvero il numero di vertici uscenti ed entranti, ma nel contesto di questa tesi ci si concentrerà solo sulla sua variante per grafi non direzionati. Una proprietà di un grafo molto importante per questo ambito di studio è la *distribuzione del degree* [xxvi], cioè la probabilità di distribuzione dei degree all'interno del grafo. Infatti, se ogni vertice del grafo ha lo stesso degree $k$ il grafo è detto *k-regular*, una categoria alla quale si è già fatto riferimento nel capitolo 1 quando si è affrontato l'argomento delle topologie di grafo che LUNES è in grado di simulare. Ma tutte le simulazioni condotte finora per studiare il fenomeno del free riding hanno usato come punto di partenza i grafi aleatori, i quali invece non godono di tale caratteristica. Per via del meccanismo casuale che governa l'instaurazione di connessioni tra nodi, è facilmente comprensibile che non c'è modo di predire il numero di vicini di ogni singolo nodo prima della generazione del grafo.

A riguardo della distribuzione del degree all'interno dei grafi aleatori è tuttavia possibile fare una stima qualitativa: è molto più probabile ipotizzare che un nodo al centro del grafo possegga un degree superiore a quello di un suo pari collocato invece nelle zone periferiche che il contrario. Per quanto il sistema di generazione delle connessioni sia randomico, in un dato intorno il primo ha generalmente più simili con i quali instaurare collegamenti rispetto al secondo e quindi più candidati a diventare suoi "vicini". Se ricordiamo, nell'introduzione a questa trattazione viene specificato che in questo capitolo viene analizzato un meccanismo adattivo dove il processo per determinare se un nodo è o non è un free rider è dipendente dalla posizione di un nodo all'interno del grafo. Se l'andamento

generale della distribuzione del degree illustrato poche righe fa ci permette di avere un'idea generale della posizione di un nodo nel grafo (più precisamente, della sua distanza dal centro), il degree può diventare la base sulla quale realizzare l'algoritmo di questo nuovo meccanismo adattivo.

Ci viene a questo punto in aiuto l'algoritmo di disseminazione denominato *Degree-dependent Dynamic Gossip* (DDG), uno degli algoritmi implementati all'interno di LUNES ed il cui codice è quindi a completa disposizione degli utilizzatori. DDG cerca di sfruttare le caratteristiche degli overlay networks per far sì che i nodi raggiungano una sorta di "consapevolezza" della loro collocazione approssimativa all'interno del grado (in altre parole del loro degree) e dell'importanza del loro contributo alla disseminazione dei dati. Questa "consapevolezza" viene ottenuta costringendo ogni nodo a comunicare con i vicini informazioni sul proprio degree, così da regolare meglio la probabilità di disseminazione. [xxvii] Dopo la generazione di un nuovo messaggio, il nodo p computa il valore del suo degree e lo aggiunge al payload del messaggio - mostrato nella riga 2 dell'algoritmo 7 con la notazione del linguaggio C *struct* - e successivamente lo invia a tutti i suoi vicini.

**Algoritmo 7** Degree-dependent Dynamic Gossip: generazione di un nuovo messaggio eseguita da un nodo *p*

**Elementi:** variabile $\Lambda_p$ che contiene il numero di vicini del nodo *p* dei quali esso è a conoscenza.

1: *msg* = GenerateMessage()
2: *msg.degree* = Size($\Lambda_p$)
3: **for all** $n \in \Lambda_p$ **do**
4: Send(*msg*, *n*)
5: **end for**

Sulla base di questo algoritmo, è evidente che tutti le entità simulate che compongono i grafi contengono al loro interno proprio il dato che ci serve per far sì che questo nuovo meccanismo adattivo funzioni correttamente. I nodi sono infatti implementati come strutture dati (*struct*) al cui interno, precisamente nella parte dove è definito il loro stato, è presente una variabile *num_neighbors* che permette all'algoritmo DDG di funzionare.

Nel capitolo 3.1 è stato spiegato che la funzione lunes_real_forward() presente in `Lunes` racchiude il nucleo dei protocolli di disseminazione, poiché vi si può trovare l'implementazione delle regole di disseminazione di tali protocolli. È quindi naturale che nella parte dedicata a DDG il degree del nodo destinazione, cioè la variabile num_neighbors al suo interno, giochi un ruolo cruciale nell'ambito della valutazione probabilistica dei vicini ai quali inviare il ping. Per analogia, nella sezione dedicata alla disseminazione con probabilità fissa (l'algoritmo che costituisce il caso di studio di questa tesi) immediatamente prima della chiamata ad execute_ping(), si potrebbe andare ad inserire una chiamata ad una nuova versione di lunes_set_freeriding() nella quale, oltre al nodo destinazione, viene passato come parametro il degree del nodo mittente, come nella precedente implementazione veniva passato il time-to-live del messaggio da inviare.

A questo punto si potrebbe porre come obiezione a questo modo di procedere il fatto che, chiamando lunes_set_freeriding() in quel punto del codice, lo soglia adattiva per determinare se i vicini sono free rider o meno viene ricalcolata ad ogni inoltro di messaggio e non stabilita una volta sola come sarebbe lecito aspettarsi, generando una mole di operazioni assolutamente non necessaria. Purtroppo questa scelta si rivela essere paraticamente obbligata a causa del concetto fondamentale

alla base del Degree-dependent Dynamic Gossip: affinché i nodi raggiungano "consapevolezza" della propria posizione nel grafo, è necessario che questi si scambino informazioni sul proprio degree. La variabile num_neighbors non viene quindi inizializzata al momento della creazione dell'entità simulata al suo valore definitivo. Parte da un valore pari a 0 e viene poi incrementata di 1 ogni volta che "scopre" un nuovo vicino. Per un osservatore esterno, il degree di un nodo è una caratteristica immutabile per tutto il corso della simulazione, ma per un nodo è un attributo che subisce delle variazioni. Nell'implementazione del nuovo meccanismo adattivo, questo è un fatto di cui bisogna assolutamente tenere conto.

Un altro aspetto fondamentale che deve essere considerato è di natura, per così dire, topografica. In quale zona dell'overlay network desideriamo che siano maggiormente concentrati i nodi che si comportano come free rider? A seconda dell'implementazione dell'algoritmo, si può fare in modo che sia più probabile che un nodo trovi free rider all'altro capo delle connessioni nella zona centrale della rete (dove sono comuni nodi con degree alto) o nelle regioni periferiche (dove la maggior parte dei nodi ha degree basso). Nel contesto di questa trattazione si è scelta la prima tra la due opzioni. Nella fase di interpretazione dei risultati sarà ovviamente necessario tenere presente che è statisticamente più probabile incontrare free rider nelle zone più interne del grafo.

Rimane ora da determinare l'algoritmo del suddetto meccanismo e, alla luce di quanto spiegato nel paragrafo precedente, il primo passo in assoluto sarà approfondire i due valori di *num_neighbors* che meritano di essere approfonditi a parte rispetto a qualunque altra casistica: 0 e 1. Come è stato già illustrato nel paragrafo precedente, ogni simulazione inizia con il degree di ogni nodo impostato a 0, la cosiddetta fase di start-up, dovuta al fatto che nessuna informazione sui vicini è ancora stata scambiata. In questo caso l'algoritmo DDG è impostato per garantire la piena disseminazione, poiché in una situazione del genere è sensato che nessuno dei nodi vicini venga privato delle importanti informazioni veicolate nel payload dei messaggi, le quali permettono loro di giungere in fretta alla "consapevolezza" dei propri degree. Per la medesima ragione sarà opportuno che nella fase di start-up nessun nodo venga considerato un free rider. Nel caso il nodo invece abbia num_neighbors uguale a 1, questa condizione è raggiungibile (a) temporaneamente da un nodo che abbia in realtà più di un vicino ma che ha ricevuto informazioni solo da uno di loro e (b) fino al termine della simulazione da un nodo che abbia effettivamente un solo vicino e pertanto è collocato (probabilmente) all'estrema periferia del grafo. Nel primo caso è consigliabile che il nodo non abbia free rider tra i suoi vicini, dal momento che conviene che essi adeguino il loro degree alla situazione reale. Nel secondo è indifferente che l'unico vicino sia un free rider o meno, poiché il nodo potrebbe ri-inoltrare i messaggi ricevuti dal vicino solo a quest'ultimo e ciò va contro le regole della disseminazione. Anche in questo caso il nodo non avrà free rider tra i vicini.

Nell'algoritmo 8 riportato di seguito è possibile vedere il trattamento riservato ai valori speciali di num_neighbors e in più la procedura generale da adottare per qualunque altro valore. Essa consiste nel calcolare la nuova soglia di probabilità adattiva (dipendente dal degree) $t$ come l'inverso del degree $d$ elevato ad una potenza pari alla soglia di probabilità stabile $c$, in altre parole $t = 1 / (d \hat{} c)$. Una volta ottenuto questo nuovo numero, si confronta ad esso il valore casuale compreso tra 0 e 1 generato dall'apposita funzione e, se quest'ultimo è al di sotto della soglia, il nodo destinatario è un free rider, altrimenti non lo è.

**Algoritmo 8** Meccanismo di determinazione dei free rider a soglia di probabilità adattiva rispetto al degree del nodo mittente

**Elementi:** il nodo (*node*) destinatario di un messaggio passato come argomento a lunes_set_freeriding(), la variabile d'ambiente *environment_threshold* che esprime la probabilità impostata dall'utente che un nodo sia un free rider ed il *degree* del nodo mittente, cioè il numero di vicini dei quali il nodo esso è a conoscenza.

```
1:  if environment_threshold > 0 then
2:      freerider ← Generate_random_double(0, 1)
3:      if degree = 0 or degree = 1 then
4:          node non è un free rider
5:      else
6:          adaptive_threshold ← 1 / degree^environment_threshold
7:          if freerider >= adaptive_threshold then
8:              node è un free rider
9:          else
10:             node non è un free rider
11:         end if
12:     end if
13: end if
```

Il motivo che sta dietro la necessità di adottare una formula che impiega l'operazione di potenza per calcolare la soglia adattiva è presto detto. Se nel meccanismo adattivo dipendente dal TTL calcolare un fattore moltiplicativo compreso tra 0 e 1 era relativamente semplice poiché abbiamo a disposizione una variabile d'ambiente (modificabile dall'utente) che esprime un limite massimo al valore di TTL, questa situazione non si ripete nel caso del degree. Non essendo possibile calcolare un rapporto tra il valore corrente di num_neighbors ed un suo ipotetico limite massimo, è necessario adottare un'altra strategia e quella individuata fa sì di ottenere un risultato sempre appartenente al range prefissato. Poiché essa ha l'effetto di ottenere un valore di soglia adattiva tanto più tendente a 1 quanto più il valore di soglia stabile è tendente a 0, ricordando l'intenzione di sperimentare il meccanismo adattivo in un contesto dove i nodi free rider sono maggiormente concentrati al centro della rete, sarà necessario usare la nuova soglia come limite inferiore (e non superiore, come nel caso dipendente dal time-to-live) per l'assegnamento di un nodo come free rider.

Nelle tabelle riportate nelle pagine successive, divisi per valore della variabile d'ambiente espressione della soglia di probabilità stabile, viene illustrato come varia la soglia adattiva (arrotondata alla terza cifra decimale) a seconda del degree del nodo. Per ragioni di spazio, sono stati presi in considerazione solo i valori di *num_neighbors* fino a 6. 0 e 1 sono esclusi in quanto casi speciali. Queste tabelle ci saranno utili nella fase di interpretazione dei risultati dei test.

| Degree del nodo | Soglia adattiva per quel nodo |
|---|---|
| 2 | 0,933 |
| 3 | 0,896 |
| 4 | 0,871 |
| 5 | 0,851 |
| 6 | 0,836 |

Tabella 1 Soglia di probabilità adattiva in relazione al degree di un nodo per FREERIDER = 0,1.

| Degree del nodo | Soglia adattiva per quel nodo |
|---|---|
| 2 | 0,871 |
| 3 | 0,803 |
| 4 | 0,758 |
| 5 | 0,725 |
| 6 | 0,7 |

Tabella 2 Soglia di probabilità adattiva in relazione al degree di un nodo per FREERIDER = 0,2.

| Degree del nodo | Soglia adattiva per quel nodo |
|---|---|
| 2 | 0,812 |
| 3 | 0,719 |
| 4 | 0,66 |
| 5 | 0,617 |
| 6 | 0,584 |

Tabella 3 Soglia di probabilità adattiva in relazione al degree di un nodo per FREERIDER = 0,3.

| Degree del nodo | Soglia adattiva per quel nodo |
|---|---|
| 2 | 0,757 |
| 3 | 0,644 |
| 4 | 0,574 |
| 5 | 0,525 |
| 6 | 0,488 |

Tabella 4 Soglia di probabilità adattiva in relazione al degree di un nodo per FREERIDER = 0,4.

| Degree del nodo | Soglia adattiva per quel nodo |
|---|---|
| 2 | 0,707 |
| 3 | 0,577 |
| 4 | 0,5 |
| 5 | 0,447 |
| 6 | 0,408 |

Tabella 5 Soglia di probabilità adattiva in relazione al degree di un nodo per FREERIDER = 0,5.

| Degree del nodo | Soglia adattiva per quel nodo |
|---|---|
| 2 | 0,66 |
| 3 | 0,517 |
| 4 | 0,435 |
| 5 | 0,381 |
| 6 | 0,341 |

Tabella 6 Soglia di probabilità adattiva in relazione al degree di un nodo per FREERIDER = 0,6.

| Degree del nodo | Soglia adattiva per quel nodo |
|---|---|
| 2 | 0,616 |
| 3 | 0,463 |
| 4 | 0,379 |
| 5 | 0,324 |
| 6 | 0,285 |

Tabella 7 Soglia di probabilità adattiva in relazione al degree di un nodo per FREERIDER = 0,7.

| Degree del nodo | Soglia adattiva per quel nodo |
|---|---|
| 2 | 0,574 |
| 3 | 0,415 |
| 4 | 0,33 |
| 5 | 0,276 |
| 6 | 0,238 |

Tabella 8 Soglia di probabilità adattiva in relazione al degree di un nodo per FREERIDER = 0,8.

| Degree del nodo | Soglia adattiva per quel nodo |
|---|---|
| 2 | 0,536 |
| 3 | 0,372 |
| 4 | 0,287 |
| 5 | 0,235 |
| 6 | 0,199 |

Tabella 9 Soglia di probabilità adattiva in relazione al degree di un nodo per FREERIDER = 0,9.

