UNIVERSITÀ
DEGLI STUDI
DI PALERMO

NDS
NETWORKING AND DISTRIBUTED SYSTEMS
RESEARCH GROUP

Progettazione e sviluppo di una piattaforma per la valutazione delle vulnerabilità dei sistemi di gestione della reputazione

Tesi di Laurea Magistrale in Ingegneria Informatica

S. Pecora Cucchiara

Relatore: Prof. Giuseppe Lo Re
Correlatori: Ing. V. Agate

Progettazione e sviluppo di una piattaforma per la valutazione delle vulnerabilità dei sistemi di gestione della reputazione

Tesi di Laurea di
Dott. Salvatore Pecora Cucchiara

Relatore:
Ch.mo Prof. Giuseppe Lo Re

Controrelatore:

Correlatore:
Ing. Vincenzo Agate

Sommario

Nelle applicazioni distribuite attraverso la rete, agenti sconosciuti interagiscono scambiandosi servizi al fine di ottenere mutui benefici. In questo contesto, i *Sistemi di Gestione della Reputazione* (*Reputation Management Systems* o RMS), incoraggiano comportamenti cooperativi e scoraggiano comportamenti anti sociali, offrendo supporto a ciascun agente nella scelta del fornitore a cui richiedere il servizio. Data l'eterogeneità dei sistemi di reputazione presenti in letteratura, non esistono piattaforme e metodologie di uso comune che possano assistere il progettista di un nuovo RMS. In questo lavoro di tesi, viene presentata una piattaforma di simulazione per la valutazione della vulnerabilità dei sistemi di reputazione. Il simulatore proposto è pensato per offrire la massima versatilità nella definizione degli scenari da simulare e del sistema di reputazione da valutare.

Indice

Introduzione **1**

Struttura della Tesi **2**

1 Stato dell'Arte **4**

1.1 Valutazione dei Sistemi di Reputazione 4
1.1.1 Valutazione di sistemi di reputazione con esperimenti *custom-made* 5
1.1.2 Esperimenti sotto scenari ristretti 5
1.1.3 Valutazione con piattaforme generiche 6
1.1.4 Limiti dei correnti CEF basati su simulazione 7
1.1.5 Proprietà desiderabili di un CEF 7

1.2 Attacchi ai sistemi di reputazione 8
1.2.1 Self Promoting 9
1.2.2 Slandering 9
1.2.3 Whitewashing 10
1.2.4 Attacco del Traditore 11
1.2.5 Attacchi orchestrati 11
1.2.6 Denial Of Service 12

1.3 Sistemi di reputazione 12

1.4 Eigentrust 13
1.4.1 Considerazioni Progettuali 14
1.4.2 Basic EigenTrust 14
1.4.3 Considerazioni pratiche 15
1.4.4 Secure Eigentrust 16

1.5 RMS caso di studio 17

2 Progettazione **19**

2.1 Obiettivi 19

2.2 Glossario e Acronimi 19

2.3 Sistema Proposto 19
2.3.1 Requisiti funzionali 19
2.3.2 Requisiti non funzionali 20
2.3.3 Documentazione 20
2.3.4 Considerazioni sull'hardware 20
2.3.5 Pseudo requisiti 21
2.3.6 Modelli del sistema: scenari 21

2.4 Diagrammi delle classi 24
2.4.1 Gerarchia degli Agenti 24
2.4.2 Gerarchia dei Behaviors 25
2.4.3 Relazione Behavior membri RMS 26
2.4.4 Relazione behavior Provider 26
2.4.5 Composizione behavior 27

2.4.6 Diagramma completo 28
2.5 Diagrammi di sequenza 29
2.5.1 Ciclo di vita di un agentNode 29
2.5.2 Dettaglio del singolo passo di un nodeAgent 31
2.5.3 Dettaglio singolo passo del leadingAgent 32
2.5.4 Dettaglio singolo passo del providersDispatcher 33
2.5.5 Esempio calcolo reputazione 33
2.5.6 Esempi sequenze di attacco 35
2.5.7 Mock-ups 37
2.6 MPI e MPICH 38
2.6.1 Comunicazione bloccante e non bloccante 38
2.6.2 Tipi composti 38
2.6.3 Impossibilità di creare nuovi processi durante l'esecuzione 39
2.6.4 Le primitive più importanti utilizzate 39
2.6.5 Esecuzione di un programma MPI 40
3 Architettura Funzionale 41
3.1 Architettura a due livelli 41
3.2 Modello basato su Agenti 42
3.2.1 I nodi agente 43
3.2.2 Il *leading agent* 44
3.2.3 Il *dispatcher* dei provider 45
3.3 Overview 46
3.4 Il generatore della topologia 46
3.4.1 Interazione con l'utente 47
3.4.2 Comportamenti base 47
3.4.3 Comportamenti personalizzati 47
3.4.4 Specifiche 48
3.4.5 Numero di nodi 48
3.4.6 Percentuale di vicini 48
3.4.7 Percentuale di nodi per dato comportamento 48
3.4.8 Lunghezza media elenco degli obiettivi 48
3.4.9 Beginning Step 49
3.4.10 Tipo del comportamento 49
3.4.11 File creati 49
3.4.12 Nodelist.txt 49
3.4.13 Topology.txt 50
4 Esperimenti 51
4.1 Hardware delle simulazioni 51
4.2 Esempi di metriche 52
4.2.1 Reputazione media 52
4.2.2 Errore su singolo nodo ed errore totale medio 52
4.3 Il caso di studio 53
4.3.1 Setting sperimentale 54

4.3.2 Slandering....54
4.3.3 Promoting....56
4.3.4 Attacco del traditore....57
4.3.5 Whitewashing....59
4.4 Caso Studio con filtro in invio pseudocasuale.... 61
4.5 Caso studio con filtro in ricezione pseudocasuale.... 61
4.6 Caso studio con soglia in invio.... 62
4.7 Secure Eigentrust.... 63
4.7.1 Attacco del traditore....64
4.7.2 Slandering e Promoting....65
Conclusioni....67
Indice delle figure....68
Bibliografia....70

Introduzione

Diverse applicazioni odierne, ad esempio quelle di e-commerce o per lo scambio di file (p2p), si basano sull'interazione di agenti che cooperano scambiando servizi al fine di raggiungere un obiettivo. Tali applicazioni operano nei cosiddetti ambienti distribuiti. Gli agenti di un tale sistema possono però essere motivati da interessi diversi. Alcuni ad esempio potrebbero essere spinti da interessi egoistici, altri da interessi malevoli, altri ancora da entrambi. Un'architettura distribuita fornisce quindi la struttura ideale per quegli agenti che, attaccando il sistema, desiderano far degradare le performance dell'applicazione o arrecare danni ad altre entità della rete. I sistemi di reputazione forniscono uno strumento valido per contrastare tali comportamenti, promuovendo atteggiamenti onesti e collaborativi e scoraggiando comportamenti antisociali. Un RMS infatti, permette, per ciascun partecipante, di stimare un valore di reputazione. Sulla base dei valori di reputazione, ciascun partecipante può scegliere (secondo un determinato criterio) le entità con cui interagire.

Tipicamente anche la struttura dei sistemi di gestione della reputazione è di tipo distribuito. Infatti, se un'architettura centralizzata permette il calcolo e l'accesso ai valori di reputazione, questa esibisce anche un unico punto di rottura (in caso di guasti o attacchi, ad esempio di tipo DOS) e si pone verosimilmente come collo di bottiglia per le performance del sistema. D'altra parte, un'architettura distribuita rende più complicati la valutazione della convergenza dell'algoritmo e dell'accuratezza dei valori calcolati rispetto al comportamento effettivo degli agenti, e si espone ad attacchi che prevedano la manipolazione dei valori durante la diffusione di informazioni. In questo tipo di architettura infatti, ciascun agente, attraverso la conoscenza di un valore locale di fiducia (*local trust*) e i valori diffusi (reputazione) da altri partecipanti (attraverso *feedback)*, calcola un valore di fiducia globale (*global trust*).

Durante la fase di progettazione di un nuovo sistema di reputazione, la scelta più comune da parte dei progettisti è quella di sviluppare simulatori ad-hoc. Diverse proposte di simulatori generici sono state effettuate in molti lavori di ricerca ma nessuna di queste al momento soddisfa a pieno le esigenze dei progettisti. Questo lavoro di tesi dunque, ispirandosi ai lavori [1] [2] [3] e reingegnerizzando il sistema ivi proposto, si pone l'obiettivo di assistere i ricercatori fornendo classi già implementate da utilizzare, interfacce da implementare e uno strumento per la generazione di topologie casuali per simulazioni su larga scala, permettendo ai designer dei RMS di occuparsi unicamente dei dettagli di alto livello. L'impiego di

un linguaggio di programmazione ad oggetti, quale il C++, conferisce poi al sistema una maggiore modularità.

Rispetto ai lavori esistenti [1], [2] e [3], l'ambiente di simulazione proposto permette una maggiore dinamicità nella valutazione dei servizi offerti dai fornitori e nel calcolo delle metriche. Il simulatore infatti, permette ai fornitori di offrire più servizi e di variarli durante la simulazione. In questo modo, è possibile valutare il lo stesso fornitore al variare della tipologia di servizio. La piattaforma inoltre, tenendo traccia dello storico delle transazioni reali, permette di calcolare metriche che prendano in considerazione sia un punto di vista distribuito, cioè i valori calcolati dai partecipanti al sistema di reputazione, sia un punto di vista centralizzato, caratterizzato dalle transazioni reali. Misure di questo tipo permettono al progettista stimare la distanza tra la realtà delle transazioni e quanto riportato dagli agenti della rete.

Nel simulatore presentato, il sistema di reputazione è modellato come un algoritmo distribuito. La comunicazione tra i nodi è garantita dall'utilizzo del protocollo *Message Passing Interface,* che permette l'esecuzione parallela anche su dispositivi connessi in rete. In questo modo simulazioni con numero elevati di nodi possono essere eseguite su differenti architetture di calcolo.

Struttura della Tesi

Nella prima parte di questo lavoro di tesi vengono discussi due esempi di sistemi di reputazione e vengono illustrate le caratteristiche generali di questo tipo di algoritmi. Nello stesso capitolo, vengono presentati anche alcuni approcci possibili per la valutazione dei sistemi di reputazione e i tipi più comuni di attacco a cui essi sono esposti.

I capitoli successivi sono invece volti interamente alla presentazione del sistema. Nel secondo capitolo, in particolare, vengono presentate le considerazioni progettuali alla base del simulatore. Più precisamente, in questa sezione sono illustrati alcuni diagrammi UML che mostrano la struttura della piattaforma ed è presentato brevemente il protocollo utilizzato per la programmazione distribuita.

Nella terza parte vengono presentati l'architettura funzionale del simulatore, strettamente legata alle considerazioni progettuali fatte nel capitolo precedente, e il generatore della topologia, un modulo dell'ambiente di sviluppo utile per simulazioni complesse e randomizzate.

Infine, nell'ultimo capitolo vengono presentati i risultati sperimentali ottenuti

dai due algoritmi descritti nella prima sezione e da alcune varianti di uno di essi. .I risultati ottenuti dimostrano la bontà dell'approccio seguito e permettono considerazioni utili ai progettisti che volessero cimentarsi nell'ideazione di un nuovo sistema di reputazione o verificare la vulnerabilità ad alcune classi di attacco.

1 Stato dell'Arte

I sistemi di reputazione (*Reputation Management Systems* o *RMS)* giocano un ruolo cruciale per il corretto funzionamento delle applicazioni distribuite [4]. In questo tipo di sistemi, entità sconosciute collaborano scambiando risorse o servizi. Se si immagina di mappare gli agenti di un'applicazione distribuita su un grafo, ad ogni agente corrisponderà un nodo del grafo. Di seguito dunque si farà riferimento al termine nodo come sinonimo del termine agente.

Il compito del RMS è quello di assistere ciascun nodo nella scelta di altre entità fidate con cui interagire [5]. Progettare un sistema di reputazione robusto non è certamente un compito semplice. Diventa quindi essenziale per i progettisti poter testare in maniera adeguata questo tipo di algoritmi, al fine di prendere le decisioni più adeguate già in fase di progettazione.

1.1 Valutazione dei Sistemi di Reputazione

Fin adesso, la valutazione dei sistemi di reputazione si è basata per lo più su metodi proprietari, a causa della mancanza di metodi e misure standardizzate o di ampia accettazione. Di conseguenza, diventa complicato poter confrontare i risultati ottenuti dai diversi RMS sia per i designers di nuovi algoritmi, i quali potrebbero prendere spunto per studiare migliorie ai loro sistemi, sia per progettisti e sviluppatori di applicazioni che vogliono scegliere il sistema di reputazione più adatto alle loro esigenze [6] [7] [8].

Per valutare i loro sistemi di reputazione, i progettisti possono scegliere di utilizzare esperimenti e criteri costruiti su misura, esperimenti pensati da altri ricercatori o piattaforme e strumenti appositamente pensati per facilitare la simulazione e la valutazione dei RMS. Negli esperimenti proposti in letteratura, i progettisti hanno scelto principalmente di basarsi su esperimenti e simulatori personalizzati. Sono anche stati proposti dei meccanismi per il confronto dei sistemi di reputazione. Tali meccanismi però prevedono scenari specifici, che, sebbene siano comuni a diversi sistemi di valutazione, non possono essere generalizzati. Alcuni di questi meccanismi ad esempio prevedono esperimenti basati sul dilemma del prigioniero o test specifici impiegati nel caso di applicazioni di commercio elettronico.

Data la necessità di poter confrontare in ambienti generici differenti RMS, molti lavori di ricerca hanno avuto come oggetto lo sviluppo e l'impiego di piattaforme comuni di valutazione. Alcune di questi ambienti di simulazione sono teorici, altri offrono piattaforme e strumenti per l'implementazione, la simulazione e il confronto dei propri sistemi. Gli autori di [9] classificano i vari approcci in:

- lavori che usano simulazioni personalizzate o proposte da altri ricercatori;
- lavori che si basano su simulazioni effettuate su scenari ristretti;
- lavori che offrono un ambiente di simulazione generico.

Nei paragrafi seguenti sono illustrati brevemente le tre categorie di ricerche elencate. Successivamente sono illustrati invece i limiti dei simulatori proposti.