E adesso che l'implementazione di questo secondo sistema adattivo è completa, possiamo cominciare la fase di test in cui ne valuteremo le performance.

## 4.2 Fase di simulazione

Giunti a questo punto vale la pena ricordare i parametri delle simulazioni i cui risultati ci apprestiamo a visualizzare: tutte le simulazioni qui illustrate sono state realizzate su reti a grafo aleatorio, dove cioè le connessioni tra i nodi sono generate randomicamente. Tutti i grafi che vedremo in questo e nel prossimo capitolo contano al loro interno 500 nodi e 1500 connessioni. Il diametro massimo del grafo è 7. Il time-to-live massimo di un messaggio è fissato a 10 *hop*. La cache di ogni singolo nodo è fissata a 256 elementi. Ogni grafico riportato di seguito è del tipo già utilizzato nel capitolo 2, con la misurazione del rapporto di copertura (in percentuale) sull'asse delle ordinate e l'overhead ratio (che funge anche da misura indiretta del numero di messaggi veicolati dal sistema) sull'asse delle ascisse. L'algoritmo di disseminazione dei messaggi è sempre quello a probabilità fissa.

Come già visto nel capitolo 3.2, in tutti i grafici riportati di seguito è possibile trovare due insiemi disgiunti di dati: quelli raccolti a seguito di un insieme di simulazioni che sfruttano il meccanismo stabile e quelli raccolti a seguito di un insieme di simulazioni che sfruttano il meccanismo dipendente dal degree di un nodo, entrambi con riferimento al medesimo valore della variabile d'ambiente FREERIDER. Si inizia con un valore di FREERIDER impostato a 0,1 (immagine 4.1) e si continua incrementando di volta in volta di 0,1 fino a quando la variabile non è uguale a 0,9 (immagine 4.9).

È possibile osservare anche nel caso di questo secondo meccanismo adattivo come la "curva" formata dai risultati segnati nel diagramma con un asterisco rosso registri uno scostamento rispetto alla "curva" blu e anche in questo caso ciò è dovuto ad una pendenza maggiore della seconda rispetto alla prima nella fase in cui il rapporto di copertura cresce velocemente. Sono tuttavia presenti due importanti differenze rispetto al caso di studio precedente, che ora andremo ad esaminare una alla volta.

In primo luogo, diversamente dal caso dipendente dal time-to-live, il rapporto di copertura questo meccanismo adattivo con probabilità di disseminazione pari al 100% rimane minore del corrispettivo facente capo al meccanismo stabile fino ad un certo valore di FREERIDER, superato il quale la dinamica si inverte. Basta mettere a confronto le figure 4.7 e 4.8 per rendersi conto dell'inversione di tendenza. Fino ad un valore di FREERIDER pari a 0,7 il delta fra il valore di ordinata massimo raggiunto dalla curva blu e quello raggiunto dalla curva rossa rimane positivo e anzi aumenta [ +0,176 +0,119 +0,431 +2,005 +3,618 +7,732 +11,767 ], al punto che nell'immagine 4.7 il primo è pari approssimativamente al [ 120% ] del secondo. Superata questa soglia, il delta cambia segno in negativo [ -11,247 ] e rimane più o meno stabile in valore assoluto nell'immagine 4.9 [ -11,605 ]. Quello che cambia è la proporzione tra il rapporto di copertura massimo raggiungibile dal meccanismo stabile e quello raggiungibile dal sistema adattivo dipendente dal degree: nell'immagine 4.8 il primo è pari circa al [ 55% ] del secondo e passa ad essere appena il [ 18% ] nell'immagine 4.9.

La seconda differenza riguarda i valori massimi di overhead ratio raggiungibili dalle due "curve". Se nel caso del meccanismo adattivo dipendente dal TTL non capita mai (almeno tra i casi esaminati) che l'overhead ratio del meccanismo adattivo per una probabilità di disseminazione pari al 100% scenda al di sotto della sua controparte relativa al meccanismo stabile, in questo secondo caso di studio questa regola non è valida. Come è ben visibile nell'immagine 4.7, per FREERIDER=0,7

anche l'ascissa massima raggiungibile dalla "curva" rossa è inferiore all'ascissa massima raggiungibile dalla "curva" blu. Esaminando i diagrammi a partire dal primo, è possibile vedere come il massimo valore di overhead ratio del meccanismo adattivo è anche qui come in 3.2 in continuo decremento a partire dal valore massimo registrato per FREERIDER=0,1 [ 4,947 ], ma costituisce in proporzione un incremento rispetto al corrispettivo del meccanismo stabile sempre più consistente fino a FREERIDER=0,5 [ +9% +10,45% +33,47% +41,836% +43,41% ]. Oltre questa soglia, come visibile nell'immagine 4.6, la differenza tra i due massimi passa ad essere molto più contenuta (l'ascissa massima della curva rossa è solo il [ 120% ] di quella della curva blu) e per FREERIDER pari a 0,7 il rapporto addirittura si inverte, portando il massimo valore di overhead ratio del meccanismo stabile ad essere superiore di quello del meccanismo adattivo. Per i valori 0,8 e 0,9 di FREERIDER si nota invece un ritorno ad una situazione più consueta e parallela a quella già esaminata nel paragrafo precedente, con l'ascissa massima della curva blu pari circa al [ 43% ] di quella della curva rossa per il primo valore, rapporto che scende ad appena il [ 17% ] per il secondo.

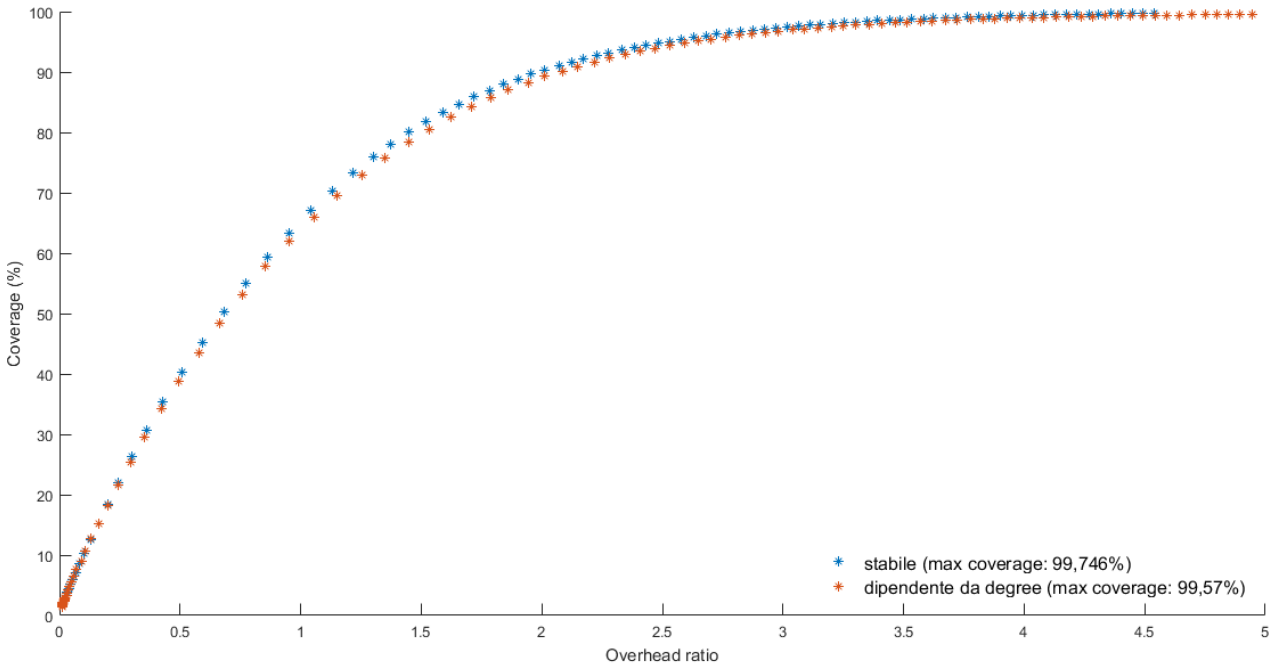



Immagine 4.1 Confronto tra meccanismo stabile e meccanismo dipendente da degree, FREERIDER=0,1.

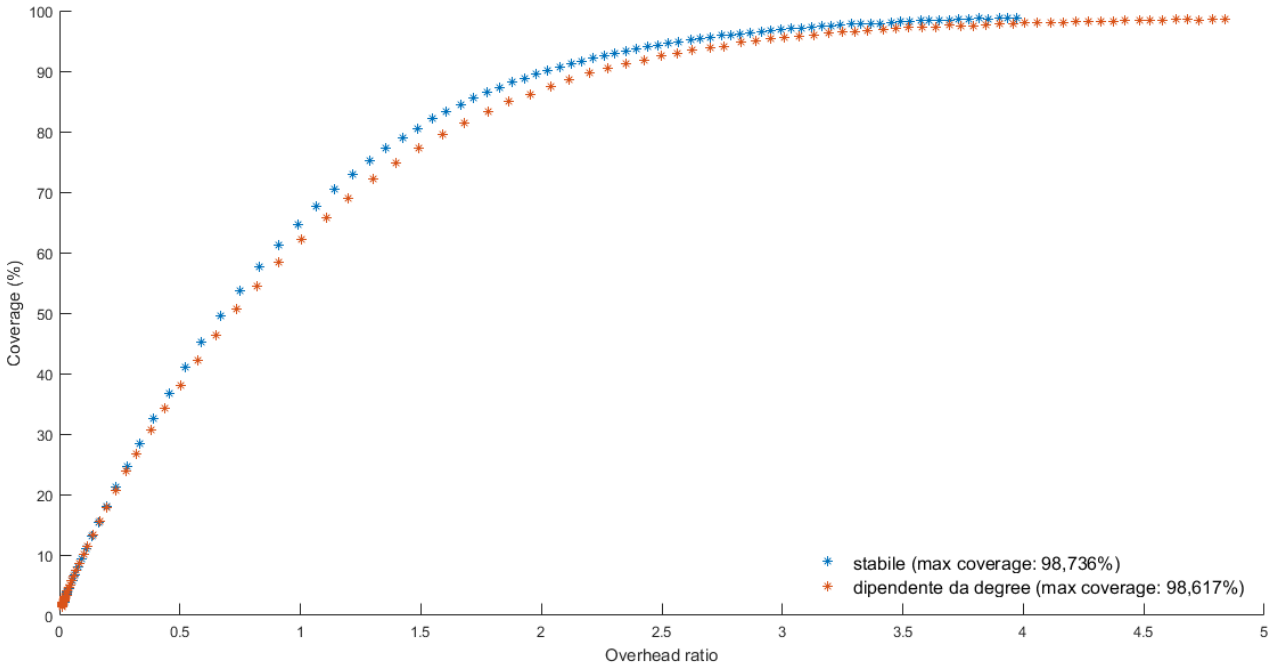



Immagine 4.2 Confronto tra meccanismo stabile e meccanismo dipendente da degree, FREERIDER=0,2.

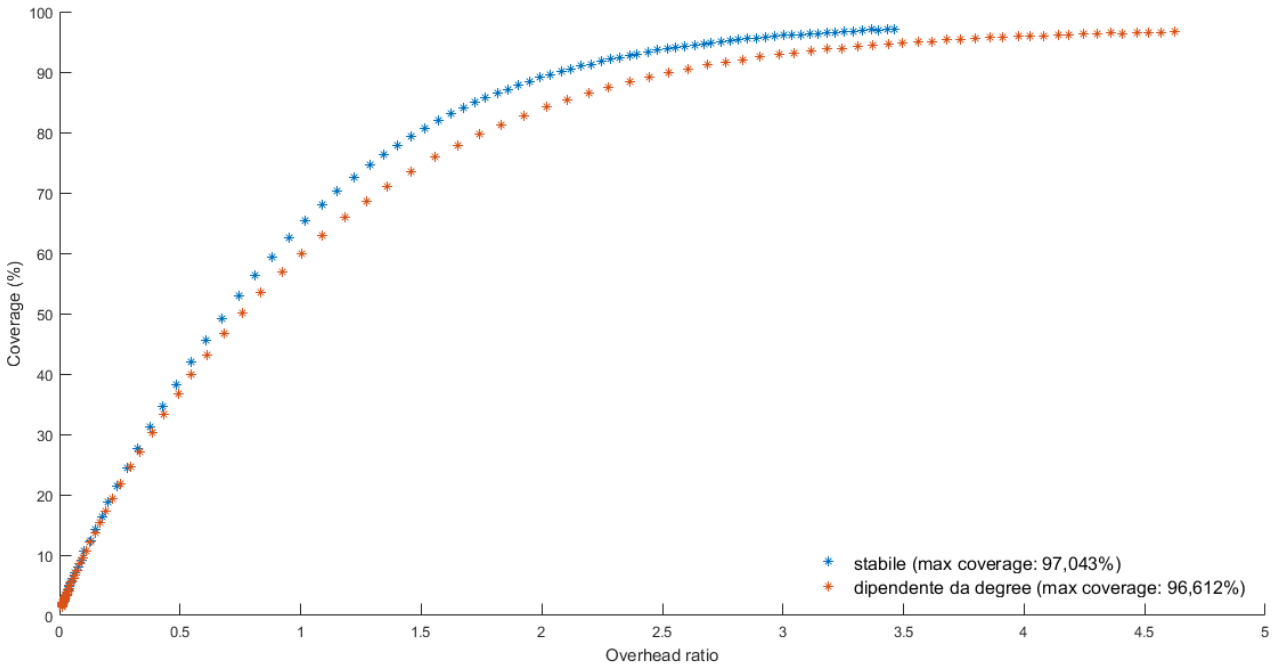



Immagine 4.3 Confronto tra meccanismo stabile e meccanismo dipendente da degree, FREERIDER=0,3.

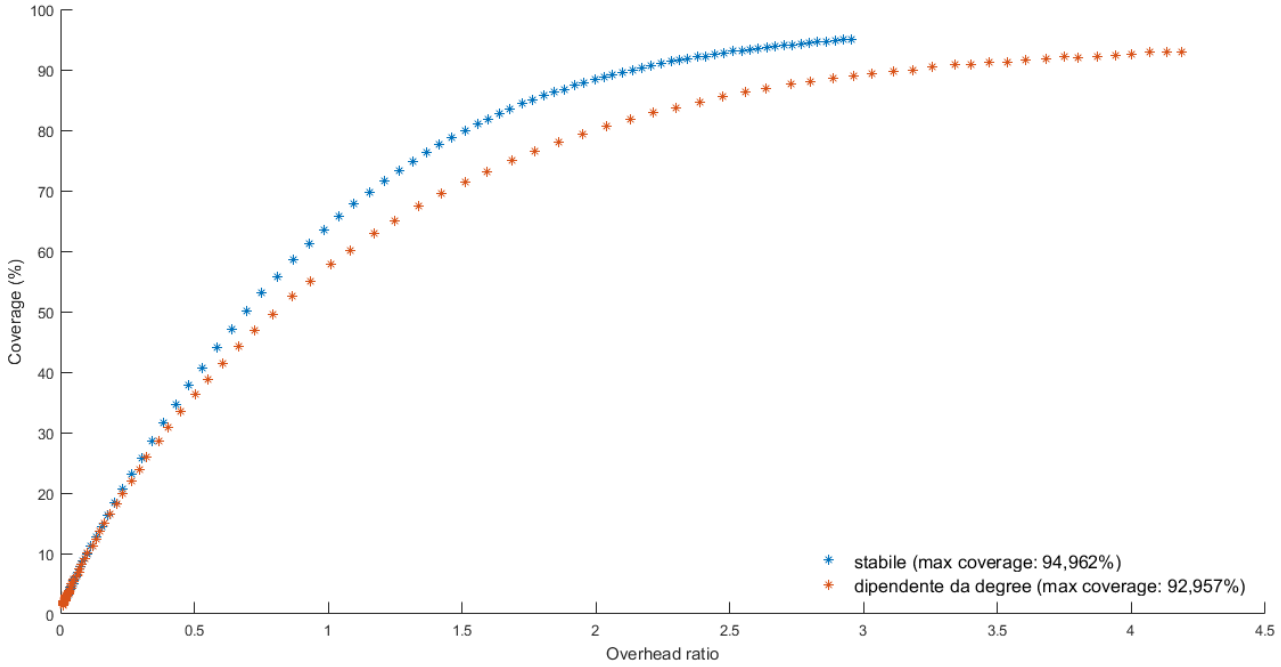



Immagine 4.4 Confronto tra meccanismo stabile e meccanismo dipendente da degree, FREERIDER=0,4.

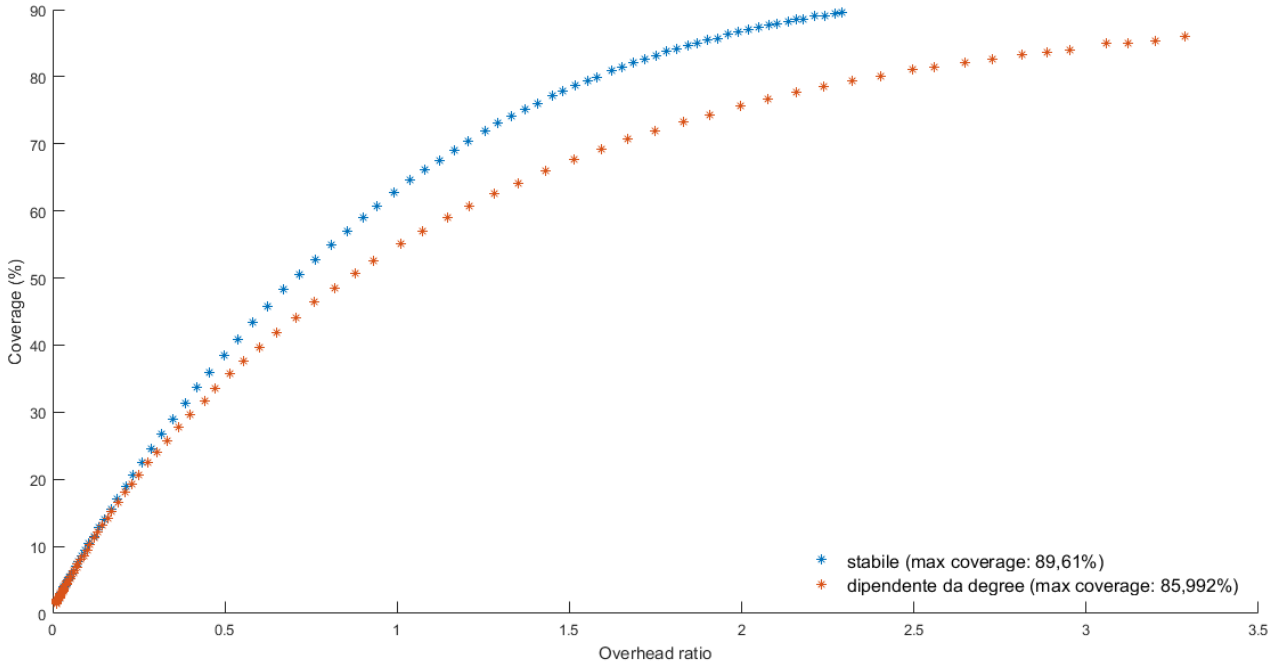



Immagine 4.5 Confronto tra meccanismo stabile e meccanismo dipendente da degree, FREERIDER=0,5.

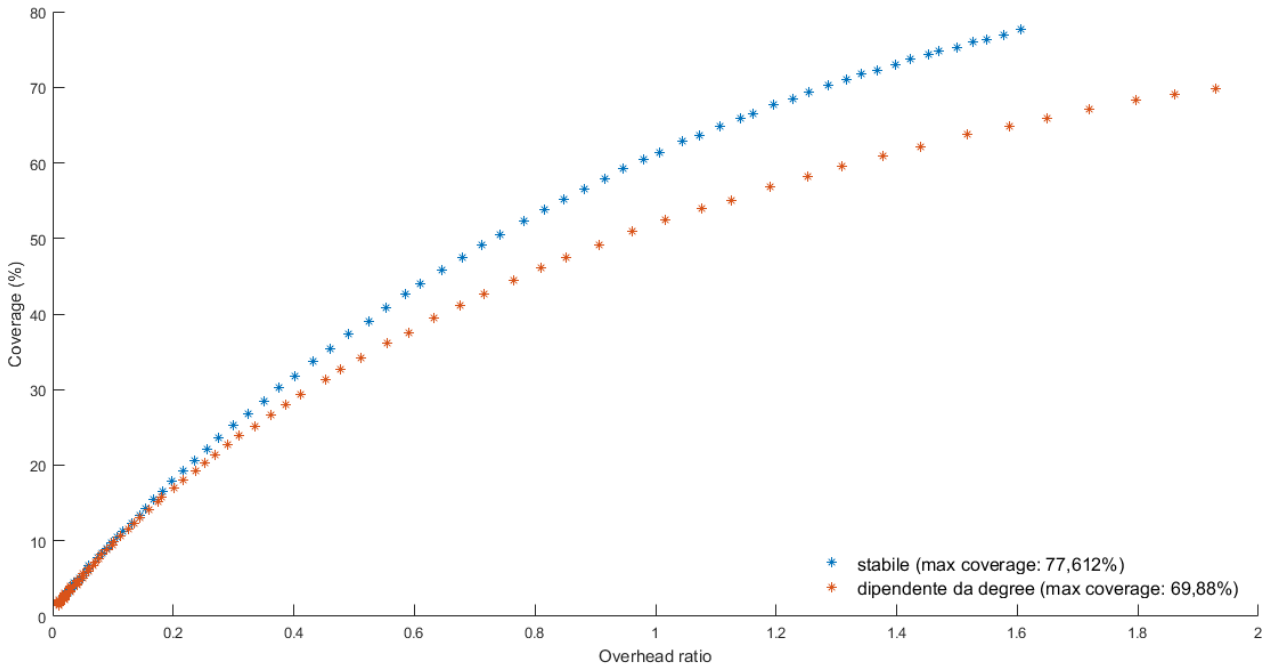



Immagine 4.6 Confronto tra meccanismo stabile e meccanismo dipendente da degree, FREERIDER=0,6.

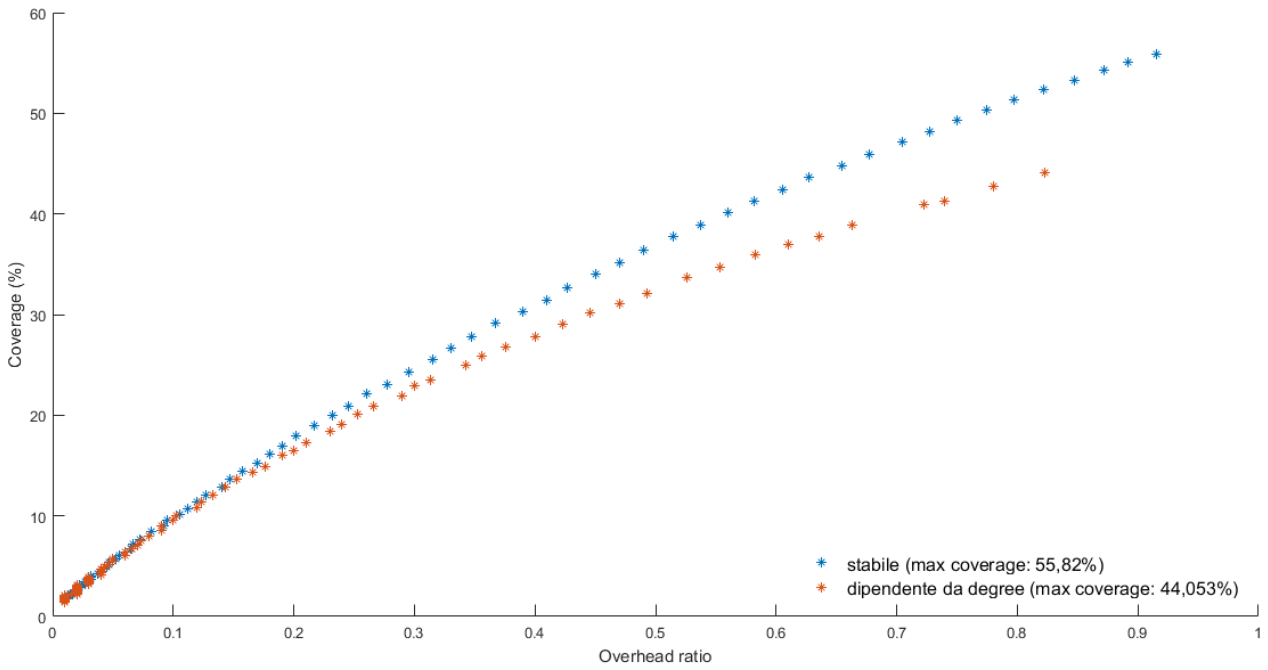



Immagine 4.7 Confronto tra meccanismo stabile e meccanismo dipendente da degree, FREERIDER=0,7.