1.1.1 Valutazione di sistemi di reputazione con esperimenti *custom-made*

Le simulazioni con esperimenti personalizzati differiscono per parametri della simulazione e natura. Gli esperimenti di questa categoria pongono l'attenzione sulla "distanza" tra i valori di reputazione calcolati e quelli effettivi (*correctness of predictions*) e sulle performance del sistema, ossia il tempo necessario affinché l'algoritmo di calcolo converga. Data l'assenza di metriche standardizzate, simulazioni di lavori differenti utilizzano metriche differenti. Gli autori di [10] valutano l'affidabilità della valutazione (*reliability of trust assessments*) attraverso il rapporto delle valutazioni delle transazioni infruttuose. Similmente, gli autori di [11], utilizzano in PeerTrust il tasso di transazioni positive (*success rate of transactions*) e l'errore di computazione (*computation error*) che corrisponde alla varianza interna tra i valori di fiducia calcolati e quelli effettivi. Metriche per la valutazione della convergenza sono invece la *convergence overhead,* calcolata come il numero di iterazioni necessarie affinché i valori di reputazioni assumano valori considerati reali, e la *ranking discrepancy,* data semplicemente dalla differenza tra valori stimati e valori reali. In altre simulazioni è preso in considerazione *l'overhead* necessario dei messaggi per lo scambio dei valori di reputazione tra i nodi. In NICE [12] gli autori valutano la scalabilità del sistema in termini di memoria e costi di esecuzione, e la stabilità del sistema in termini di numero di transazioni necessarie affinché il sistema si stabilizzi.

Queste metriche, che sono alcune tra le numerose impiegate nelle simulazioni, illustrano bene le proprietà del sistema.

1.1.2 Esperimenti sotto scenari ristretti

I simulatori che rientrano in questa categoria permettono la simulazione degli RMS solo sotto determinati tipi di scenario. Ad esempio alcuni di questi prevedono che la reputazione sia un valore binario, ipotesi spesso non soddisfatta, altri prevedono esperimenti basati sulla Teoria dei Giochi e più nello specifico sul

Dilemma del Prigioniero e cosi via. Per questo motivo i simulatori di questo tipo non possono considerarsi piattaforme di simulazione generica, più comunemente conosciute come *Common Evaluation Framework* (CEF), in quanto queste per definizione permettono la simulazione sotto scenari generici.

1.1.3 Valutazione con piattaforme generiche

Le piattaforme per la valutazione e il confronto dei sistemi di reputazione vengono distinti in: piattaforme basate sul confronto teorico e piattaforme basate sul confronto delle simulazioni con diversi sistemi di reputazione [9].

Le prime confrontano i RMS valutandone la resistenza e l'adattabilità alle diverse minacce o valutandone specifiche proprietà attraverso una serie di criteri teorici, o entrambi. Le seconde invece, attraverso il confronto dei risultati dei diversi sistemi di gestione della reputazione ai diversi scenari di simulazione, evidenziano debolezze e punti di forza di ciascuno. Nel paragrafo seguente saranno illustrati sommariamente i limiti delle principali piattaforme basate su simulazione, categoria alla quale appartiene il simulatore presentato in questo lavoro, e ne vengono messi in evidenza i limiti più stringenti [9].

Come accennato precedentemente, il vantaggio di un approccio basato su simulazioni è che questo offre la possibilità di confrontare diversi sistemi di reputazione sotto gli stessi scenari simulativi.

Gli autori di [13] propongono DTT, un insieme di strumenti per l'implementazione e la valutazione dei RMS in diversi scenari simulativi. Sebbene questo contenga una serie di strumenti utili per il miglioramento, l'estensione e il riuso di sistemi già implementati, non fornisce tuttavia un metodo semplice per confrontare i risultati ottenuti attraverso diversi sistemi di reputazione. Similmente, PACE, proposto in [14], contiene una serie di strumenti per l'integrazione del modello di trust in un'architettura distribuita.

TOSim [15] permette simulazioni con comportamenti dinamici dei nodi, è ampiamente estensibile e scalabile e permette la simulazione di diversi modelli di attacco. Lo scenario simulativo è quello del *P2P file Sharing*.

ART Testbed [16] permette ai progettisti dei RMS di eseguire simulazioni personalizzate e fornisce all'utente un insieme di metriche. Tuttavia, ART pone dei vincoli sui sistemi simulabili che limita la gamma di sistemi di reputazione che si possono testare. Inoltre il supporto non è più disponibile.

TREET [17] permette di testare diversi sistemi di reputazione ma offre la possibilità di effettuare simulazioni solo su scenari di tipo *marketplaces*.

DART [18] si basa sul dilemma del prigioniero e fornisce una serie di strumenti e metriche per la simulazione e la valutazione dei sistemi di reputazione. È una piattaforma flessibile in quanto fornisce la possibilità di implementare RMS e metriche personalizzate. I due principali limiti sono la dipendenza dei risultati della simulazione non solo dal modello di RMS ma anche da altre politiche, e la non disponibilità pubblica dei sistemi implementati in DART e degli esperimenti.

1.1.4 Limiti dei correnti CEF basati su simulazione

Dagli esempi indicati, gli autori di [9] individuano una serie di limiti che caratterizzano i CEF basati su simulazione. In particolare indicano i seguenti vincoli:

- caratteristiche supportate dall'ambiente di simulazione, ad esempio numero di agenti, comportamento dinamico dei nodi ecc…;
- differenza nelle metriche usate;
- tipi di attacchi simulabili;
- tipo di sistemi di reputazione testabili;
- impossibilità di modellare meccanismi decisionali nell'ambiente di simulazione.

1.1.5 Proprietà desiderabili di un CEF

Sempre gli autori di [9], individuano una serie di proprietà desiderabili per un CEF. La prima di queste proprietà è la *standardizzazione* secondo cui un CEF deve includere un modello astratto di reputazione che possa essere utilizzata per implementare i RMS e le metriche della simulazione.

Un CEF dovrebbe essere indipendente dal tipo di architettura del sistema di reputazione, dalle caratteristiche delle transazioni, dal tipo di metriche utilizzate e dal tipo di servizi scambiati (*indipendenza dalle caratteristiche del sistema di reputazione*).

La terza proprietà è la *flessibilità*: la piattaforma deve permettere la simulazione di diversi tipi di attacco, di diversi aspetti del sistema di reputazione e dell'ambiente da simulare (dimensione della popolazione, scenario delle transazioni, tipi di servizi, ecc..).

Per concludere, sono sicuramente auspicabili la possibilità di implementare in maniera *semplice* nuovi sistemi di reputazione e nuovi esperimenti, nonché la *disponibilità* di RMS già implementati, al fine di rendere semplice il confronto tra i risultati ottenuti da un nuovo sistema di reputazione ed altri già noti.

Vedremo nei capitoli successivi come l'ambiente di simulazione proposto in questo lavoro cerchi di soddisfare le proprietà appena elencate e di superare i limiti presenti nei lavori su citati.

1.2 Attacchi ai sistemi di reputazione

La natura aperta dei sistemi di reputazione distribuiti e delle applicazioni distribuite in cui essi sono impiegati, li rende particolarmente esposti ad attacchi da parte di utenti malevoli che possono manipolare i valori di reputazione. In [19] troviamo una trattazione esaustiva degli attacchi tipici di questo tipo di applicazioni. Secondo gli autori, gli attaccanti possono essere catalogati in base al fatto che questi siano interni o meno alla rete. Nel primo caso si parla di attaccanti di tipo *insider,* ossia utenti che hanno legittimamente accesso al sistema, nel secondo caso invece di *outsider*. Gran parte delle applicazioni che impiegano sistemi di reputazione utilizzano meccanismi di autenticazione per evitare l'accesso ad agenti non autorizzati. Ne deriva che gli attaccanti sono per lo più *insider*.

Un'altra classificazione può essere effettuata sulla base dell'atteggiamento partecipativo degli agenti. Dato che gli attacchi prevedono la manipolazione di informazioni prima dell'invio nella rete, gli attaccanti devono partecipare *attivamente*. A seconda del meccanismo di diffusione delle informazioni però, un nodo onesto potrebbe trovarsi suo malgrado ad essere mezzo di diffusione per attaccanti attivi. In questo senso, un attaccante potrebbe partecipare *passivamente* all'attacco.

Si potrebbe anche distinguere tra attaccanti singoli e attaccanti organizzati in coalizioni. Dato che applicazioni di questo tipo comprendono popolazioni dell'ordine delle migliaia o dei milioni di nodi, un attaccante singolo non sarebbe comunque rilevante ai fini del corretto funzionamento del sistema. Praticamente tutte le classi di attacco presentate successivamente prevedono quindi coalizioni di nodi collusi. Questo rende ancora più critico l'attacco in quanto attacchi da parte di coalizioni sono più difficili da rilevare e contrastare.

L'ultimo criterio di classificazione è l'interesse che spinge un nodo a comportarsi in maniera antisociale. Distinguiamo infatti i nodi spinti da interesse *egoistici,* il cui interesse è trarre massimo profitto dal sistema modificando i valori a proprio vantaggio, dai nodi animati da interessi *malevoli* che, anch'essi modificando i valori di reputazione, cercano invece di arrecare danni ad altri nodi della rete.

Chiaramente le due motivazioni non sono mutuamente esclusive. Nei paragrafi successivi sono presentate alcune delle classi di attacco più comuni.

1.2.1 Self Promoting

In un attacco di tipo *Self Promoting*, i nodi collusi sono animati da interessi egoistici. L'obiettivo degli attaccanti è quello di far crescere i propri valori di reputazione. Tipicamente infatti il loro comportamento non è, o almeno non completamente, cooperativo. Per far ciò ciascun attaccante potrebbe ad esempio sfruttare eventuali lacune nella formula di calcolo dei valori o agire nella fase di invio dei valori, modificandoli prima opportunamente.

La forma più semplice di attacco di questo tipo infatti, potrebbe consistere semplicemente nella creazione di falsi feedback o nella modifica dei valori in fase di invio. Chiaramente ciò è possibile se il sistema di reputazione non prevede meccanismi per l'autenticazione dei dati o di controllo dell'integrità dei dati, caso in cui non è possibile distinguere tra un feedback legittimo ed uno falso. Sistemi che non prevedono meccanismi di autenticazione degli utenti e di verifica riguardo l'effettiva esistenza delle transazioni, sono vulnerabili ad un altro tipo di attacco: uno o più nodi infatti, possono assumere identità multiple (*sybil attack*). Queste identità possono poi recensirsi positivamente l'un l'altra.

Il risultato di questo tipo di attacco è che i nodi effettivamente onesti vengono penalizzati in quanto le recensioni reali vengono inserite con una frequenza minore rispetto a quella con cui le false recensioni vengono create o modificate. Per lo stesso motivo, l'effetto di eventuali recensioni negative nei confronti dei conduttori viene annullato dalle più numerose recensioni positive.

Per contrastare questo tipo di attacco, i sistemi dovrebbero verificare l'effettiva esistenza delle transazioni valutate e impedire la possibilità ad un nodo di assumere identità multiple. Si potrebbe anche pensare di individuare nella rete i conduttori, cercando i nodi che interagiscono quasi esclusivamente tra di loro. Si può però verificare che questo problema è NP-completo.

1.2.2 Slandering

In un attacco di *Slandering*, i nodi collusi, in maniera speculare a quanto illustrato nel caso precedente, creano falsi feedback negativi o modificano i valori prima di trasmetterli, in modo da danneggiare uno specifico nodo target dell'attacco.

I sistemi più vulnerabili a questo tipo di attacco sono quelli che non prevedono meccanismi di autenticazione e che hanno un'alta sensibilità alle recensioni negative. La determinazione della giusta sensibilità da parte dei progettisti del RMS è un compito molto delicato. Infatti, una bassa sensibilità alle recensioni negative permetterebbe ai nodi di comportarsi malevolmente per lunghi intervalli temporali. Viceversa, una sensibilità alta potrebbe penalizzare nodi onesti che vengono presi di mira da parte di nodi *diffamatori.*

I meccanismi che un RMS può integrare per combattere questa classe di attacco sono essenzialmente gli stessi discussi per il *Self Promoting*. In più, il sistema di reputazione potrebbe semplicemente calcolare i valori di reputazione solamente prendendo in considerazione i valori calcolati dalle informazioni "dirette". Nel caso però di basso numero di interazioni, queste non sono sufficienti ed è quindi necessario un meccanismo di inferenza di fiducia [20].

Nella nostra trattazione gli attacchi di *Self Promoting* e di *Slandering* sono stati etichettati come attacchi da membri del sistema di reputazione, in quanto questi appunto prevedono che gli attaccanti facciano parte del sistema di calcolo e/o diffusione dei valori.

1.2.3 Whitewashing

In un attacco di tipo *Whitewashing,* i conduttori abusano del sistema per brevi intervalli temporali, facendo quindi diminuire il loro valore di reputazione. Dopo questo intervallo temporale, i conduttori, in qualche modo, cercano di rimediare alla loro cattiva condotta. Nel caso più semplice, in cui l'autenticazione avvenga per pseudonimi facilmente sostituibili, i nodi in questione lasciano la rete per rientrare con una nuova identità, ottenendo quindi una reputazione ripulita.

I sistemi di reputazione più vulnerabili a questo tipo di attacco sono quelli in cui i valori di fiducia vengono calcolati tenendo conto esclusivamente delle transazioni negative. In questo caso infatti, la reputazione di un nodo che nel lungo termine ha agito onestamente sarà uguale a quella di un nodo appena entrato nella rete.

Per contrastare questo tipo di attacco, un RMS potrebbe evitare che entità appena entrate in rete assumano stessi valori di entità che hanno agito onestamente nel lungo termine. In più, potrebbe evitare che ciascun agente possa facilmente rientrare nel sistema assumendo una nuova identità.

1.2.4 Attacco del Traditore

Secondo [19], una variante del caso precedente è rappresentata dall'attacco del Traditore. Nei sistemi che considerano sia transizioni positive che transazioni negative, un agente malevolo potrebbe alternare un atteggiamento antisociale ad un atteggiamento cooperativo per impedire che la sua reputazione scenda a valori che non gli permetterebbero di essere più scelto per le successive transazioni.

Gli attacchi di *Whitewashing* e del Traditore si prestano ad essere combinati con attacchi di *Self Promoting* e diffamazione nei confronti di quei nodi che, giustamente, li recensiscono negativamente. In questo modo, i nodi antisociali minimizzano la necessità di ripulire la propria reputazione.

I sistemi a rischio per questo tipo di attacco sono quelli che pesano in egual misura transazioni recenti e transazioni più remote. Per contrastare agenti antisociali di questo tipo quindi, un RMS dovrebbe pesare in maniera differente transazioni che avvengono in momenti temporali differenti, dando un peso maggiore alle transazioni più recenti. Addirittura il sistema di reputazione potrebbe prendere in considerazione solo le transazioni avvenute in un intervallo temporale limitato della storia recente.

Contrariamente alle prime due classi di attacco, queste ultime due sono state indicate in questa tesi come attacchi in quanto fornitori di servizi, poiché appunto si manifestano nel momento in cui, da fornitore, un nodo deve rispondere alla richiesta di un determinato servizio o di una determinata risorsa.

1.2.5 Attacchi orchestrati

Gli attacchi orchestrati sono più complessi delle classi analizzate precedentemente. In questa classe di attacco infatti, i conduttori impiegano strategie diverse, prendono di mira diversi target e assumono comportamenti dinamici nel tempo. Se il numero di attaccanti diventa significativo, questi possono addirittura alterare i valori delle metriche a loro vantaggio.

Un esempio di attacco di questo tipo è conosciuto come *oscillation attack* [21]. In questo caso i conduttori si dividono in team, ognuno dei quali assume un ruolo diverso nel tempo durante l'attacco. In un determinato istante, un team si comporta onestamente per costruirsi una reputazione positiva. L'altro si comporta disonestamente, sfruttando il sistema finché la reputazione non raggiunge un valore troppo basso. A questo punto i due team si scambiano di ruolo, cosicché il team disonesto possa farsi ripulire la propria reputazione. Chiaramente è possibile compiere attacchi con più ruoli e più comportamenti.

Individuare i nodi collusi in questo tipo di attacchi è un compito ancora più arduo. Nel grafo della rete ad esempio, si tratterebbe di individuare nodi di cluster che potrebbero anche non collaborare per lunghe finestre temporali. Dato l'elevato numero di nodi coinvolti infatti, i legami tra i singoli nodi sono meno intensi.

1.2.6 Denial Of Service

Gli attacchi di negazione del servizio (*denial of service*) cercano di sovvertire il sistema in modo che gli utenti onesti non possano più accedere al sistema di reputazione. I conduttori possono ottenere dalla negazione dei servizi di un RMS un vantaggio paragonabile a quello ottenibile alterando i valori di reputazione. Impedendo la fruizione del RMS ad un nodo onesto ad esempio, questo sarà costretto a scegliere gli agenti con cui interagire senza poter contare su valori di fiducia.

I sistemi più vulnerabili a questo tipo di attacchi sono chiaramente quelli centralizzati, in cui i conduttori possono prendere di mira le entità o le strutture centrali ad esempio sovraccaricandole. I sistemi totalmente distribuiti sono meno esposti invece a questo tipo di attacchi.

1.3 Sistemi di reputazione

Molti dei sistemi che vengono utilizzati possono essere modellati come Sistemi Multi Agente (MAS Multi-Agent System) aperti e dinamici [22]. Come discusso in precedenza, in questo tipo di sistemi ogni agente deve utilizzare servizi o risorse offerti da altri agenti al fine di conseguire un proprio obiettivo. Al momento della decisione riguardo al nodo a cui richiedere il servizio o la risorsa, l'agente non può prevedere la qualità del servizio o della risorsa offerta. In generale, la scelta del fornitore a cui rivolgersi, prevede da parte dell'agente prima il calcolo dell'affidabilità dei fornitori e poi la scelta del fornitore in base ai valori ottenuti. Gli autori di [22] suddividono i metodi impiegati dai sistemi di reputazione in quattro categorie:

- modelli basati sulla fiducia diretta;
- modelli basati sulla valutazione della fiducia da reputazione fornita da terze parti;
- modelli di valutazione della fiducia socio-cognitiva;
- modelli di valutazione della fiducia organizzativa

I sistemi di reputazione della prima categoria si basano sul meccanismo adottato dagli esseri umani quando devono scegliere un potenziale partner con cui interagire. Tipicamente infatti, in casi come questo, la scelta viene basata sull'esito delle transazioni passate. Il risultato di tali transazioni viene usato per stimare l'affidabilità di un'entità e quindi il rischio di incorrere in transazioni infruttuose. Uno dei primi sistemi di reputazione di questo tipo è il *Beta Reputation System*, proposto in [23].

In molti casi, ad esempio nel caso di applicazioni o ambiti in cui non si hanno molte interazioni, un nodo potrebbe non avere esperienze dirette con nessun altro agente. I RMS con modelli di fiducia basati su reputazione cercano di superare questo inconveniente utilizzando le informazioni ricevute da terze parti. In questo modo però, analogamente a quanto succede nel caso umano, ciascuna entità si espone al rischio di ricevere informazioni non veritiere. Per questo motivo, i sistemi di gestione della reputazione di questo tipo devono essere dotati di meccanismi di aggregazione delle informazioni che mitighino l'effetto di eventuali agenti mendaci.

I sistemi con modelli di valutazione della fiducia socio-cognitiva cercano di completare i sistemi di valutazione basati su evidenza nei casi di scarse informazioni disponibili. In questo caso, i sistemi socio-cognitivi, analizzano le caratteristiche intrinseche degli agenti e i fattori esterni per stimare i comportamenti futuri degli agenti da valutare. A titolo di esempio, SUNNY, proposto in [24] , è il primo sistema ad utilizzare un meccanismo di inferenza della fiducia basandosi sulle informazioni raccolte attraverso un social network.

Per concludere, l'ultimo gruppo di sistemi di reputazione prevede una struttura organizzativa. Una struttura di questo tipo è possibile solo in sistemi in cui è presente almeno una entità terza fidata.

1.4 Eigentrust

Eigentrust [25] è un algoritmo distribuito di gestione della reputazione per le reti P2P. L'obiettivo del suo utilizzo è minimizzare il numero di download di file inautentici. Sebbene infatti le reti P2P offrano diversi vantaggi, essi ad esempio permettono l'introduzione nel sistema di virus o file non autentici da parte di entità anonime. Nei paragrafi successivi sono illustrati i punti salienti di *Eigentrust* sia allo scopo di mettere in evidenzia le caratteristiche più importanti di un generico RMS, sia per illustrare quelle che potrebbero essere le esigenze di un simulatore.

1.4.1 Considerazioni Progettuali

Gli autori elencano alcune problematiche importanti per ogni RMS da impiegare in applicazioni P2P. La prima tra queste problematiche è il *self-policing,* cioè il sistema è autogestito e non è presente nessuna autorità centrale.

La seconda peculiarità di questo tipo di applicazioni è l'anonimità dei *peer.* Il sistema infatti dovrebbe mantenere la riservatezza sull'identità degli utenti. Anche i valori di reputazione quindi, dovrebbero essere associati ad un identificativo opaco e non ad esempio all'IP.

Un'altra problematica riguarda il profitto ai *newcomers.* Il sistema di reputazione non dovrebbe fornire profitti ai nuovi utenti. La reputazione di ciascun nodo deve essere costruita su un numero cospicuo di transazioni positive. Un utente malevolo non dovrebbe ottenere vantaggi nel rientrare nel sistema con una nuova identità.

Non meno importante, il RMS associato al sistema deve garantire il minimo *overhead* in termini computazionali, di infrastrutture, di memoria e di complessità dei messaggi.

Per concludere, il sistema di reputazione dovrebbe essere robusto contro i collettivi di nodi malevoli che cooperano per sovvertire il sistema.

1.4.2 Basic EigenTrust

Dopo aver ricevuto la risorsa o il servizio che aveva richiesto ad un nodo provider, l'agente può valutare positivamente la transazione se questa è avvenuta correttamente o negativamente se ad esempio il file scaricato non è conforme alle aspettative. In simboli:

$$tr(i,j) = \begin{cases} 1 \; se \; transazione \; positiva \; tra \; i \; e \; j \\ -1 \; se \; transazione \; negativa \; tra \; i \; e \; j \end{cases}$$

dove *i* è il nodo consumatore che sta valutando la transazione, *j* è il nodo fornitore che ha fornito il servizio ad *i*.

In più, ciascun nodo *i* può calcolare il numero di transazioni positive e negative avvenute con ciascun fornitore j e calcolarne la differenza:

$$s_{ij} = \sum tr(i,j) = sat(i,j) - \; unsat(i,j)$$

La difficoltà principale di questa tipologia di algoritmi è l'aggregazione dei valori di fiducia raccolti dai diversi nodi. Trasmettendo le informazioni sui nodi solo ad un numero ristretto di agenti infatti, ciascun nodo non avrebbe una visione

completa della rete. Al contrario, trasmettendo le informazioni ad un numero troppo elevato di nodi, si rischierebbe la congestione della rete. Eigentrust riesce ad aggregare le reputazioni di tutti i nodi della rete con il minor *overhead* in termini di complessità dei messaggi. Esso (come molti sistemi di reputazione) si basa su una sorta di "fiducia transitiva": i nodi onesti nel ruolo di fornitori di servizi, verosimilmente saranno onesti come "fornitori dei valori di reputazione". Per cui un generico *peer* avrà una buona considerazione di tutti i nodi con cui ha effettuato transazioni fruttuose e dei nodi di cui questi ultimi hanno una buona opinione.

Prima di aggregare i valori di reputazione è necessario normalizzarli. Il valore normalizzato di fiducia c_{ij} è definito in questo modo:

$$c_{ij} = \frac{\max(s_{ij}, 0)}{\sum_j \max(s_{ij}, 0)}$$

I valori così calcolati sono compresi tra 0 e 1 e sono relativi: anche conoscendo il valore di reputazione relativo ad un nodo, non è possibile stabilire il grado di onestà del nodo. Al contrario invece, i sistemi di reputazione a valori assoluti (come l'altro algoritmo oggetto delle simulazioni) permettono di avere un'idea chiara del comportamento dei nodi. Dalle considerazioni fatte in precedenza sulla transitività delle reputazioni, per aggregare i valori, ciascun nodo potrebbe semplicemente chiedere i valori di reputazione ai propri vicini e aggregarli pesandoli per la reputazione che il nodo che sta effettuando il calcolo dei vicini. In simboli:

$$t_{ik} = \sum_j c_{ij} c_{jk}$$

dove, al solito, il primo pedice indica il nodo consumatore, il secondo il nodo fornitore valutato. Da notare che il risultato di tale sommatoria è 1. Gli autori dell'algoritmo dimostrano che, anche per grandi numeri di nodi e per termini della sommatoria non relativi solo ai vicini di primo grado, tale somma converge allo stesso valore qualunque sia l'agente che sta effettuando il calcolo.

1.4.3 Considerazioni pratiche

Spesso, i primi nodi di una rete sono riconosciuti come "fidati", in quanto, trattandosi in pratica dei progettisti e degli sviluppatori del sistema, non hanno reale interesse a comportarsi disonestamente. Un modo semplice per assegnare tale valore di *fiducia a priori* (o *pretrust*) è semplicemente quello di assegnare ciascun nodo i della rete un valore $p_i = \frac{1}{|P|}$ se *i* appartiene a tale insieme e 0 altrimenti, dove |P| indica il numero di nodi *pretrusted*. Gli autori dimostrano inoltre che, calcolando il vettore $\vec{t}$ secondo la $\vec{t} = (C^T)^n \vec{p}$ (forma matriciale del calcolo di t_{ik}, con la matrice C

contenente i valori c_{ij} ricevuti da distanza *n*, moltiplicata per il vettore contenente i valori di *pretrust* di j), l'algoritmo converge più rapidamente.

La fiducia a priori viene utilizzata anche per risolvere altre due problematiche. Nel calcolo di c_{ij} , per come lo abbiamo definito in precedenza, si ottiene un valore indeterminato nel caso in cui un nodo non abbia effettuato alcun download o nel caso la differenza tra transazioni riuscite correttamente e non, risulti nulla. Utilizzando il valore di *pretrust*, il calcolo dei valori normalizzati viene modificato in questa maniera: $c_{ij} = \begin{cases} \frac{\max(s_{ij},0)}{\sum_j \max(s_{ij},0)} & se \ \ \sum_j \max(s_{ij},0) \neq 0 \\ p_j & altrimenti \end{cases}$.

Infine, un algoritmo di questo tipo deve sicuramente tenere conto dei collettivi malevoli che possono agire nella rete. Per evitare che un collettivo malevolo prenda di mira un target annullandone completamente la reputazione, gli autori hanno pensato di includere un termine additivo che dia a ciascun partecipante almeno il contributo dei nodi *pretrusted*. Quindi in definitiva otteniamo per il vettore $\vec{t}$ al passo k+1 :

$$\vec{t}^{(k+1)} = (1-a)C^T\vec{t}^{(k)} + \ a\vec{p} \ ,$$

con *a* costante minore di 1.

1.4.4 Secure Eigentrust

Nella trattazione dell'algoritmo, abbiamo finora tralasciato il problema di come ogni nodo potrebbe reperire i valori c_{ij} prima di effettuare il calcolo dei valori di reputazione. Deciso che l'architettura del sistema di reputazione in questione debba essere di tipo distribuito, è ovviamente da escludere che ogni nodo calcoli la propria reputazione. In questo modo infatti, ciascun fornitore potrebbe facilmente riportare falsi valori di reputazione per ottenere benefici. Scelto che il valore di reputazione di un *peer* debba essere calcolato da un altro nodo della rete, si pone il problema di come quest'ultimo potrebbe riportare valori di reputazione più bassi per danneggiare il nodo di cui sta effettuando il calcolo. È buona prassi quindi che il calcolo venga effettuato da più nodi e che il valore finale venga o calcolato attraverso la media dei singoli valori o attraverso un meccanismo di *majority voting.*

Il risultato di queste considerazioni conduce gli autori all'algoritmo *Secure Eigentrust*. In questa versione finale dell'algoritmo, all'ingresso di un nuovo nodo nella rete, vengono associati a quest'ultimo uno o più *score managers*, il cui compito è appunto calcolare i valori di reputazione dei *nodi figlia* assegnatigli. L'associazione

degli *score managers* avviene attraverso l'uso di una DHT (*Distributed Hash Table)* quali ad esempio CAN o Chord. Quando un nodo della rete necessità dei valori di reputazione di un altro agente, può richiederli agli opportuni *score managers* utilizzando la funzione hash impiegata dal sistema.

Gli autori di Eigentrust sottolineano tre aspetti importanti di questa versione sicura:

- *anonimità*: gli score manager non conoscono l'identità dei nodi di cui stanno calcolando la reputazione. In questo modo vengono sicuramente scoraggiati attacchi che prevedono la modifica dei valori di reputazione (Es: *Slandering* e *Promoting*);
- *randomizzazione:* quando entra nella rete, il nodo non può scegliere le coordinate dello spazio hash in cui posizionarsi. Di conseguenza, è scongiurato il caso in cui un nodo possa fare in modo di calcolare la propria reputazione;
- *ridondanza:* attraverso l'utilizzo di funzioni hash multi dimensionali, a ciascun nodo vengono associati più *score managers*, come discusso in precedenza.

1.5 RMS caso di studio

In [1] [2] [3] è presentato il secondo sistema di reputazione ampiamente testato. Nel capitolo dedicato alle simulazioni saranno discusse ulteriori varianti dell'algoritmo considerato.