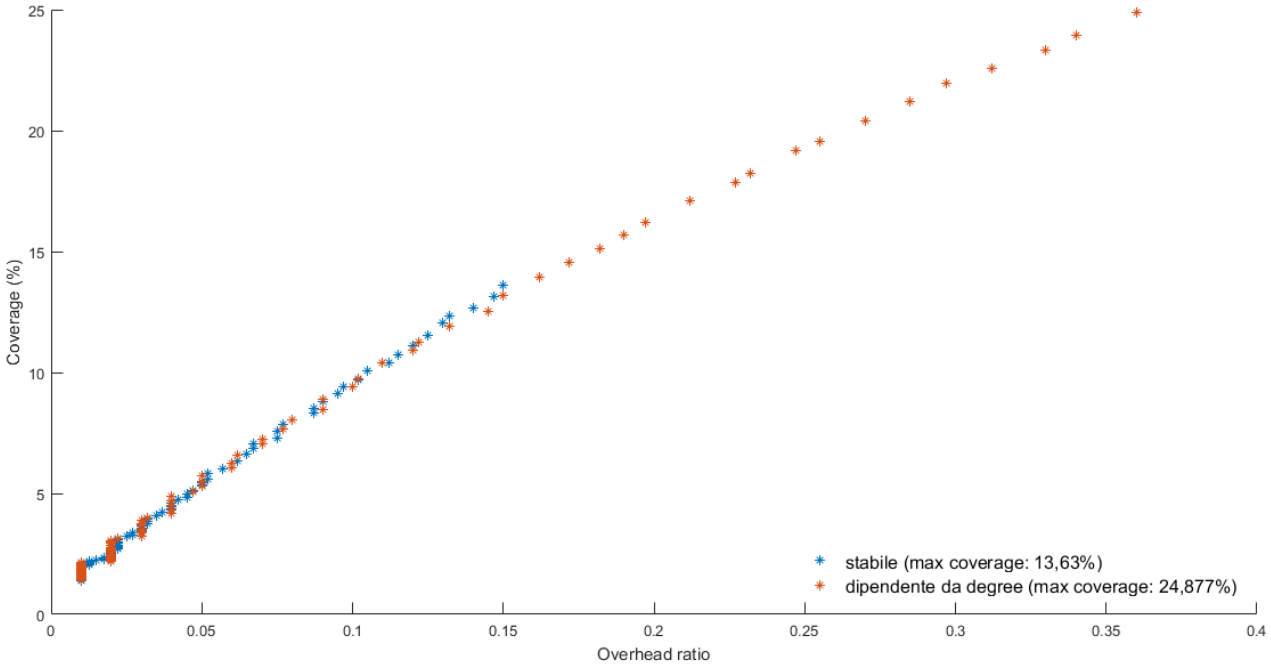



Immagine 4.8 Confronto tra meccanismo stabile e meccanismo dipendente da degree, FREERIDER=0,8.

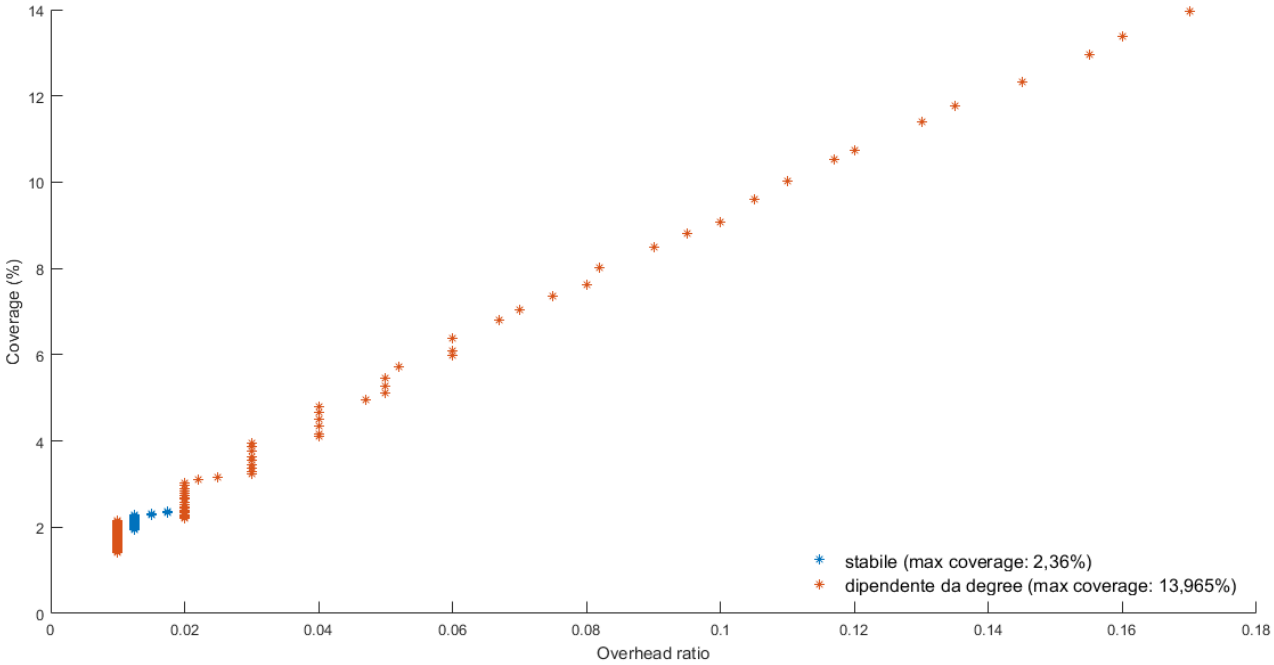



Immagine 4.9 Confronto tra meccanismo stabile e meccanismo dipendente da degree, FREERIDER=0,9.

## 4.3 Interpretazione dei risultati

Riprendendo la domanda posta nell'introduzione al capitolo 3.3, i risultati dei test svolti sul nuovo meccanismo adattivo sono sensati? Certamente, rendendo la determinazione dei free rider dipendente dal degree dei nodi abbiamo ottenuto dati dalle simulazioni che testimoniano trend in certa misura meno omogenei rispetto al precedente meccanismo adattivo, dove il comportamento della curva rossa rispetto alla curva blue era pressoché riconducibile allo stesso schema per ogni valore di FREERIDER. Tuttavia, l'assenza nei grafici riportati nelle immagini da 4.1 a 4.9 di andamenti totalmente controintuitivi (come potrebbe essere, ad esempio, un'insieme di dati che parte da un'alta percentuale di copertura per una probabilità di disseminazione dell'1% e tende poi a decrescere) o di elementi in posizioni inaspettate sul piano cartesiano fornisce una ragione più che valida per sospettare che alcuni sorprendenti comportamenti del meccanismo adattivo sotto esame siano dovute a peculiari dinamiche insite nel suo funzionamento e non ad una fallace implementazione dello stesso. I risultati che andremo ad analizzare nel proseguio di questo capitolo possono quindi ritenersi sensati.

Procedendo con le analisi dei trend individuati, quello che più di tutti merita un approfondimento è quello relativo alle percentuali massime di copertura dei grafi, che come visto nel capitolo 4.2 vede in vantaggio il meccanismo stabile fino ad un valore di FREERIDER (ricordiamo, tra quelli esaminati) pari a 0,7 per poi subire un'inversione di ruoli, con il meccanismo adattivo che si rivela molto più performante dello stabile per valori maggiori della variabile d'ambiente. Questo curioso comportamento in realtà è dovuto alla tendenza all'abbattimento delle performance che è comune ad entrambi i meccanismi (oltre a quello adattivo dipendente dal TTL), ma subisce un crollo molto più repentino tra FREERIDER=0,7 e FREERIDER=0,8 rispetto alla controparte adattiva, la quale di conseguenza riesce ad effettuare il "sorpasso". Nell'immagine 4.10 viene offerta una rappresentazione visiva di questi andamenti così da poter apprezzare a colpo d'occhio le dinamiche appena descritte.

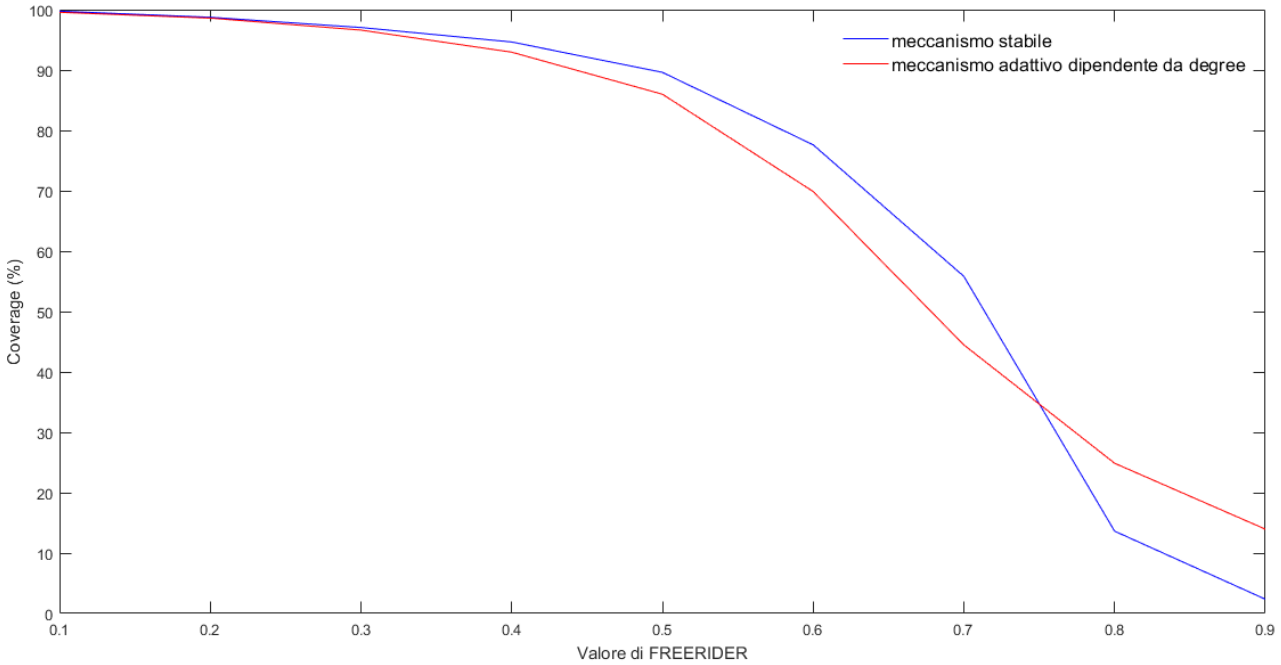



Immagine 4.10 Confronto dei valori di coverage per possibilità di disseminazione pari al 100% tra meccansmi.

Se si vanno ad osservare i dati riportati nelle tabelle al capitolo 4.1, è possibile però giungere alla conclusione che questo andamento è in larga parte giustificato dall'algoritmo che implementa il

meccanismo adattivo.

Prendendo in esame la tabella 1, noteremo che per FREERIDER=0,1 la probabilità di avere free rider è minore rispetto a quella del meccanismo a soglia fissa per nodi che abbiano degree pari a 2, mentre è maggiore per tutti i valori al di sopra di esso. Ciò aiuta la propagazione di messaggi sia nella fase di start-up che nelle zone periferiche della rete, dove nodi con 1 o 2 vicini (riconosciuti, nel primo caso) sono più comuni. Questo ovviamente ha ripercussioni sul parametro di overhead ratio, che infatti raggiunge livelli maggiori rispetto al meccanismo stabile. Una volta però che la fase di start-up è terminata e i nodi al centro della rete sono giunti ad una migliore "consapevolezza" della propria posizione e perciò del proprio numero di vicini, la probabilità che questi siano determinati come free rider al recepimento di un messaggio aumenta, bilanciando gli effetti positivi sul rapporto di copertura che un maggior numero di messaggi circolanti apporterebbe. Infatti, come è possibile vedere nella figura 4.1, lo scostamento tra i due insiemi di dati è minimo lungo tutto il cammino delle "curve" che descrivono ma, prendendo un qualunque valore di probabilità di disseminazione, il rapporto di copertura ad esso associato sarà sempre e comunque minore, anche se di poco, nel caso del meccanismo adattivo dipendente dal degree rispetto al corrispettivo del meccanismo stabile.

I fattori che causano il comportamento illustrato nel paragrafo precedente sono molto probabilmente i responsabili anche dell'abbattimento più rapido delle prestazioni del meccanismo adattivo rispetto a quello stabile, che ha come corollario una precipitosa diminuzione del volume di messaggi veicolato dal sistema osservabile nelle immagini dalla 4.4 alla 4.7. La presenza di un numero importante di free rider concentrati nelle zone centrali dell'overlay network deve essere quindi particolarmente di disturbo per l'obiettivo di ottenere alti rapporti di copertura, molto più che averne un numero paragonabile ma distribuito uniformemente su tutto il grafo. Tuttavia, come anticipato in 4.2, esiste una particolare casisitica nella quale le performance del meccanismo adattivo dipendente dal degree sono considerevolmente migliori di quelle del meccanismo stabile, dettaglio questo che rende la nostra formulazione algoritmica comunque degna di nota.

I valori soglia riportati nelle tabelle 8 e 9 mostrano chiaramente come il degree di un nodo debba crescere notevolmente (specialmente nel caso di FREERIDER=0,9) perché i suoi vicini abbiano la medesima probabilità di essere impostati come free rider che avrebbero in una simulazione effettuata impiegando il meccanismo stabile. Semplicemente, per come è stato impostato il calcolo del fattore moltiplicativo (algoritmo 4, riga 7), utilizzando questo secondo meccanismo adattivo nei test, troveremo in generale meno nodi che si comportano come free rider per valori della variabile d'ambiente tendenti a 1, con tutti i vantaggi che questo comporta per quanto riguarda i livelli massimi di copertura raggiungibili.

# Capitolo 5

# Free Riding, Time-to-Live e Degree del Nodo

Giunti a questo punto, con due meccanismi adattivi sviluppati e funzionanti che collegano il processo di determinazione dei free rider a caratteristiche proprie dei nodi o dei messaggi da recapitare, potrebbe essere interessante studiare cosa succede in presenza contemporanea di entrambi. Questo capitolo della trattazione sarà quindi dedicato allo sviluppo di un terzo meccanismo che ha come scopo quello di produrre nuove soglie probabilistiche adattive dipendenti sia dal TTL del messaggio in procinto di essere spedito che dal numero di vicini del nodo mittente.

## 5.1 Definizione del meccanismo adattivo

A seguito dei risultati raggiunti nei capitoli 3 e 4, sviluppare un meccanismo adattivo che tenga in considerazione sia l'attributo di time-to-live del messaggio da inviare sia il degree del nodo mittente non è un compito complesso. Ancora più intuitiva risulta essere la posizione nel codice nella quale la nuova versione di lunes_set_freeriding() dovrà essere richiamata. I due meccanismi visti finora sono stati implementati all'interno di funzioni richiamate entrambe esattamente nello stesso punto del codice, cioè all'interno di lunes_real_forward(), immediatamente prima della chiamata a execute_ping(). Ha senso che il nuovo lunes_set_freeriding() per il meccanismo adattivo ibrido sia richiamato sempre in quel punto, stavolta passando tre argomenti invece di due: un riferimento al nodo destinatario, il TTL rimanente del messaggio che deve essere inoltrato ed il numero di vicini del nodo mittente.

Il lavoro di implementazione vero e proprio si riduce in pratica all'operare una fusione tra i due algoritmi preesistenti in maniera che non si crei conflitto tra loro. Nel meccanismo adattivo dipendente da degree (algoritmo 8, capitolo 4), la nuova soglia di probabilità di incontrare un free rider è calcolata come l'inverso del degree elevato ad una potenza pari alla probabilità impostata dall'utente, tranne per due valori speciali di degree (0 e 1) dove si presume che il nodo all'altra estremità del vertice non sia mai un free rider. Nel meccanismo adattivo dipendente da TTL (algoritmo 6, capitolo 3), il valore di TTL viene in tutti i casi utilizzato come base per calcolare un fattore moltiplicativo con il quale giungere alla nuova soglia di probabilità adattiva. La soluzione proposta in questa trattazione è riportata nell'algoritmo 9.

Si calcola prima di tutto il fattore moltiplicativo relativo al time-to-live del messaggio (righe

3-4) che troverà comunque un utilizzo qualunque sia il degree del nodo. In seguito si aprirà un'istruzione condizionale riservata ai casi speciali di degree (riga 5) nella quale la soglia adattiva sarà calcolata con il solo apporto del fattore moltiplicativo (riga 6) e la determinazione del comportamento da free rider sarà effettuata con il solito sistema della comparazione (riga 7) tra soglia e numero casuale generato per l'occorrenza (riga 2). Per tutti gli altri valori di degree sarà invece calcolata una prima soglia adattiva (riga 13) come l'inverso dell'elevamento a potenza che abbiamo già visto nell'algoritmo 8 e poi una seconda moltiplicando la prima per il fattore dipendente dal TTL (riga 14). Il numero così ottenuto è quello effettivamente utilizzato per il confronto con il numero casuale (riga 15). L'operazione di confronto presenta, come nell'algoritmo 8, un segno di maggiore uguale data la nostra intenzione di studiare overlay network dove la popolazione di free rider è maggiormente concentrata al centro del grafo.

**Algoritmo 9** Meccanismo di determinazione dei free rider a soglia di probabilità adattiva rispetto al degree del nodo mittente e al time-to-live del messaggio da inviare

**Elementi:** il nodo (*node*) destinatario di un messaggio passato come argomento a lunes_set_freeriding(), la variabile d'ambiente *environment_threshold* che esprime la probabilità impostata dall'utente che un nodo sia un free rider, il *degree* del nodo mittente, cioè il numero di vicini dei quali il nodo esso è a conoscenza, il *ttl* del messaggio che sta per essere ricevuto da *node* ed il valore massimo del TTL di un generico messaggio (*max_ttl*).

```
1:    if environment_threshold > 0 then
2:        freerider ← Generate_random_double(0, 1)
3:        delta ← ( max_ttl - ttl ) / max_ttl
4:        factor ← 1 - delta
5:        if degree = 0 or degree = 1 then
6:            adaptive_threshold ← environment_threshold * factor
7:            if freerider <= adaptive_threshold then
8:                node è un freerider
9:            else
10:               node non è un freerider
11:           end if
12:       else
13:           adaptive_threshold ← 1 / degree^environment_threshold
14:           adaptive_threshold_2 ← adaptive_threshold * factor
15:           if freerider >= adaptive_threshold_2 then
16:               node è un free rider
17:           else
18:               node non è un free rider
19:           end if
20:       end if
21:   end if
```

Rispetto al meccanismo adattivo dipendente solo dal degree, non sono presenti casi nei quali l'algoritmo prevede che un nodo non sia un free rider qualunque sia il valore della variabile d'ambiente impostata dall'utente (FREERIDER). Invece, considerando il range del fattore moltiplicativo derivante dal TTL, per degree del nodo pari a 0 o 1 la probabilità di incontrare free rider all'altro capo dei collegamenti è sempre minore o uguale al valore di FREERIDER. Per quanto riguarda tutti

i valori di degree diversi da 0 e 1, il meccanismo di determinazione della nuova soglia di probabilità è effettivamente ibrido, poiché sfrutta entrambe le operazioni matematiche sulle quali si basano gli algoritmi 6 e 8. In base a questo, la soglia risultante sarà sempre inferiore o uguale alla medesima calcolata seguendo l'algoritmo 8 e questo comporta che in genere ci sia maggiore probabilità che il messaggio da inviare venga recapitato ad un free rider. Si rimanda alle tabelle 1-9 riportate nel capitolo 4.1 come riferimenti per avere un'idea generale dei valori di soglia adattiva risultato dell'operazione matematica in riga 13.

Nella fase di test che ci apprestiamo ad iniziare, vedremo le prestazioni di questo nuovo meccanismo adattivo ibrido a confronto con quelle dei due meccanismi di partenza.

## 5.2 Fase di simulazione

A differenza delle immagini riportate nei capitoli 3 e 4, i grafici rapporto di copertura/overhead ratio presenti in questa sezione riportano tre insiemi di dati invece di due: in blu l'andamento delle performance del meccanismo adattivo dipendente dal time-to-live del messaggio, in verde il corrispettivo riferito al meccanismo dipendente dal degree del nodo e la successione di asterischi rossi rappresentano le prestazioni del meccanismo ibrido per probabilità di disseminazione da 1% a 100%. Il caso di studio (topologia di grafo e protocollo di disseminazione) è ovviamente lo stesso dei capitoli precedenti.

Ciò che salta immediatamente all'occhio scorrendo le figure da 5.1 a 5.9 è il fatto che i risultati dei test sul meccanismo ibrido si adagiano su curve simil-logaritmiche che ricalcano fedelmente, al netto di scostamenti nell'ordine di grandezza di un errore di arrotondamento, le curve verdi. Per ogni valore della variabile di soglia di probabilità stabile che è stato preso in considerazione per le simulazioni, i risultati del meccanismo ibrido sono in tutto e per tutto assimilabili a quelli prodotti dal meccanismo adattivo dipendente dal degree. L'unica vera discriminante tra i due sistemi è il rapporto di copertura massimo (e di conseguenza l'overhead ratio massimo) che il meccanismo ibrido consente di raggiungere per probabilità di disseminazione del 100%.

Per valori di FREERIDER compresi tra 0,1 e 0,4, la copertura massima raggiunta in media alla fine di una simulazione in cui si sia sfruttato il meccanismo adattivo ibrido (che chiameremo *mch*) è assimilabile o persino lievemente superiore a quella per il meccanismo adattivo dipendente dal degree (*mcd*), con [ *mch* - *mcd* ] che si matiene su valori assoluti comunque estremamente ridotti [ -0,002  -0,035  +0,114  +0,023 ]. Passata questa soglia, [ *mch* - *mcd* ] si mantiene stabilmente su valori negativi e in diminuzione fino a FREERIDER=0,7 [ -1,112  -2,064  -3,193 ]. Passata questa ulteriore soglia, il valore assoluto di questa differenza cala [ -2,08  -1,475 ], ma il rapporto [ *mch* / *mcd* ] continua a diminuire [ 91,6%  89,4% ], a fronte di un [ 92,8% ] per FREERIDER=0,7.

Per FREERIDER=0,3 e FREERIDER=0,4 è possibile anche osservare come questi valori di copertura marginalmente migliori sono ottenuti con un minore overhead ratio rispetto al meccanismo dipendente da degree. La diminuzione dell'overhead ratio è un fenomeno che comincia a diventare veramente percepibile in figura 5.3 ed è presente da lì in poi per ogni valore della variabile d'ambiente sul quale siano state condotte simulazioni: la differenza tra *orh* (overhead ratio del meccanismo ibrido) e *ord* (overhead ratio del meccanismo adattivo dipendente da degree) rimane infatti negativa fino alla figura 5.9. [ *orh* - *ord* ] aumenta in valore assoluto fino a FREERIDER=0,5 compreso per poi diminuire [ -0,009  -0,081  -0,211  -0,19  -0,11  -0,048  -0,023 ]. Anche qui come nel paragrafo prece-

dente, il fatto che questa differenza cresca in valore assoluto fino a FREERIDER=0,5 per poi calare è bilanciata dal fatto che il rapporto [ *orh* / *ord* ] continui invece a crescere con l'aumentare del valore di FREERIDER. In figura 5.5, dove la differenza è massima in valore assoluto, il rapporto ci dice che orh è il [ 93,6% ] di *ord*. In figura 5.9, dove la differenza è poco più di un decimo della corrispettiva nell'esempio precedente, *orh* è l'[ 86,5% ] di *ord*.

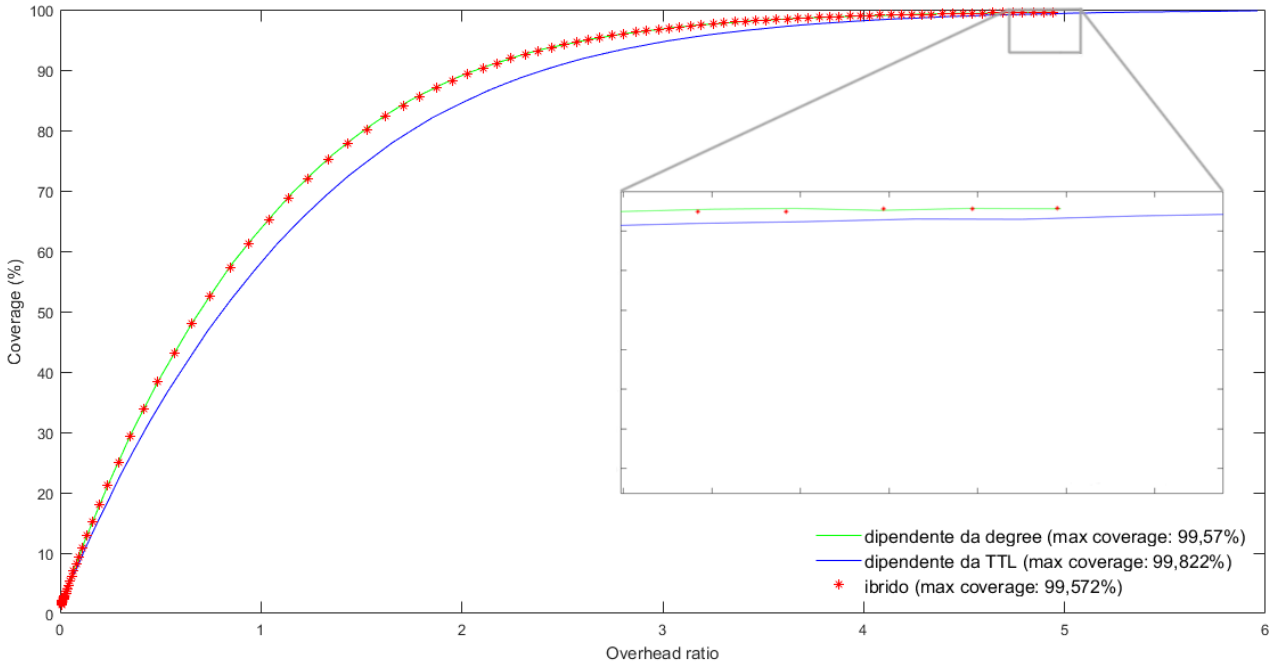



Immagine 5.1 Confronto tra meccanismo dipendente da degree, meccanismo dipendente da TTL e meccanismo ibrido degree/TTL (con dettaglio), FREERIDER=0,1.