Il meccanismo di valutazione della fiducia locale prevede che ogni nodo calcoli la percentuale di transazioni andate a buon fine con gli altri nodi. Se *i* è il nodo che sta effettuando il calcolo e *j* è un generico nodo con cui *i* ha avuto transazioni:

$$lt_{ij} = \frac{sat(i,j)}{sat(i,j) + unsat(i,j)}$$

dove *sat(i, j)* indica il numero di transazioni avvenute correttamente tra *i* e *j*, *unsat(i,j)* indica transazioni di cui il nodo *i* non è soddisfatto. La *local trust* influenza la reputazione locale (che *i* ha di *j*), secondo la:

$$lr_{ij} = \alpha * lt_{ij} + (1 - \alpha) * r_{ij}(t - 1)$$

con $\alpha \in [0,1]$ costante utilizzata per pesare la fiducia locale al passo t rispetto alla reputazione globale al passo t – 1. La reputazione globale all'istante t è calcolata secondo la:

$$r_{ij}(t) = (1-\beta) * lr_{ij}(t) + \beta \frac{\sum_{k \in K} r_{ij}(t-1) * r_{kj}(t-1)}{\sum_{k \in K} r_{ik(t-1)}}$$

dove β è una costante compresa tra 0 e 1, usata per pesare la fiducia locale rispetto alla reputazione ottenuta dai vicini e K è l'insieme dei vicini "affidabili", ossia quei vicini il cui valore di reputazione è maggiore di una terminata soglia τ. In simboli: $K = \{ k : r_{ik}(t-1) \geq \tau \}$. In sintesi, la media pesata dei valori ottenuti dai vicini viene sommata con peso β al valore di fiducia locale con peso 1-β. Nel capitolo dedicato alle simulazioni saranno illustrati i risultati ottenuti da questo algoritmo per i tipi di attacco prima descritti e si confronteranno tali risultati con quelli di tre varianti di questo algoritmo. Sarà mostrato anche un rapido confronto con Eigentrust nella versione sicura.

2 Progettazione

2.1 Obiettivi

L'obiettivo del progetto è la creazione di una piattaforma parallela che permetta al progettista di un sistema di reputazione di creare uno scenario di simulazione e di effettuare simulazioni che possano essergli utili per valutarne le performance. Più precisamente, riguardo lo scenario di simulazione, l'ambiente di sviluppo permetterà di variare:

- il numero di nodi della simulazione, non necessariamente costante;
- la connessione tra tali nodi;
- il comportamento dei nodi;
- il numero di passi di simulazione;
- l'algoritmo di reputazione da testare.

2.2 Glossario e Acronimi

SSH Abbreviazione di *Secure Shell*, protocollo per la comunicazione remota cifrata tra *host* di una rete.

MPI Acronimo di *Message Passing Interface*, è un protocollo per la comunicazione tra nodi di un cluster di macchine.

Shell Componente del sistema operativo che permette all'utente di impartire comandi. Nel caso del software proposto la *shell* è di tipo testuale e i comandi vengono impartiti tramite riga di comando.

Terminale Può intendersi come sinonimo di Shell.

Log Con il termine log vengono indicati i file in cui i nodi registrano i valori calcolati.

2.3 Sistema Proposto

2.3.1 Requisiti funzionali

Al fine di creare un sistema completo e funzionale, sono state pensate le seguenti funzionalità:

- definizione interattiva del comportamento dei nodi, con parametri pseudocasuali;
- creazione interattiva della topologia con parametri pseudocasuali;
- simulazione con file log con gli output della simulazione.

2.3.2 Requisiti non funzionali

Il software finale dovrà essere utilizzato da professionisti del mondo dell'informatica che non necessariamente possiederanno competenze avanzate di programmazione e, ancora di più, di programmazione distribuita. Requisito fondamentale del progetto è quindi che il sistema sia semplice da utilizzare dal punto di vista del programmatore del RMS e che lo strato della programmazione parallela sia "trasparente". Il sistema quindi deve offrire una serie di interfacce di alto livello che semplifichino la programmazione di basso livello.

Fondamentali, sono la versatilità e la flessibilità delle simulazioni. Il progettista deve essere in grado di effettuare simulazioni in scenari diversi.

Non trascurabile è l'efficienza del sistema. I sistemi di reputazione infatti, soprattutto in certi contesti simulativi, possono diventare molto onerosi in termini di carico computazionale. È quindi importante trovare il giusto compromesso tra flessibilità nelle simulazioni e performance del simulatore.

2.3.3 Documentazione

Tipicamente i documenti previsti per un software sono:

- RAD (*Requirements Analysis Document*): descrive il sistema in termini di requisiti funzionali e non funzionali ed è destinato a clienti e sviluppatori;
- SDD (*Software Design Document*): descrive l'architettura del sistema ed è rivolto agli sviluppatori;
- ODD (*Object Design Document*): descrive l'architettura del sistema in termini di oggetti ed è rivolto anch'esso agli sviluppatori.

Tali documenti però non si prestano al software proposto in questo lavoro di tesi. Di seguito saranno comunque mostrati alcuni diagrammi tipici di Ingegneria del Software.

2.3.4 Considerazioni sull'hardware

Si presuppone che il progettista del sistema di reputazione che si trovi ad utilizzare il simulatore disponga di un computer con elevate capacità computazionali. Non sono previsti requisiti minimi, tuttavia è consigliabile l'impiego del software

presentato su cluster di macchine ad elevate prestazioni. I requisiti consigliati variano comunque in base alla dimensione della popolazione di nodi da testare. Nel caso di utilizzo su cluster di macchine è indispensabile che le queste siano dotate di scheda di rete e che siano mutuamente autenticate su SSH. Per simulazioni con popolazioni dell'ordine del centinaio di nodi e sistemi di reputazione non particolarmente complessi, un computer di medie capacità è sufficiente.

2.3.5 Pseudo requisiti

È esplicitamente richiesto l'utilizzo del linguaggio di programmazione C++ e della libreria MPICH per la comunicazione tra processi. D'altra parte, il linguaggio C++ offre vantaggi importanti quali l'efficienza e il paradigma ad oggetti, particolarmente indicati per questo tipo di applicazioni. Di MPICH si parlerà alla fine del capitolo.

2.3.6 Modelli del sistema: scenari

A rigore, il generatore della topologia è un modulo software distinto dal simulatore. I seguenti "pseudo scenari" costituiscono tuttavia importanti esempi di utilizzo della piattaforma di simulazione.

Nome Scenario	**Simulazione con configurazione esistente**
Attore	Vincenzo: RMS Designer
Descrizione	1. Vincenzo avvia il simulatore digitando da terminale il comando mpiexec –n 100 nodelist.txt topology.txt 2. Il sistema esegue le simulazioni con i parametri specificati sui file configurazione e da terminale 3. Il sistema salva i dati output sui log 4. Vincenzo può usare i file log per valutare il sistema di reputazione

Nome Scenario	**Simulazione con topologia semplice**
Attore	Vincenzo: RMS Designer
Descrizione	1. Vincenzo crea un file chiamato nodelist.txt 2. Vincenzo scrive nel file, su due righe distinte: id 1 (1 *slander* 0) < 2 >, a capo: id 2 3. Vincenzo salva il file e crea il file topology.txt, dove scrive: 1, a capo 2 4. Vincenzo avvia la simulazione come nel caso precedente.

Nome Scenario	**Simulazione con topologia random**
Attore	Vincenzo: RMS Designer
Descrizione	1. Vincenzo avvia il software per la generazione della topologia 2. Il software chiede di inserire il numero di nodi 3. Vincenzo inserisce il numero 100 4. Il sistema chiede il nome del comportamento attaccante da simulare 5. Vincenzo inserisce il nome: *slander*. 6. Il sistema riconosce il nome e chiede percentuale, *beginning step* e cooperatività 7. Vincenzo inserisce 20, 50 e 0,9 e preme invio 8. Il sistema chiede la lunghezza media dell'elenco target 9. Vincenzo inserisce 10 e preme invio.

	10. Il sistema genere i file topology.txt e nodelist.txt 11. Vincenzo avvia la simulazione come nei casi precedenti.

Nome Scenario	**Simulazione con topologia random e comportamento personalizzato**
Attore	Vincenzo: RMS Designer
Descrizione	1. Vincenzo esegue i primi quattro passi del caso precedente. 2. Vincenzo inserisce *myslander* 3. Il sistema nono riconosce il comportamento, quindi chiede un numero arbitrario di parametri. 4. Vincenzo inserisce 1 10 20 stop e preme invio. 5. Il generatore crea i file 6. Vincenzo avvia la simulazione

Come si può notare dagli scenari illustrati, il nostro sistema prevede un solo attore: il progettista o colui che vuole valutare le prestazioni del sistema di reputazione. Non sono previste altre figure professionali o altre entità (quale ad esempio potrebbe essere il DBMS in altre applicazioni). Il comando per l'avvio della simulazione sarà discusso in uno dei successivi paragrafi, dedicato alla libreria utilizzata per la programmazione concorrente. Mentre la struttura dei file con la definizione dei nodi e della topologia sarà illustrata nel capitolo successivo.

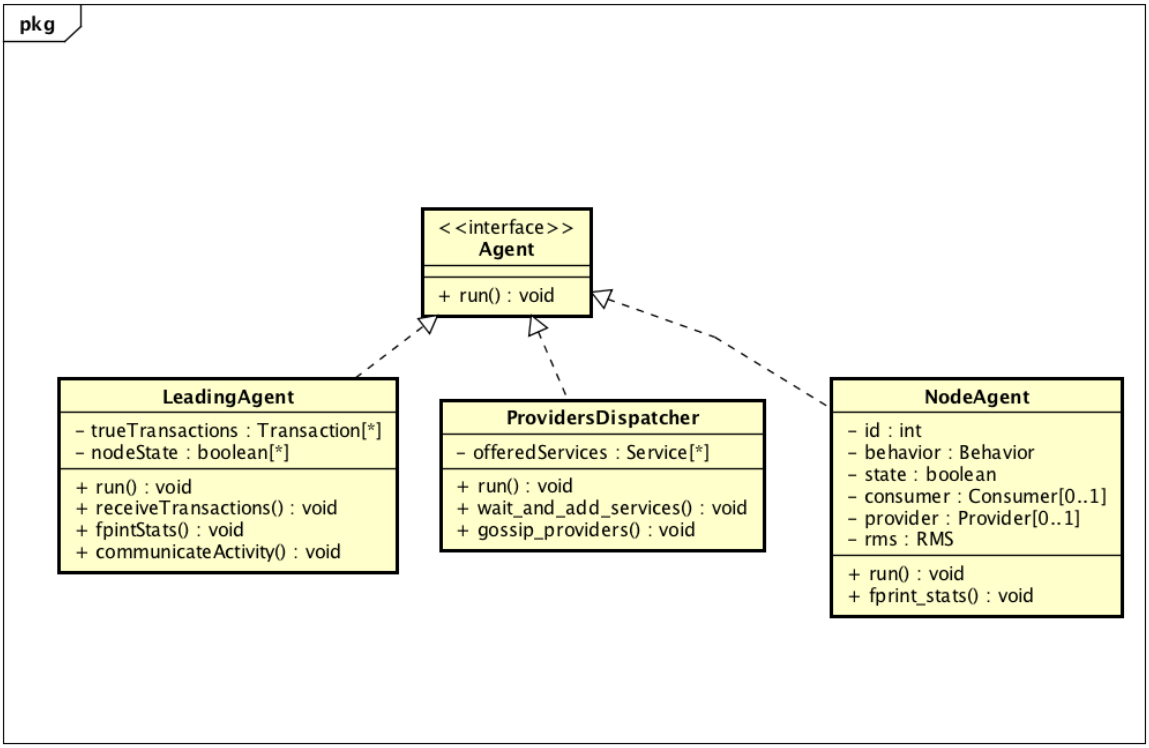

Figura 2-1 Relazione tra gli agenti

2.4 Diagrammi delle classi

In questo paragrafo vengono presentati alcuni diagrammi delle classi. I diagrammi mostrati sono "snelliti" per facilitarne la comprensione, stessa motivazione per cui si è preferito spezzare il singolo diagramma completo in diagrammi più semplici.

2.4.1 Gerarchia degli Agenti

Il primo diagramma (Figura 2-1) mostra la relazione tra le *entity* che rappresentano gli agenti. La superclasse astratta *Agent* (in C++, diversamente da altri linguaggi come Java, non esiste il concetto di interfaccia) viene implementata dalle sottoclassi *LeadingAgent*, *ProvidersDispatcher* e *NodeAgent,* che implementano il metodo virtuale *run()*. Ciascuno di questi agenti ha infatti un ciclo diverso e il metodo *run()*, come il nome può far intuire, implementa proprio il ciclo di vita. L'entità *LeadingAgent* conserva la mappa delle transazioni vere (non modificate da eventuali comportamenti malevoli) e lo stato che dovrà assumere ciascun *NodeAgent* per ogni passo della simulazione. I restanti metodi indicati, servono appunto per la raccolta delle transazioni effettive, per il salvataggio su file di valori ad esse legate e al coordinamento degli agenti. La sottoclasse *ProvidersDispatcher* modella il processo

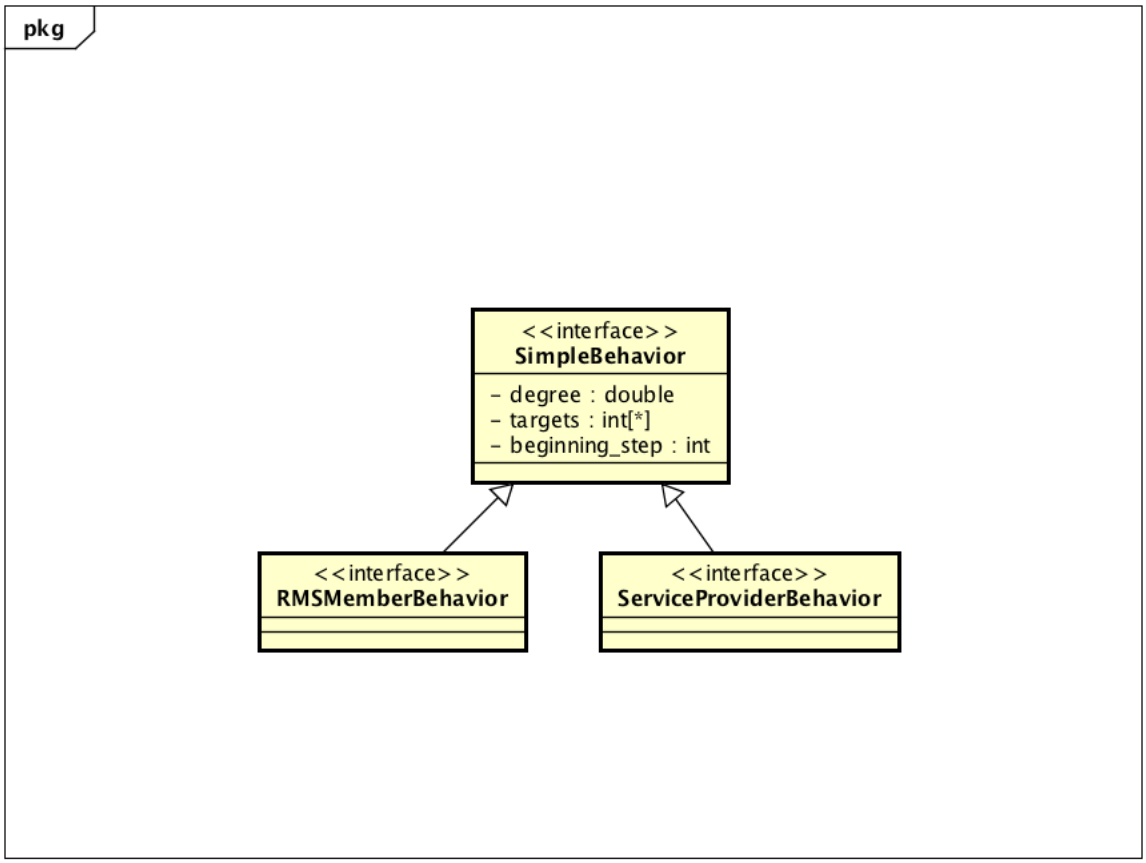

Figura 2-2 Gerarchia tipi di comportamento

che si occupa di conservare l'elenco dei servizi offerti dai relativi provider (nella mappa *offeredServices*) e diffonderlo ai consumatori attraverso il metodo *gossip_providers()*. La mappa viene aggiornata ad ogni passo attraverso il metodo *wait_and_add_services()*. L'ultimo tipo di agente conserva le informazioni dei nodi. In particolare ciascun *nodeAgent* ha un id che può cambiare nel tempo, un comportamento (*Behavior*) che sarà dettagliato nei prossimi diagrammi, si comporterà da consumatore e/o da provider, ha uno stato che, come indicato in precedenza, viene gestito dal *leadingAgent* e infine, un oggetto *RMS* per il calcolo dei valori di reputazione, in accordo con la natura distribuita discussa all'inizio della trattazione. Come per la classe che modella il leader, il metodo *fprintStats()* si occupa del salvataggio dei dati su file.