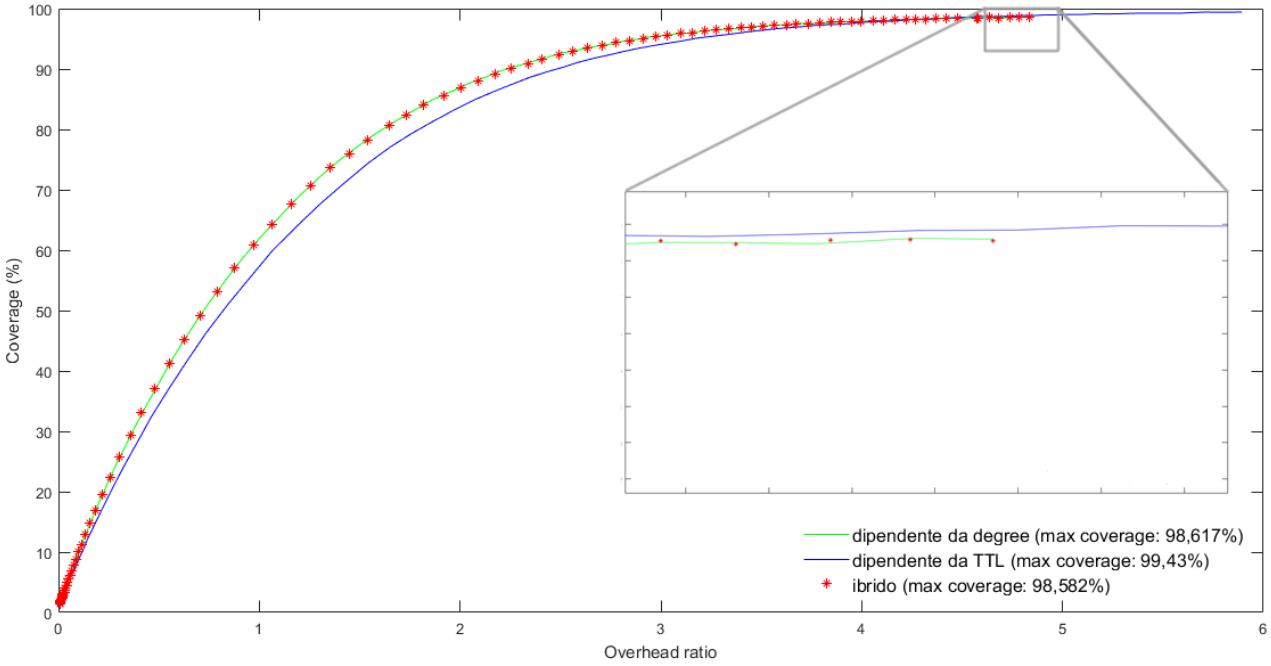



Immagine 5.2 Confronto tra meccanismo dipendente da degree, meccanismo dipendente da TTL e meccanismo ibrido degree/TTL (con dettaglio), FREERIDER=0,2.

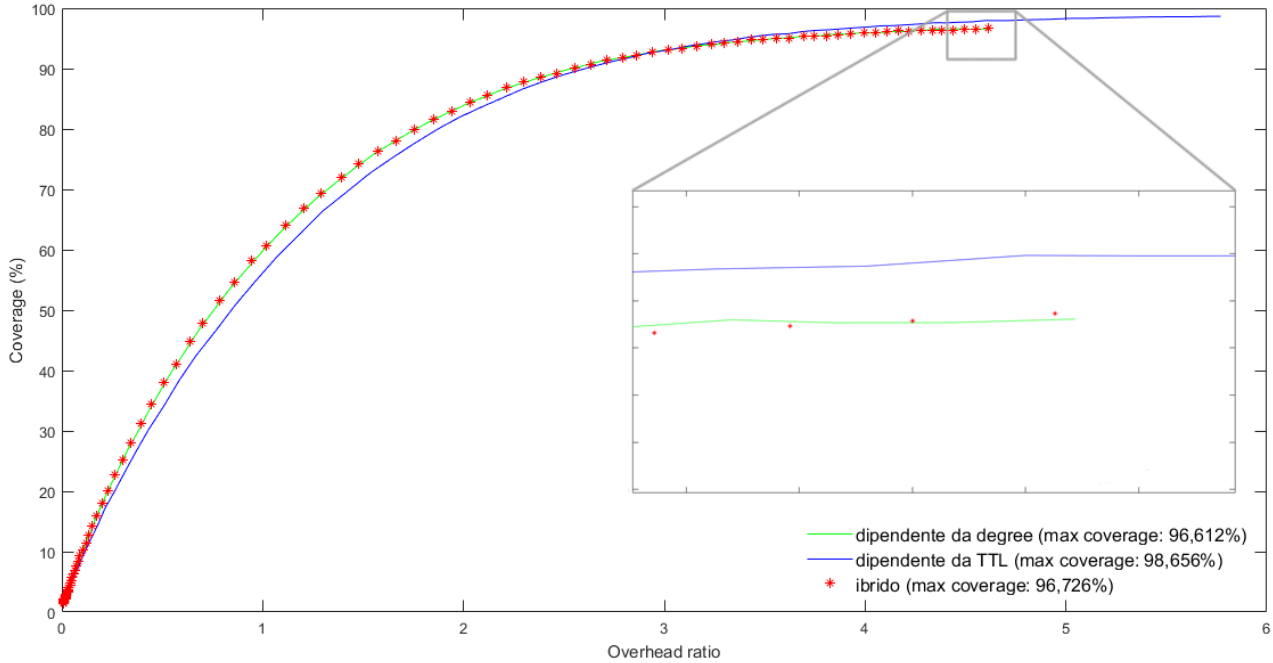



Immagine 5.3 Confronto tra meccanismo dipendente da degree, meccanismo dipendente da TTL e meccanismo ibrido degree/TTL (con dettaglio), FREERIDER=0,3.

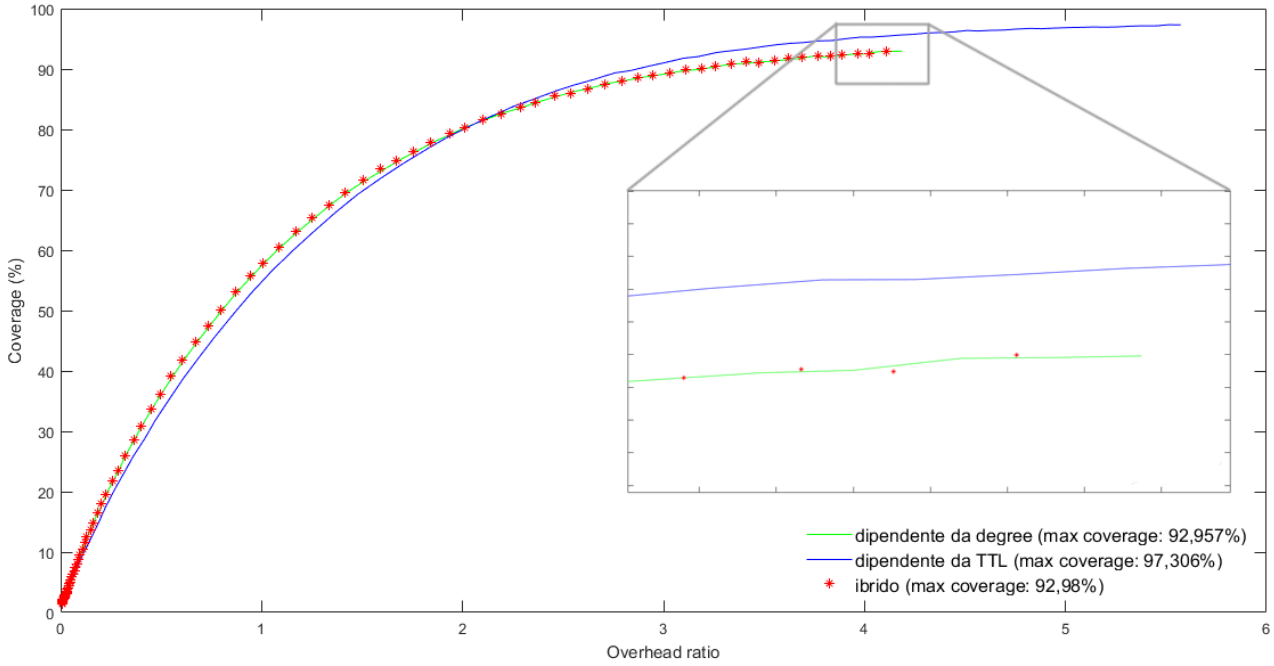



Immagine 5.4 Confronto tra meccanismo dipendente da degree, meccanismo dipendente da TTL e meccanismo ibrido degree/TTL (con dettaglio), FREERIDER=0,4.

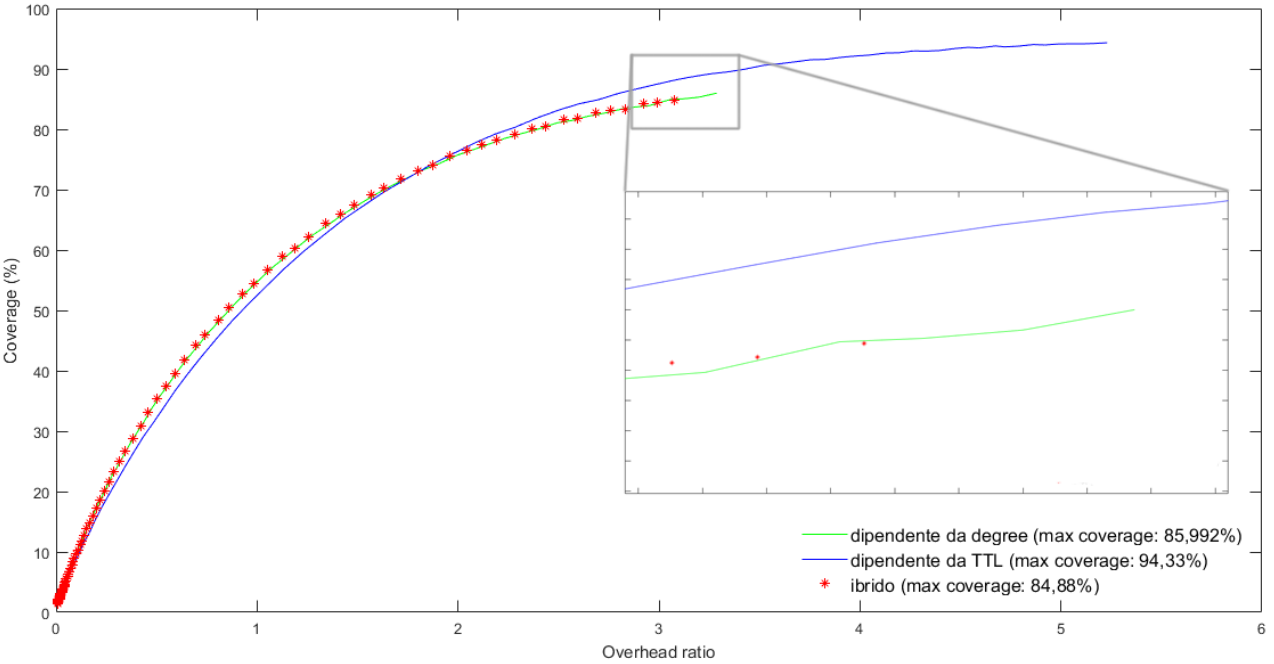



Immagine 5.5 Confronto tra meccanismo dipendente da degree, meccanismo dipendente da TTL e meccanismo ibrido degree/TTL (con dettaglio), FREERIDER=0,5.