2.4.2 Gerarchia dei Behaviors

I prossimi diagrammi illustrano le relazioni tra le classi che implementano i comportamenti. Il diagramma in Figura 2-2, in particolare, mostra che i due tipi di comportamento estendono una superclasse semplice contenente le informazioni essenziali per ogni tipo di comportamento. Tali informazioni sono essenzialmente il grado di cooperatività, l'insieme di target (ossia i nodi che saranno vittima

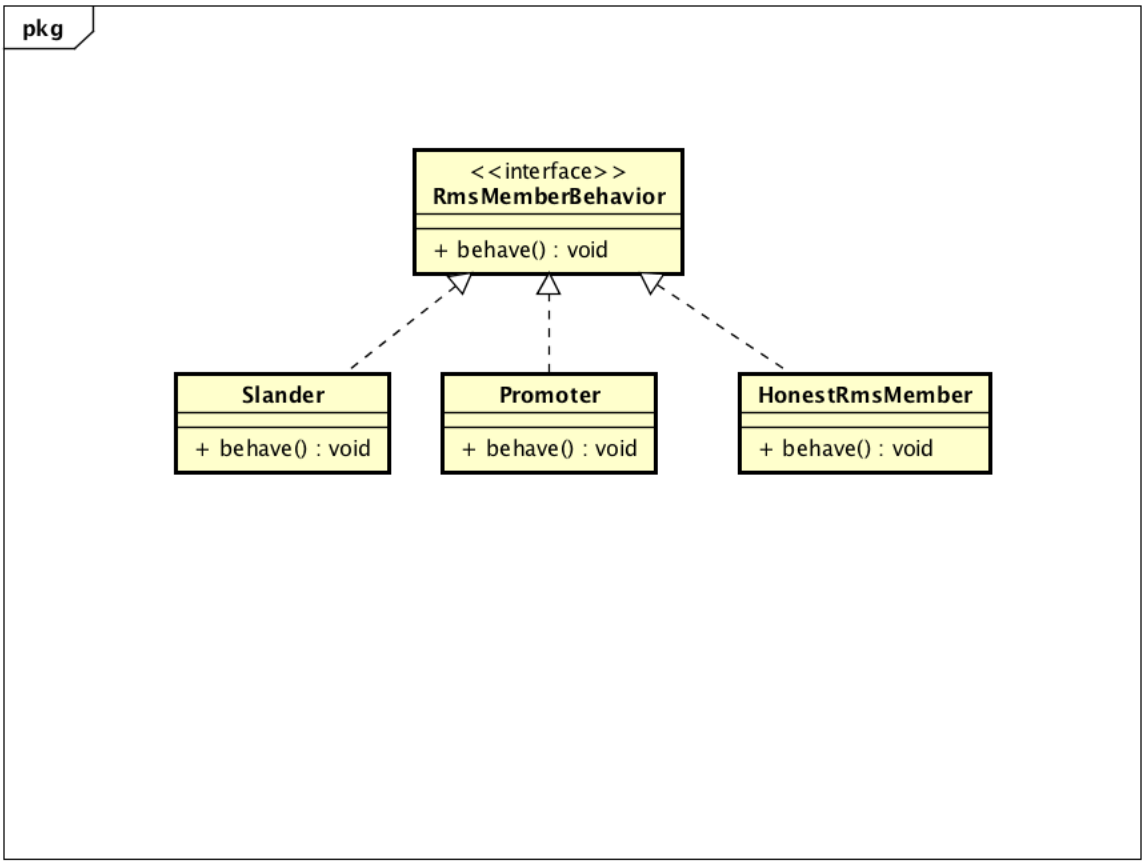

Figura 2-3 Gerarchia comportamenti di tipo membro RMS

dell'attacco) e il "beginning" step, ossia lo step a partire dal quale l'attaccante inizierà a comportarsi malevolmente.

2.4.3 Relazione Behavior membri RMS

Il diagramma in Figura 2-3 mostra la gerarchia di base per l'implementazione dei comportamenti come membro di un RMS. Si può notare l'interfaccia *RmsMemberBehavior* con il metodo virtuale astratto *behave(),* implementato poi dalle classi *Slander*, *Promoter* e *HonestRmsMember.* Il comportamento onesto, qui come nella controparte relativa al comportamento da provider, è quella assunta di default.

2.4.4 Relazione behavior Provider

In analogia al caso precedente, il diagramma in Figura 2-4 mostra le relazioni tra i comportamenti di tipo Provider. Analogamente al caso precedente, l'interfaccia *ServiceProviderBehavior* viene implementata da qualunque comportamento di tipo *ProviderBehavior*. Tra quelli implementati, oltre ai tipi di attacco discussi all'inizio della trattazione, troviamo il comportamento onesto, cioè il comportamento per cui il

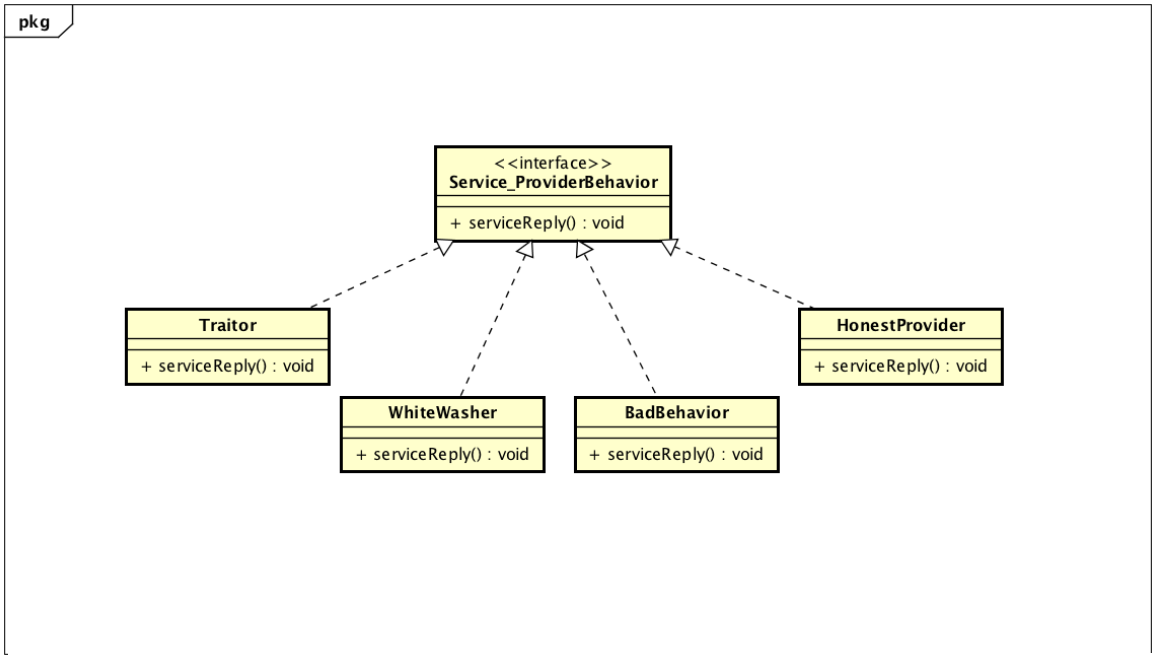

Figura 2-4 Gerarchia comportamenti di tipo fornitori

provider fornisce esattamente il servizio richiesto e il *BadBehavior*, che risponde malevolmente proporzionalmente al suo grado di cooperatività. Sia per i comportamenti di tipo *RmsMember*, sia per i comportamenti di tipo *ProviderBehavior*, il designer del sistema di reputazione può crearsi sottoclassi particolari. L'idea di definire le interfacce infatti nasce anche per dare maggiore libertà al progettista.

2.4.5 Composizione behavior

In accordo con quanto sarà descritto a livello funzionale, la relazione tra il comportamento complessivo e i due tipi di comportamento non può che essere una composizione. In particolare, come mostrato in Figura 2-5, un *Behavior* è composto esattamente da un *ServiceProviderBehavior* e da uno o più *RMSMemberBehavior*.

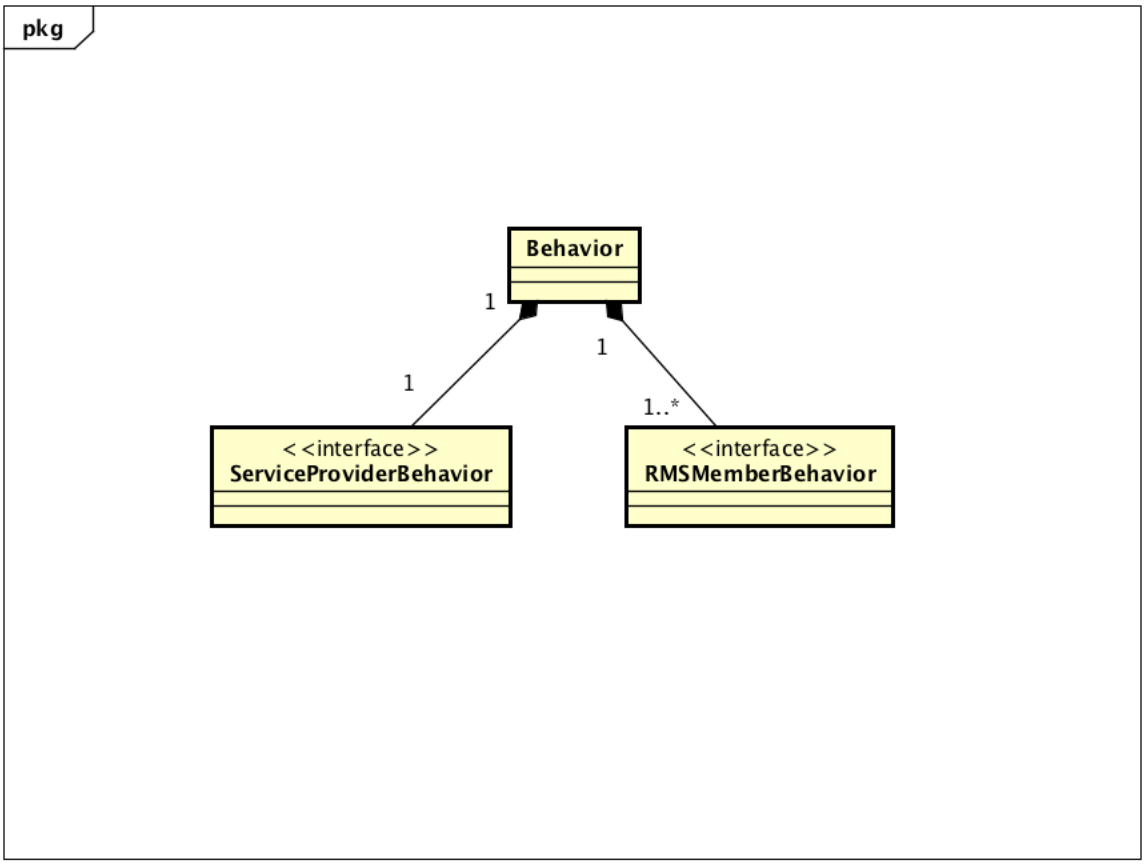

Figura 2-5 Classe Comportamento come aggregazione delle due classi specifiche

2.4.6 Diagramma completo

Per finire, l'ultimo diagramma mostrato è quello complessivo, in cui vengono illustrate le relazioni tra le principali classi dell'ambiente di simulazione (Figura 2-6). Si nota in questo diagramma complessivo che il simulatore è composto da un leader, da un *dispatcher* dei provider e da uno o più nodi agente. Il *leadingAgent*, il consumatore, il *ProvidersDispatcher* e il RMS usano i metodi statici della *MPIClass* per comunicare. La classe *Consumer* contiene inoltre lo storico delle transazioni e i metodi necessari per la scelta del provider (secondo le modalità descritte in precedenza), per la richiesta della lista dei provider e per la richiesta effettiva del servizio. La classe simulatore mantiene il passo corrente di simulazione.

Il metodo *serviceSelection()* della classe *Consumer* prende come input uno dei seguenti parametri:

- PROBABILITY_MODE
- ABSOLUTE_MODE

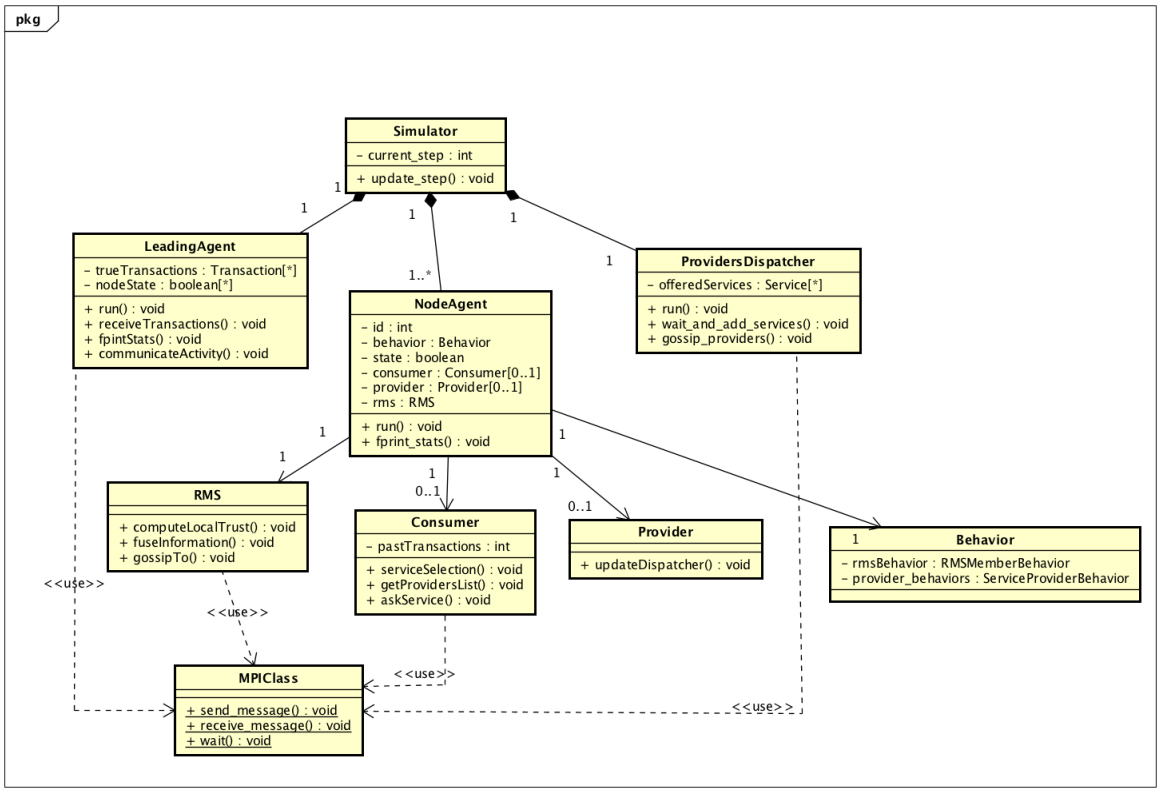

Figura 2-6 Diagramma finale semplificato

- RANDOM_MODE.
- ALL_PROVIDERS_MODE

Con i primi tre parametri, il metodo restituisce un unico provider. In particolare, nella prima modalità, viene restituito un provider in maniera casuale ma proporzionalmente ai corrispondenti valori di reputazione. Con il parametro ABSOLUTE_MODE, il metodo restituisce il provider con il più alto valore di reputazione, mentre con il terzo viene restituito un provider in maniera assolutamente pseudocasuale. Infine, l'ultimo metodo restituisce tutti i provider.

2.5 Diagrammi di sequenza

Con le stesse premesse fatte per i diagrammi delle classi, in questo paragrafo sono illustrati i diagrammi di sequenza per le funzionalità più importanti del simulatore.

2.5.1 Ciclo di vita di un agentNode

Nel diagramma che segue in Figura 2-7, viene illustrato il ciclo di vita di un generico nodo della rete.

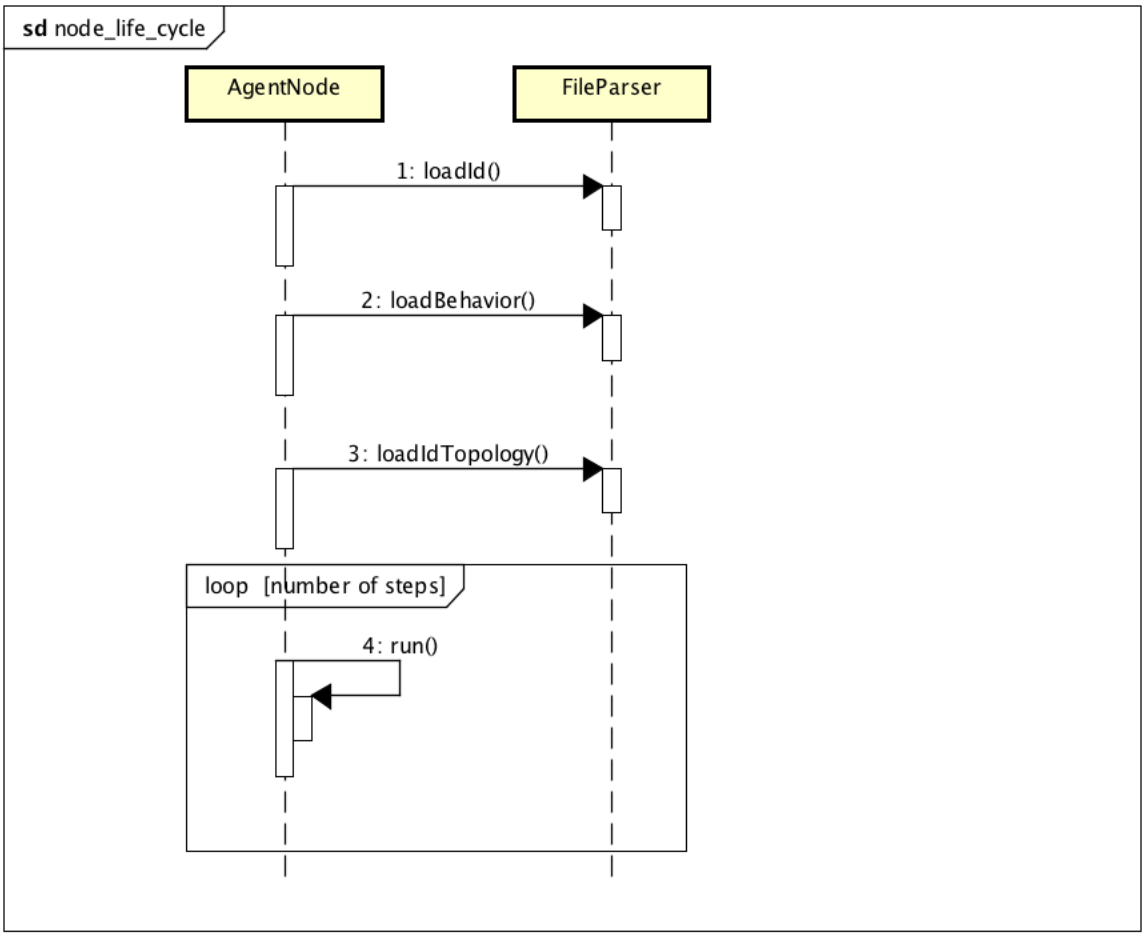

Figura 2-7 Ciclo di vita di un generico nodo

Come si nota dal diagramma, un nodo appena creato carica da file il proprio id e il proprio comportamento. Dopo aver caricato i parametri relativi a sé stesso, ciascun nodo carica da file le informazioni relative ai propri vicini, cioè eventuali agenti con cui ha già scambiato servizi prima dell'avvio della simulazione. L'idea, in questo caso, è che il designer potrebbe voler non assegnare un valore di default ma simulare un caso in cui alcune transazioni si siano già verificate. Successivamente, ciascun nodo simula un numero arbitrario (definito come parametro della simulazione) di passi.

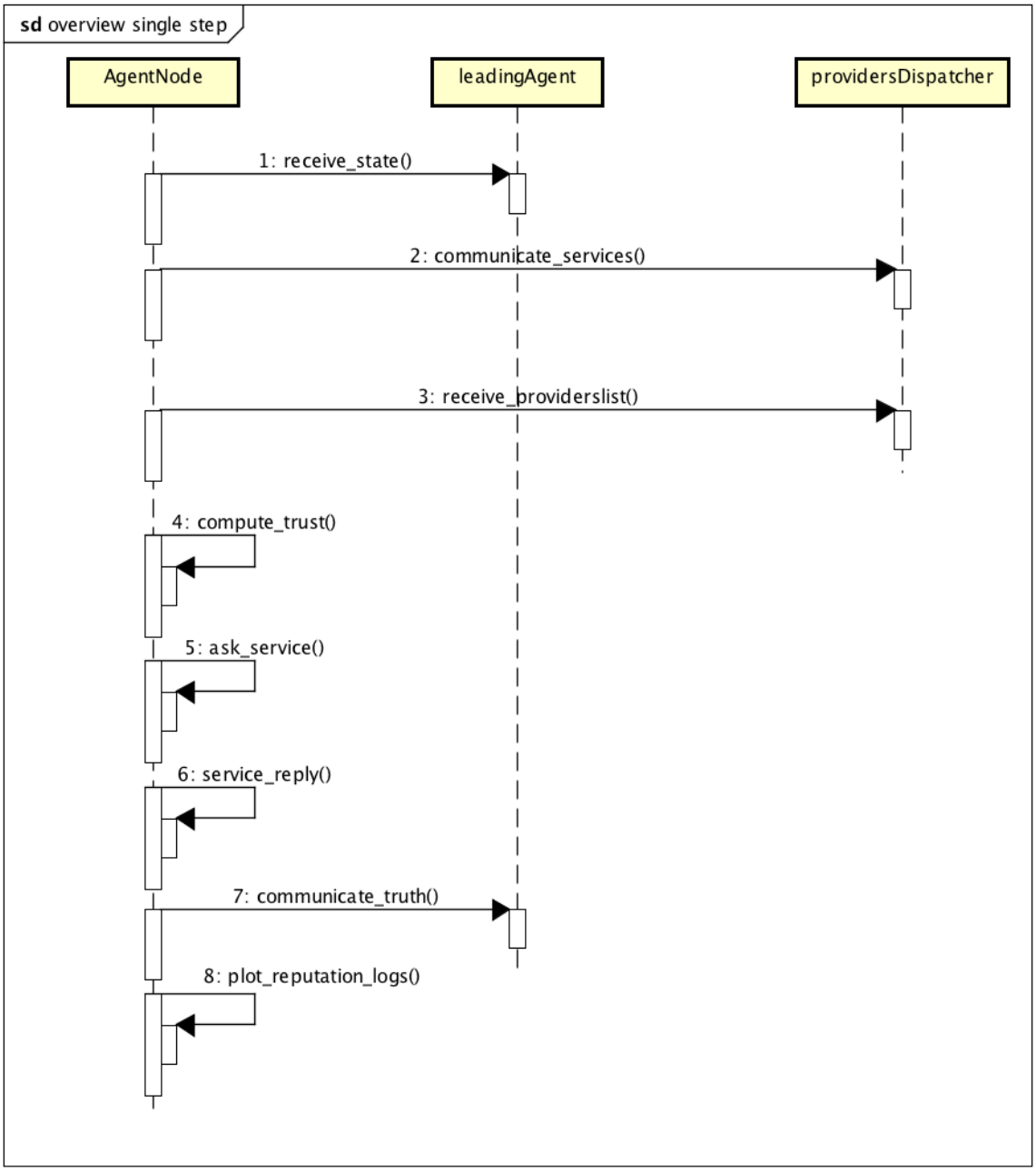

Figura 2-8 Overview nodo singolo step

2.5.2 Dettaglio del singolo passo di un nodeAgent

Il prossimo diagramma mostra più nel dettaglio le interazioni che avvengono in ogni singolo passo della simulazione. Si nota dalla Figura 2-8 che all'inizio di ogni step, il nodo riceve dal leader l'informazione riguardante lo stato che il nodo dovrà avere al passo corrente. In questo modo il leader può coordinare i nodi "attivandoli" e "disattivandoli". Come passo successivo, il nodo, in quanto provider di servizi, comunica al *dispatcher* i servizi offerti e, in quanto consumatore, riceve da questo la

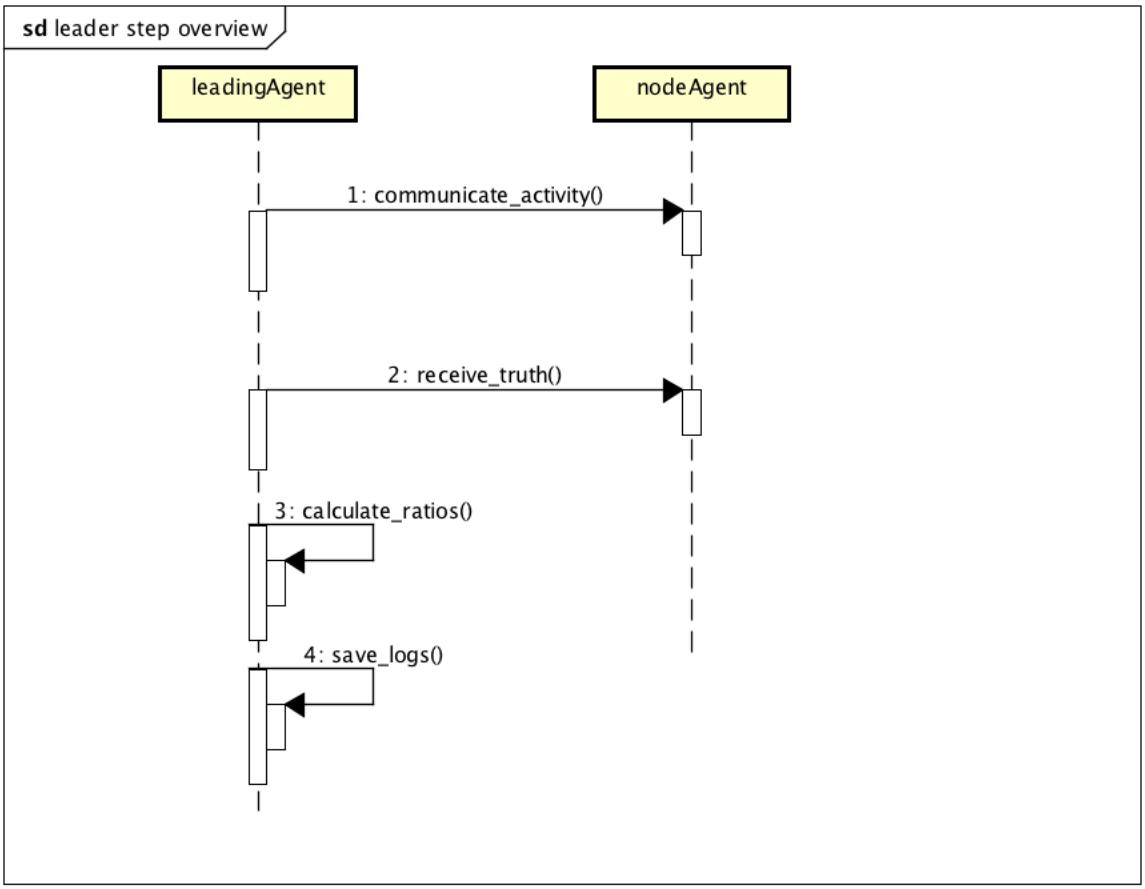

Figura 2-9 Overview singolo step del leadingAgent

lista dei servizi e dei provider presenti nella rete. Conoscendo tale elenco, il consumatore può calcolare i valori di reputazione dei provider (sarà scelta del tester se calcolare i valori di tutti i nodi o solo dei fornitori…). Sulla base di tali valori il consumatore può scegliere il provider o i provider a cui richiedere i servizi. In una simulazione realistica, un consumatore richiederebbe il servizio ad un solo provider. Tuttavia, al fine di rendere più evidenti elementi di forza e vulnerabilità, il designer può scegliere in che modo ciascun consumatore deve scegliere i provider: può farlo in maniera casuale, proporzionale al valore di reputazione del fornitore o sceglierli tutti. Come provider, ciascun nodo risponderà alle richieste che riceve. Infine, come consumatore si occuperà di comunicare al leader la verità sulle transazioni e di salvare i logs sulle reputazioni calcolate.