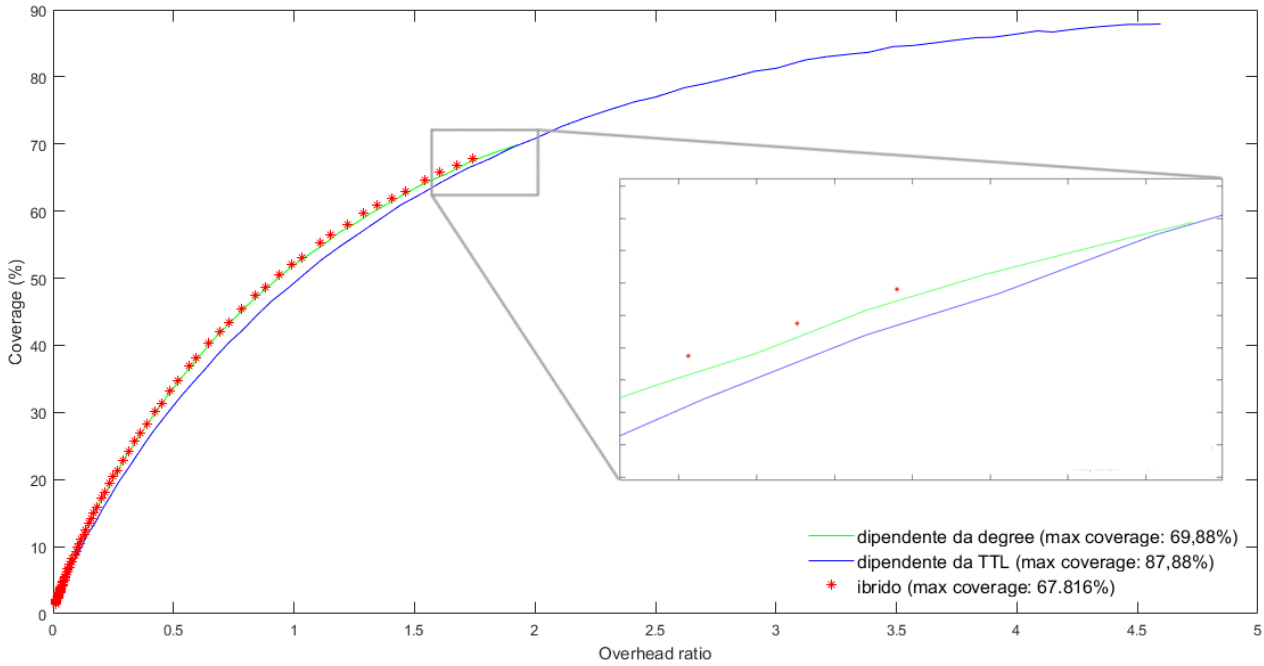



Immagine 5.6 Confronto tra meccanismo dipendente da degree, meccanismo dipendente da TTL e meccanismo ibrido degree/TTL (con dettaglio), FREERIDER=0,6.

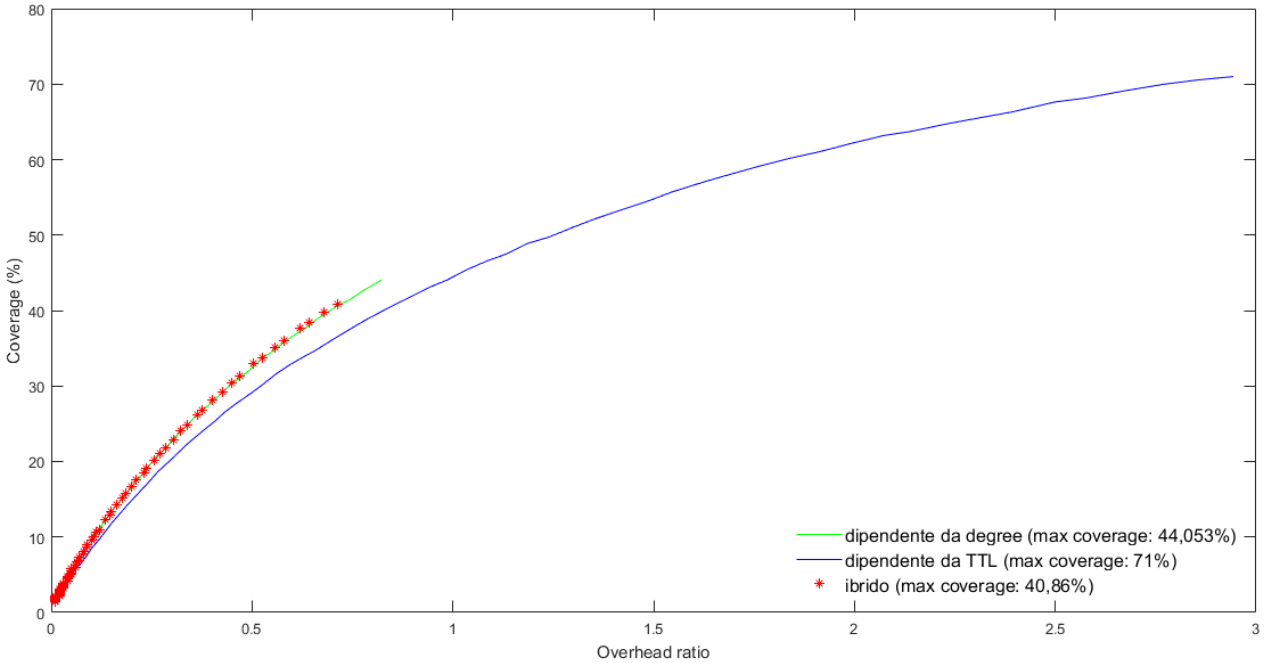



Immagine 5.7 Confronto tra meccanismo dipendente da degree, meccanismo dipendente da TTL e meccanismo ibrido degree/TTL (con dettaglio), FREERIDER=0,7.

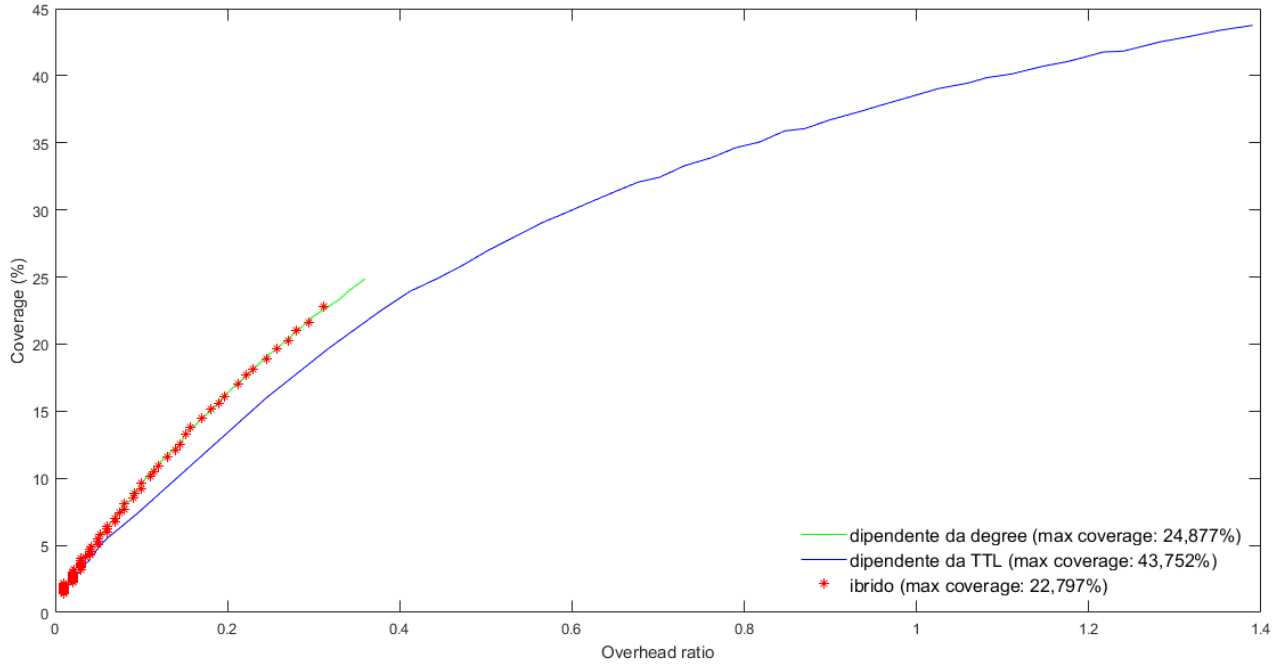



Immagine 5.8 Confronto tra meccanismo dipendente da degree, meccanismo dipendente da TTL e meccanismo ibrido degree/TTL (con dettaglio), FREERIDER=0,8.

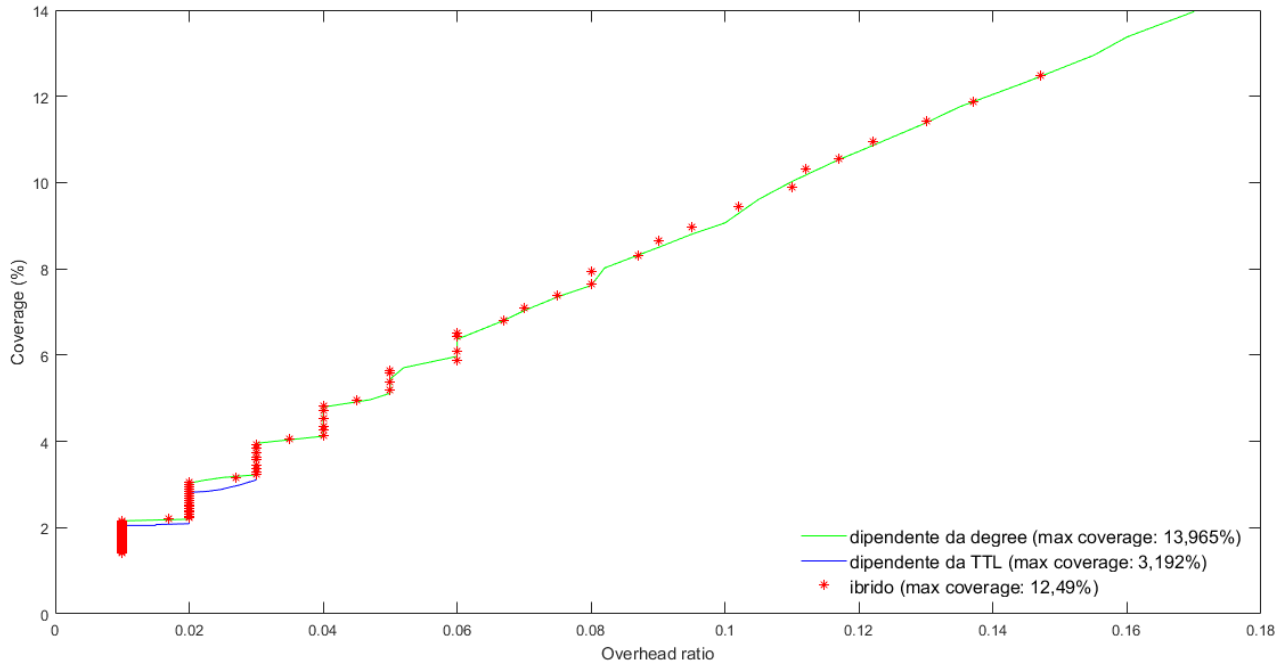



Immagine 5.9 Confronto tra meccanismo dipendente da degree, meccanismo dipendente da TTL e meccanismo ibrido degree/TTL (con dettaglio), FREERIDER=0,9.

## 5.3 Interpretazione dei risultati

Spiegare il motivo per cui gli insiemi di dati prodotti dalle simulazioni dove si è usato il meccanismo adattivo ibrido ricalchino così fedelmente le curve verdi del meccanismo dipendente da degree è piuttosto semplice. L'algoritmo 9 rappresenta sì una commistione degli algoritmi riportati nei capitoli 3 e 4, ma l'impianto generale rispecchia maggiormente quella dell'algoritmo 8, imperniato com'è sull'istruzione condizionale che valuta il modo di procedere in base al degree del nodo mittente. Il criterio di adattività basato sul time-to-live consiste invece nella generazione e successivo uso di un fattore moltiplicativo in congiunzione con la soglia di probabilità stabile senza alcun tipo di discriminazione. L'algoritmo definitivo risulta quindi essere a tutti gli effetti una riedizione dell'algoritmo 8 in cui le differenze per quanto riguarda le soglie di probabilità adattive generate non riescono ad essere sostanziali.

La fase di start-up, il momento della simulazione nella quale un nodo non è ancora "consapevole" del proprio degree, è anche il momento nel quale i nodi cominciano a scambiarsi i primi messaggi, che pertanto avranno ancora TTL massimo. Un messaggio appena generato incontra un free rider all'altro capo della connessione che sta percorrendo solo se il numero causale estratto è 0, dal momento che il fattore moltiplicativo associato a tale situazione è 0, che annulla a sua volta la soglia di probabilità adattiva. Nel momento in cui i nodi hanno cognizione di un solo vicino poiché hanno ricevuto un solo messaggio (quindi degree = 1), i messaggi inviati incontrano pochi free rider determinati con il meccanismo ibrido, considerato che il TTL è ancora sufficientemente alto da provocare un notevole riduzione della soglia di probabilità adattiva. Nel caso dei nodi effettivamente con un solo vicino, come già spiegato nel capitolo 4.2, non importa se questi siano free rider o meno dal momento che per regola non possono disseminare i messaggi ricevuti. In questo elenco di casi, quindi, la situazione tra un algoritmo e l'altro subisce poche o nessuna variazione.