2.5.3 Dettaglio singolo passo del leadingAgent

Il diagramma in Figura 2-9 mostra la sequenza di azioni del nodo leader o coordinatore. In accordo con quanto descritto in precedenza a proposito dei generici nodi della rete, la prima azione compiuta dal *leadingAgent* è la comunicazione dello stato di attività ai singoli nodi. Come passo successivo, il leader raccoglie i dati provenienti dai nodi consumatori sulle transazioni. Attraverso tale informazioni il

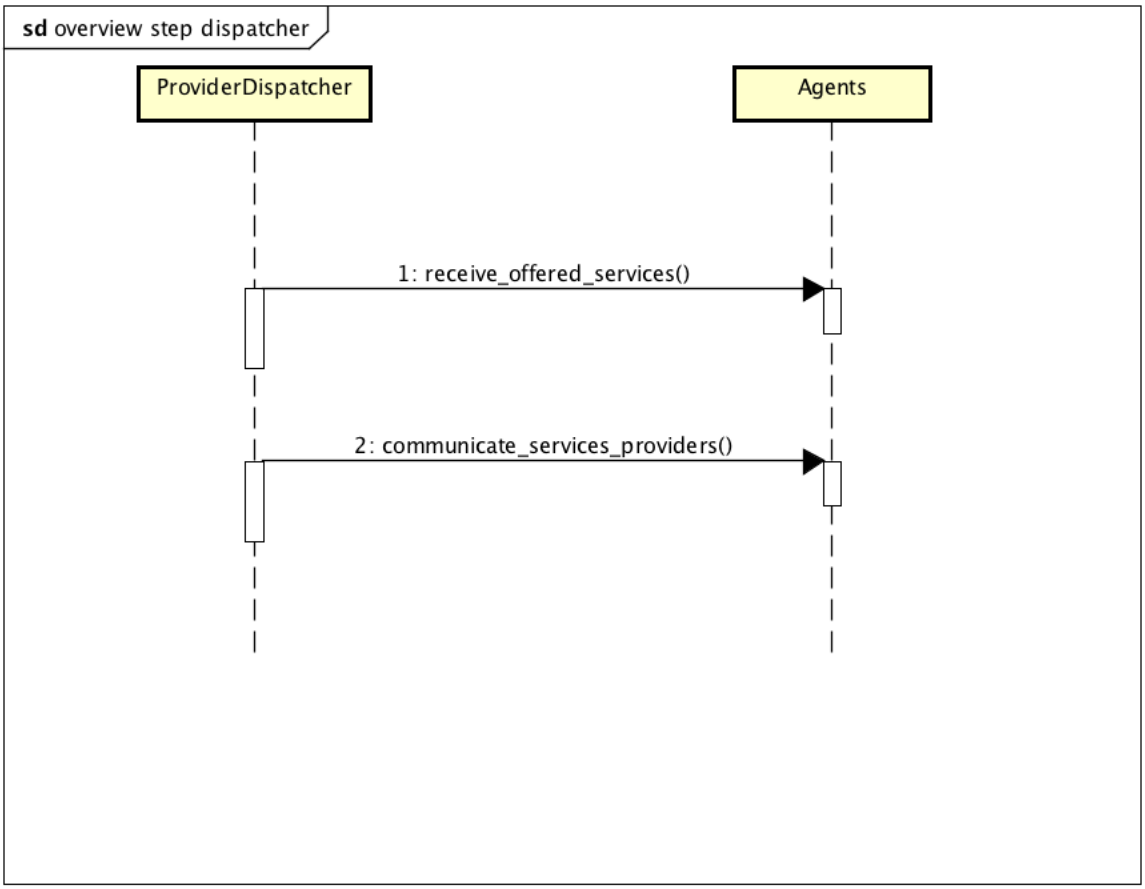

Figura 2-10 Overview singolo providersDispatcher

leader può calcolare e salvare sui file di log il rapporto tra transazioni andate a buon fine e totale delle transazioni.

2.5.4 Dettaglio singolo passo del providersDispatcher

Anche il flusso di esecuzione del *dispatcher* è abbastanza semplice. Esso infatti si occupa semplicemente di ricevere dai singoli nodi l'elenco dei servizi offerti e comunicare ad ogni nodo l'elenco dei servizi e relativi providers. La sequenza delle azioni è illustrata in Figura 2-10

2.5.5 Esempio calcolo reputazione

Gli ultimi diagrammi illustrano invece il flusso di azioni inerenti al sistema di reputazione vero e proprio. Il primo di questi diagrammi, mostrato in Figura 2-11, illustra un esempio di calcolo dei valori di reputazione. Tale sequenza infatti varia da

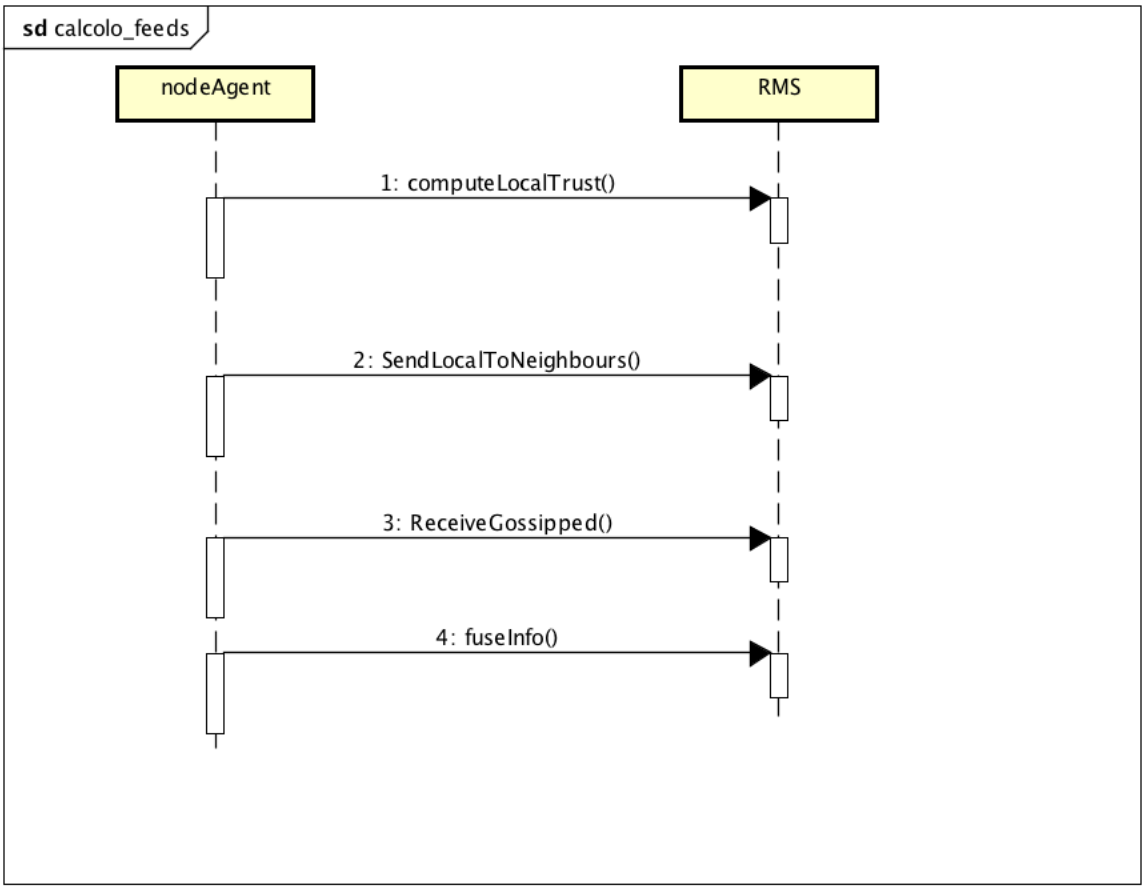

Figura 2-11 Esempio di calcolo valori di reputazione

sistema di reputazione a sistema di reputazione, con differenze anche profonde. Il flusso mostrato è molto simile a quello che avviene nel RMS del caso di studio. Nell'esempio illustrato vediamo che inizialmente ciascun nodo calcola un proprio valore di fiducia locale basandosi sulle transazioni avvenute. I valori ottenuti vengono poi trasmessi ai propri vicini e, simmetricamente, vengono integrati con i valori ricevuti da questi ultimi, in modo che ciascun nodo abbia una visuale più ampia di quella che avrebbe basandosi unicamente sulle transazioni dirette. Queste operazioni vengono eseguite invocando gli opportuni metodi. In particolare, i metodi per l'invio e la ricezione dei feedback sono stati implementati nella classe astratta *RMSInterface*, mentre i metodi per il calcolo dei valori di reputazione locali e per l'aggregazione delle informazioni devono necessariamente essere implementati dal designer del sistema di gestione della reputazione, in quanto appunto variabili da algoritmo ad algoritmo.

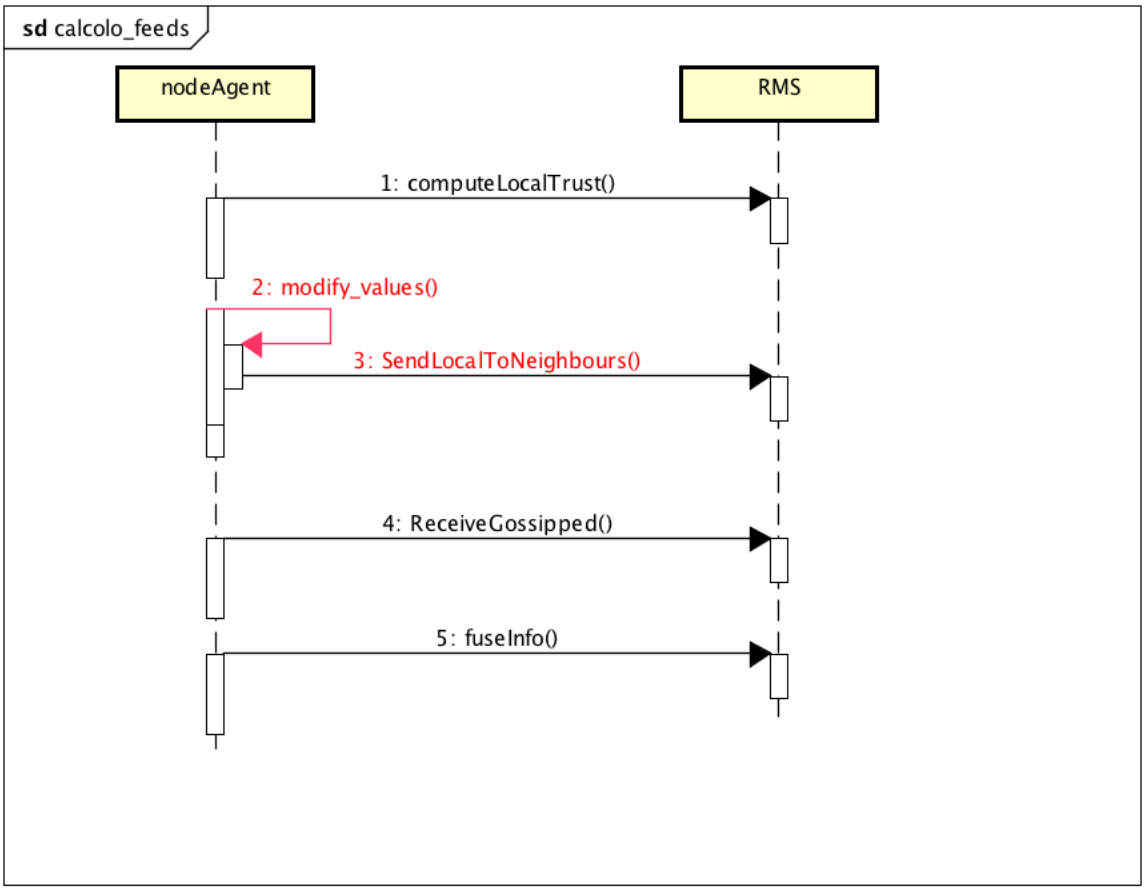

Figura 2-12 Attacco da parte di membri RMS

2.5.6 Esempi sequenze di attacco

I successivi diagrammi illustrano invece quello che avviene nel caso di attacco da parte di eventuali nodi malevoli. Nel primo (**Errore. L'origine riferimento non è stata trovata.**) dei diagrammi vediamo un attacco da membro RMS, in cui prima di inviare ai propri vicini i valori locali, l'attaccante modifica i valori di reputazione dei target. In un attacco di *slandering* tali valori vengono impostati al minimo valore possibile per il sistema di reputazione sotto simulazione, mentre nel caso di *promoting* saranno impostati al valore massimo.

Il secondo diagramma, in Figura 2-13, mostra invece uno dei due attacchi da service provider descritti in questa tesi: l'attacco del traditore. In questo caso, se il consumatore da cui proviene la richiesta è una sua vittima, il provider risponderà malevolmente se è nella fase antisociale.