Per tutti gli altri valori di degree del nodo mittente, la prima soglia adattiva calcolata come l'inverso del degree elevato alla soglia di probabilità stabile viene moltiplicata per il fattore moltiplicativo derivato dal TTL ed essendo quest'ultimo sempre compreso tra 0 e 1, la seconda soglia adattiva risultato di questa operazione aritmetica risulta sempre minore della prima. Non potrà nemmeno rimanere la stessa perché per avere un fattore moltiplicativo pari a 1 è necessario che il TTL corrente sià nullo ed in quel caso il messaggio viene automaticamente eliminato dal sistema. A causa del segno maggiore presente nell'istruzione condizionale alla riga 15 dell'algoritmo, la probabilità di incontrare un free rider (con conseguente decadimento del messaggio che non sarà più ritrasmesso) risulterà più elevata e questo causa la diminuzione dell'overhead ratio, percepibile soprattutto a partire dal batch di simulazioni con FREERIDER impostato a 0,3.

Per quanto riguarda il marginale miglioramento delle prestazioni di copertura per alcuni valori di overhead ratio (figure 5.3-5.4), una possibile motivazione dietro questo comportamento, al di là di errori di arrotondamento, potrebbe essere l'efficienza delle cache. Per via della presenza di un maggior numero di free rider nel network, viene consegnato un minor numero di messaggi e le cache implementate in ciascun nodo sono in grado di operare con un'efficienza lievemente superiore. [xxviii] Questo comporta meno messaggi duplicati in circolazione per la rete e perciò una copertura migliore per un dato valore di overhead ratio.

# Conclusioni

Il paradigma client-server può rivelarsi inadeguato in situazioni nelle quali deve confrontarsi con un massiccio carico di messaggi scambiati ed un numero di clienti altamente dinamico ed è in situazioni come queste che il paradigma peer-to-peer dimostra il suo valore garantendo efficienza, robustezza e scalabilità. Il grande successo a partire dai primi anni 2000 dei sistemi di file-sharing, una presenza oggi quasi irrinunciabile nella vita di tutti i giorni per milioni di persone, funge da testimonianza di quanto solida sia la base offerta da quest'architettura di rete basata su un criterio di decentralizzazione.

Nell'ambito di queste applicazioni distribuite, dove la natura dinamica dei peer implica la necessità di garantire sempre un alto grado di robustezza e scalabilità, un modo semplice per gestire il sostrato di interazione è offerto dalla combinazione di strategie agili di connessione allo scopo di creare diverse topologie di overlay network e disseminazione epidemica dei dati all'interno di questi ultimi. Ed è proprio nell'ambito della disseminazione che i protocolli di gossip risultano essere un caso di studio particolarmente interessante in quanto caratterizzati dall'assenza di piani di dirottamento precisi e non basati sui contenuti trasmessi, peculiarità dovute al criterio randomico con il quale il protocollo gestisce le comunicazioni.

In questa tesi si è affrontato il comportamento tipico delle reti P2P noto come free riding e grazie a LUNES, applicazione sviluppata allo scopo di testare le performance di protocolli di disseminazione tramite gossip, si sono condotte sperimentazioni su uno specifico caso di studio (topologia di rete + protocollo di gossip) per osservare come il fenomeno incidesse su due parametri in particolare, cioè il rapporto di copertura e l'overhead ratio. Nei primi test presentati nel capitolo 2 il criterio di determinazione dei nodi free rider all'interno del grafo era basato su una soglia di probabilità definita dall'utente di LUNES. Nei capitoli successivi (3, 4 e 5) invece i risultati dei test sono stati prodotti da versioni leggermente modificate dell'applicazione, nelle quali la soglia di probabilità che il nodo destinatario sia un free rider viene ricalcolata all'invio di ogni messaggio per far sì che sia dipendente anche dal numero di *hop* rimasti prima che il messaggio decada, dal numero di vicini del nodo mittente o da entrambi. Un sistema dove un nodo viene individuato o meno come free rider in base ad un criterio governato esclusivamente dalla casualità è un caso di studio interessante a livello teorico, ma facendo in modo che questo processo di determinazione dipenda anche da caratteristiche interne al sistema è possibile condurre analisi mirate sulle performance dei protocolli in scenari caratterizzati da concentrazioni di free rider in particolari regioni del network (quella centrale, nel caso di questa trattazione) oppure dove il numero di free rider cambia (e quindi, per esempio, può aumentare) con il passare del tempo.

Le simulazioni condotte sul meccanismo a soglia di probabilità fissa nel capitolo 2, che riprendono ed espandono gli studi sugli effetti del free riding riportati in [xxix], hanno confermato che la presenza di free rider ha un impatto sui livelli di copertura raggiungibili assai limitato fino a quando la percentuale di questo tipo di nodi rispetto a tutti quelli contenuti nel grafo si mantiene al di sotto del 50%. Allo stesso modo, i test su soglie di probabilità più alte hanno anche rivelato che le performance subiscono

un brusco declino se la percentuale di free rider nel grafo supera la metà dei nodi totali, al punto che per scenari con l'80% di free rider o più il livello di copertura è minore, anche notevolmente, della percentuale di nodi che ricevono e ritrasmettono i messaggi normalmente.

Anche per quanto riguarda i meccanismi adattivi sviluppati nell'ambito di questa tesi, che proiettano questa ricerca verso territori ancora poco esplorati vista la penuria di letteratura scientifica a riguardo, le simulazioni hanno portato alla luce comportamenti interessanti. Nel caso di dipendenza dal time-to-live del messaggio, si è osservato che la bassa probabilità che un messaggio incontri un free rider durante i suoi primi *hop* fa sì che il volume di traffico veicolato dal sistema aumenti considerevolmente e che ciò permetta un notevole miglioramento del rapporto di copertura, in special modo per valori di soglia di probabilità impostata dall'utente superiori a 0,5. Nel caso di dipendenza dal degree del nodo, si è notato invece un peggioramento delle prestazioni relative al parametro di copertura nonostante anche in questo caso il traffico di messaggi per lo più aumentasse rispetto allo scenario a soglia di probabilità fissa, almeno fino a valori della variabile d'ambiente pari a 0,8 o superiori, per i quali questo secondo meccanismo adattivo si conferma essere il più robusto tra quelli affrontati. Nel caso infine di dipendenza da caratteristiche sia del messaggio che del nodo, i risultati sono paragonabili come andamento a quelli del meccanismo dipendente da degree ma con prestazioni massime via via più ridotte all'aumentare della soglia stabile.

Il fenomeno del free riding si conferma quindi essere un grande ostacolo alla disseminazione di informazioni su overlay network, ma gli esperimenti compiuti su particolari scenari nei quali lo status di free rider di un particolare nodo non è dovuto a criteri basati esclusivamente sulla casualità gettano luce su aspetti del come questi ostacoli influenzino le performance della disseminazione tramite gossip che in certi casi possono non essere ovvi.

# Bibliografia


[i] G. Cirnigliaro, *Data dissemination on overlay networks through gossip. Tesi di laurea in reti di calcolatori,* Alma Mater Studiorum - Università di Bologna, Campus di Cesena, Scuola di Scienze, a.a. 2012-2013, pag. iii

[ii] G. D'Angelo, S. Ferretti, *Highly intensive data dissemination in complex networks*, Journal of Parallel and Distributed Computing, Elsevier, vol. 99, January 2017, pag. 2

[iii] Devavrat Shah, *Network gossip algorithms. Acoustics, speech and signal processing*, 2009. ICASSP 2009, IEEE International Conference on, pagg. 3673-3676

[iv] G. D'Angelo, S. Ferretti, *Simulation of scale-free networks*, Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, 2009, pagg. 1–10

[v] R. K. Kincaid, N. M. Alexandrov, *Scale-Free Networks: A discrete event simulation approach*, International Conference on Computational Science (1), 2005

[vi] R. Dobrescu, S. Taralunga, S. Mocanu, *Web traffic simulation with scale free network models*, AIC '07: Proc. of the 7th Conference on 7th WSEAS Internatinal Conference on Applied Informatics and Communications, WSEAS, 2007

[vii] D. Hughes, G. Coulson, J. Walkerdine, *Free riding on Gnutella revisited: the bell tolls?*, Distributed Systems Online, IEEE 6 (6)

[viii] J. Eberspächer, R. Schollmeier, *First and second generation of peer-to-peer systems*, all'interno di *Peer-to-Peer Systems and Applications*, Lecture Notes in Computer Science vol. 3485, Springer, 2005, pagg. 35–56

[ix] P. Erdös, A. Rényi, *On random graphs*, 1, Publicationes Mathematicae, Debrecen, 1959, pagg. 290-297

[x] A.-L. Barabási, R. Albert, *Emergence of Scaling in Random Networks*, Science 286, 1999, pagg. 509-512

[xi] R. Albert, A.-L. Barabási, S*tatistical mechanics of complex networks*, Rev. Mod. Phys. 74, 2002, pagg. 47–97

[xii] D. J. Watts, S. H. Strogatz, *Collective dynamics of 'small-world' networks*, Nature 393 (6684), 1998, pagg. 409–10

[xiii] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, G. Varghese, *Network monitoring using traffic dispersion graphs (tdgs)*, Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference, ACM, New York, NY, USA, 2007, pagg. 315–320

[xiv] G. D'Angelo, S. Ferretti, Highly intensive data dissemination in complex networks, Journal of Parallel and Distributed Computing, Elsevier, vol. 99, January 2017, pag. 7

[xv] S. Ferretti, *Gossiping for resource discovering: An analysis based on complex network theory*, Future Generation Computer Systems 29 (6), 2013, pagg. 1631 – 1644, che include *Special sections: High Performance Computing in the Cloud and Resource Discovery Mechanisms for {P2P} Systems*

[xvi] G. D'Angelo, S. Ferretti, M. Marzolla, *Adaptive event dissemination for peer-to-peer multiplayer online games*, Proceedings of Workshop on DIstributed SImulation and Online gaming, DISIO 2011, ICST, Bruxelles, Belgium, 2011

[xvii] G. Cirnigliaro, *Data dissemination on overlay networks through gossip. Tesi di laurea in reti di calcolatori*, Alma Mater Studiorum - Università di Bologna, Campus di Cesena, Scuola di Scienze, a.a. 2012-2013, pag. 14

[xviii] B. Garbinato, D. Rochat, M. Tomassini, *Impact of scale-free topologies on gossiping in ad hoc networks*, NCA, IEEE Computer Society, 2007, pagg. 269–272

[xix] S. Verma, W. T. Ooi, *Controlling gossip protocol infection pattern using adaptive fanout*, ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, IEEE Computer Society, Washington, DC, USA, 2005, pagg. 665–674

[xx] L. Bononi, M. Bracuto, G. D'Angelo, L. Donatiello, *Scalable and efficient parallel and distributed simulation of complex, dynamic and mobile systems*, Techniques, Methodologies and Tools for Performance Evaluation of Complex Systems, 2005, FIRBPerf, 2005 Workshop on, pagg. 136 - 145

[xxi] IEEE Computer Society, *1516-2000 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules*

[xxii] G. D'Angelo, S. Ferretti, *LUNES: Agent-based simulation of P2P systems (extended version)*, Proceedings of the International Workshop on Modeling and Simulation of Peer-to-Peer Architectures and Systems (MOSPAS 2011). Parte della 2011 International Conference on High Performance Computing and Simulation (HPCS 2011), ISBN 978-1-61284-382-7

[xxiii] G. D'Angelo, M. Bracuto, *Distributed simulation of large-scale and detailed models*, Int. J. Simulation and Process Modelling, Vol. 5, No. 2, 2009, pagg.120 - 131

[xxiv] G. D'Angelo, S. Ferretti, Highly intensive data dissemination in complex networks, Journal of Parallel and Distributed Computing, Elsevier, Vol. 99, January 2017, pagg. 17-18

[xxv] M, Feldman, C. Papadimitriou, J. Chuang, I. Stoica, *Free-riding and whitewashing in peer-to-peer systems*, Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in netowrked systems, ACM, 2004, pagg. 228-236

[xxvi] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, Maarten Van Steen. The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. H.-A. Jacobsen (Ed.): Middleware, 2004 , LNCS 3231, pagg. 79-98

[xxvii] G. Cirnigliaro, *Data dissemination on overlay networks through gossip. Tesi di laurea in reti di calcolatori*, Alma Mater Studiorum - Università di Bologna, Campus di Cesena, Scuola di Scienze, a.a. 2012-2013, pagg. 25-27

[xxviii] G. D'Angelo, S. Ferretti, Highly intensive data dissemination in complex networks. Journal of Parallel and Distributed Computing, Elsevier, Vol. 99, January 2017, pag. 27

[xxix] G. D'Angelo, S. Ferretti, Highly intensive data dissemination in complex networks. Journal of Parallel and Distributed Computing, Elsevier, Vol. 99, January 2017, pag. 27 & 44