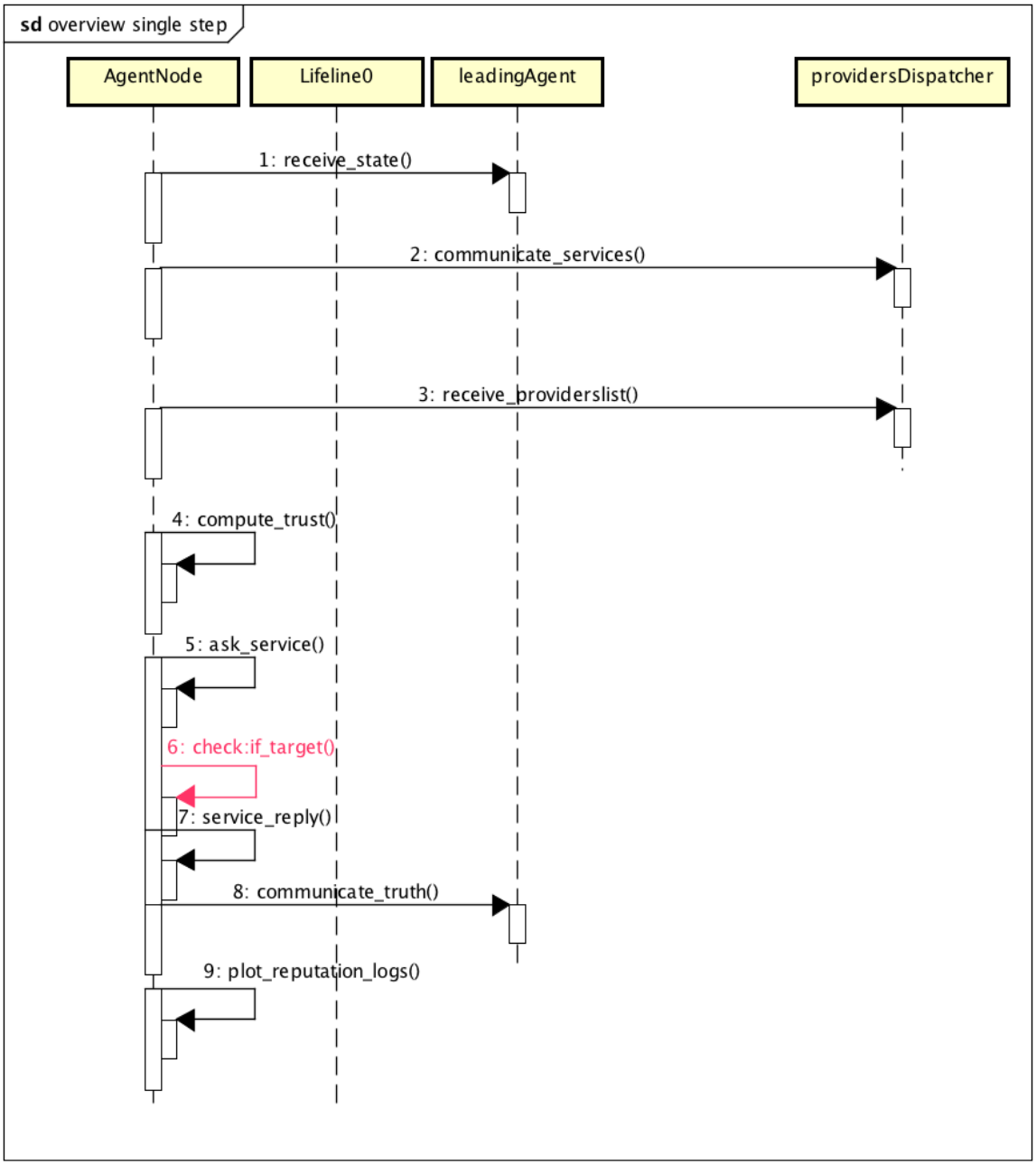

Figura 2-13 Attacco del traditore

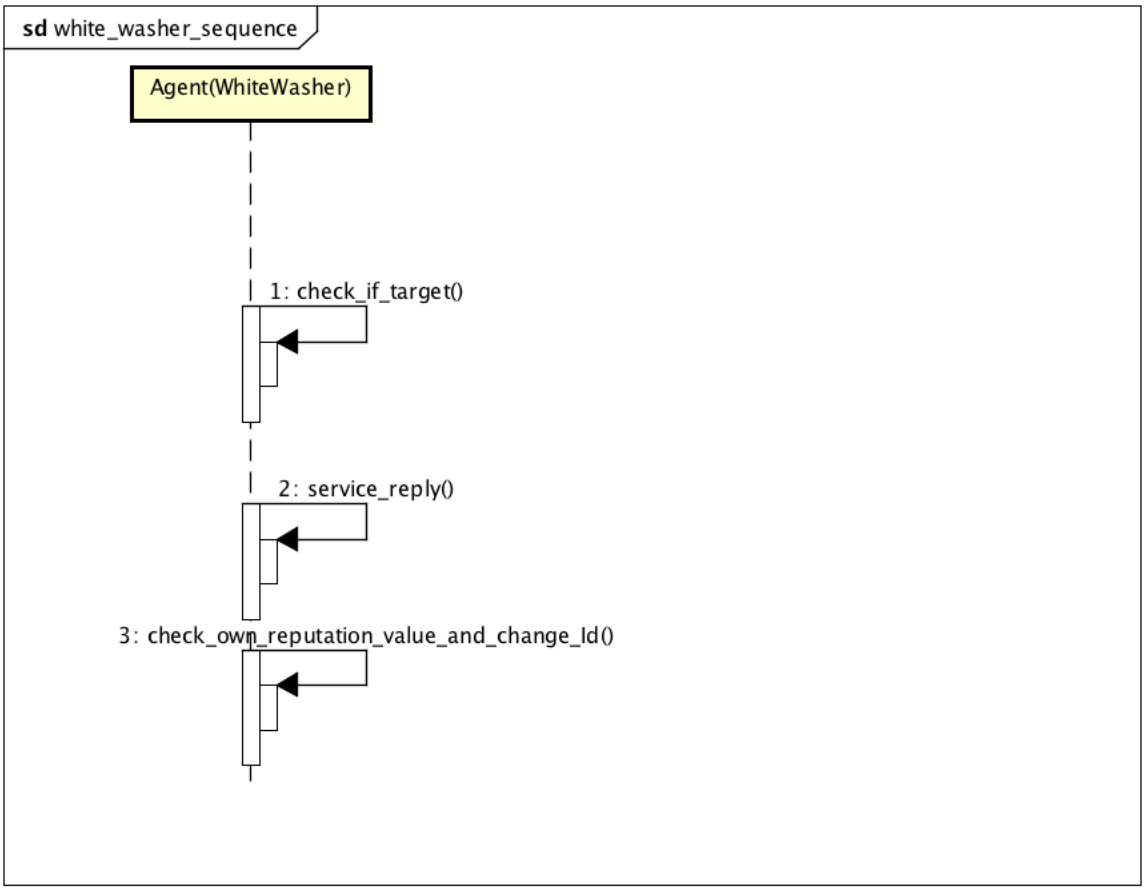

Figura 2-14 Attacco del Whitewasher

L'ultimo diagramma di sequenza,riguarda invece l'ultimo tipo di attacco tra quelli implementati nella libreria del simulatore proposto, ossia il *Whitewashing*. È possibile notare dal diagramma parziale mostrato in Figura 2-14 che, dopo aver fornito il servizio al consumatore, tenendo eventualmente in considerazione che il consumatore da cui proviene la richiesta è un nodo target, il *Whitewasher* controlla il proprio valore di reputazione e, se al disotto di una determinata soglia (parametrica), cambia id.

2.5.7 Mock-ups

Come per il generatore della topologia, l'interazione tra l'utente e il simulatore avviene tramite di riga di comando. Dopo aver configurato i file, o manualmente o tramite il generatore allegato, l'interazione con il sistema da parte dell'utente prevede semplicemente il comando di esecuzione del programma con relativi parametri di simulazione. Il software invece offre come feedback all'utente la stampa del passo corrente di simulazione e il salvataggio dei log nell'apposita directory. Data la semplicità della *shell* si ritiene superfluo allegare immagini del programma.

2.6 MPI e MPICH

Ognuno dei nodi è eseguito al livello più basso da un processo che gira su una macchina differente. Serve quindi un paradigma per la programmazione concorrente tra processi in C++. Il paradigma più utilizzato per la comunicazione tra nodi appartenenti ad un cluster che esegue un programma parallelo è **MPI**. I motivi per cui MPI è ampiamente utilizzato sono sostanzialmente *performance*, *portabilità* e *scalabilità*. Negli anni si è imposto come lo standard *de facto* per la comunicazione tra processi che eseguono programmi paralleli a memoria distribuita.

Da notare che MPI è solo una interfaccia. Le sue implementazioni più importanti sono *MPICH* e *openMPI*. Sebbene MPICH sia stata la prima implementazione, ad oggi non esistono particolari motivazioni per preferire l'una o l'altra libreria. In qualche caso, MPICH potrebbe offrire migliori performance ed una migliore compatibilità. L'implementazione scelta per il simulatore proposto è dunque MPICH.

2.6.1 Comunicazione bloccante e non bloccante

Il protocollo scelto permette sia la comunicazione bloccante (sincrona) sia quella non bloccante (asincrona). Nel caso di comunicazione sincrona, il processo che chiama la routine di invio o di ricezione si mette in attesa finché la trasmissione non è terminata. Una volta ritornata la routine dunque, i dati sono immediatamente accessibili. Al contrario, nel modello asincrono la routine ritorna immediatamente, anche se il trasferimento non è ancora terminato. Dunque i dati non sono necessariamente accessibili dopo il ritorno della routine. Se con il primo tipo di comunicazione dunque, non si corre il rischio di accedere a dati non ancora pervenuti, si configura però il rischio di un problema comune nella programmazione concorrente: il *deadlock*. Se ad esempio due processi devono mutuamente inviarsi dei messaggi ed entrambi mettono come prima istruzione un invio bloccante, ciascuno dei due processi aspetterà il completamento dell'invio indefinitamente, poiché il processo che dovrebbe ricevere sta anch'esso aspettando. Nella nostra applicazione lo scambio mutuo di messaggi è assai frequente, per cui la scelta non può che ricadere sul modello asincrono. Tale scelta è anche conveniente per motivi di efficienza: quando possibile infatti è conveniente evitare stalli.

2.6.2 Tipi composti

Una delle caratteristiche importanti di MPI è la possibilità di creare tipi composti, diversi dai tipi semplici predefiniti. Senza scendere troppo nei dettagli, per

l'implementazione del simulatore è stato utile definire strutture di tipi più semplici. Ad esempio i feedback scambiati tra i nodi sono modellati come strutture contenenti il consumatore recensore, il provider recensito e il servizio relativo al feedback. Senza la possibilità di espandere i tipi semplici molte caratteristiche del simulatore non sarebbero state implementabili.

2.6.3 Impossibilità di creare nuovi processi durante l'esecuzione

Uno dei limiti più stringenti di MPI riguarda l'impossibilità di far partire nuovi processi durante la simulazione. Come esposto nel capitolo riguardante l'architettura funzionale dunque, per simulazioni con numero di nodi dinamici è opportuno prevedere un numero adeguato di nodi aggiuntivi che rimarranno "dormienti" fino alla loro eventuale attivazione. In realtà, per alcuni tipi di attacco non è necessario però creare nuovi nodi.

2.6.4 Le primitive più importanti utilizzate

MPI_Isend(...)

È la funzione prevista dalla libreria per l'invio asincrono dei messaggi. Il primo parametro contiene un puntatore alla zona di memoria allocata per contenere i dati che si desiderano inviare. Il parametro intero *count*, indica il numero di elementi che si stanno trasmettendo, che sarà dunque uno nel caso di semplice elemento, o maggiore di uno nel caso di array di elementi. Il terzo parametro indica il tipo di dato trasmesso, secondo tipologia definita dal protocollo MPI. Il parametro intero *tag* permette di far "corrispondere" i messaggi inviati con quelli che un ricevente si aspetta di ricevere per evitare "mismatch". Sorvolando sul parametro di tipo MPI_*Comm* la cui spiegazione comporterebbe un livello di dettaglio eccessivo, l'ultimo parametro serve per sincronizzare l'invio e la ricezione dei messaggi, attraverso la funzione *wait()* illustrata successivamente.

MPI_Irecv(…)

È la funzione prevista per ricevere i messaggi. I parametri sono gli stessi della corrispondente funzione d'invio, dove questa volta al posto del destinatario sarà chiaramente indicato il ricevente. Il puntatore alla zona di memoria passato come parametro sarà utilizzato per indirizzare i dati ricevuti.

MPI_Wait(…) e MPI_Waitall(…)

Come esposto in precedenza, quando le funzioni per l'invio e la ricezione asincrono ritornano non è garantito che i dati siano stati rispettivamente inviati o ricevuti. Per evitare di invalidare dati non ancora trasmessi o di utilizzare dati

"spazzatura", la funzione *wait()* permette di attendere che l'invio o la ricezione di interesse siano completate. Similmente, la *waitall()* prende come parametri array di richieste e stati e permette di attendere su più completamenti. Questo come spiegato in precedenza previene il *deadlock* e migliora le performance evitando tempi di stallo.

MPI_Probe(…)

La funzione *Probe()* permette di verificare se è stato ricevuto un messaggio, senza però "riceverlo". Senza scendere troppo nei dettagli, una funzione di questo tipo è utile ad esempio per la ricezione dinamica dei messaggi. La funzione di ricezione, come abbiamo visto, prende come parametro il numero di elementi da ricevere. Nel caso dei feedback ad esempio, il nodo che riceve non sa quanti *feedbacks* dovrà ricevere. Facendo un test con la funzione in oggetto, è possibile verificare il numero di elementi ricevuti e allocare opportunamente la memoria necessaria.

MPI_Barrier(…)

L'ultimo metodo proposto, il più semplice, serve a tenere sincronizzati i processi. Ogni qualvolta un processo qualsiasi invoca la funzione *barrier()* rimane in attesa finché tuti gli altri processi non chiamino anch'essi la stessa funzione. In questo modo, tutti i processi rimangono sincronizzati sullo stesso passo della simulazione.

Creazione dei tipi

La creazione dei tipi personalizzati è assai macchinosa e di basso livello. In sintesi, prevede la definizione dei sottotipi più semplici, e nel caso delle strutture utilizzate, richiede di specificare lo spiazzamento in termini di memoria.

2.6.5 Esecuzione di un programma MPI

Come esposto nell'apposito scenario, il comando per l'avvio di un programma MPI è qualcosa di questo tipo:

mpiexec –f hostfile –n numero_di_nodi ./simulatore

Il flag –f con relativo file, opzionale, consente di definire l'elenco dei nodi del cluster ed eventualmente come distribuire i processi sui singoli nodi. Il flag –n invece permette di specificare il numero di processi da avviare per la simulazione. L'ultimo parametro è, chiaramente, il nome del file eseguibile.

3 Architettura Funzionale

Nel capitolo precedente abbiamo discusso i dettagli progettuali del simulatore. In questo capitolo invece, sarà illustrata la struttura del simulatore dal punto di vista funzionale. Questo dovrebbe permettere di comprenderne la modalità di utilizzo, il tipo di simulazioni per cui può essere impiegato, di comprendere alcune delle scelte progettuali illustrate e come queste siano legate all'architettura funzionale.

3.1 Architettura a due livelli

Per le considerazioni fatte nei capitoli precedenti, quali ad esempio la necessità di svincolare il progettista del sistema di reputazione dalla programmazione di basso livello e di facilitare l'implementazione del RMS e dello scenario da testare, si è scelto di sviluppare il simulatore su più livelli, cosi come proposto in [2] e [3]. In particolare, posto che il designer del sistema di gestione della reputazione deve poter fornire i parametri della simulazione (quali ad esempio la durata della simulazione), si può immaginare il simulatore organizzato su due livelli. Al livello più basso sono implementate le primitive di comunicazione. Dal livello più alto, il designer può, attraverso gli opportuni metodi, utilizzare le primitive fornite per implementare il sistema di reputazione. Una struttura a due livelli di questo tipo, conferisce all'ambiente di simulazione una buona estensibilità. Ad esempio, se il designer del sistema di reputazione volesse testare la vulnerabilità del suo sistema ad un tipo di attacco non fornito nella libreria, potrebbe modellare il comportamento di interesse estendendo uno di quelli forniti (se simile). Nel caso in cui invece volesse testare uno scenario più particolare, potrebbe implementare le opportune interfacce ed estendere il simulatore, senza comunque dover scendere nei dettagli di basso di livello della programmazione concorrente.

La Figura 3-1 indica dunque che il progettista del RMS può settare i parametri, scegliere comportamenti già disponibili nella libreria o implementarne di nuovi ma non dovrebbe accedere alle primitive del livello comunicazione.

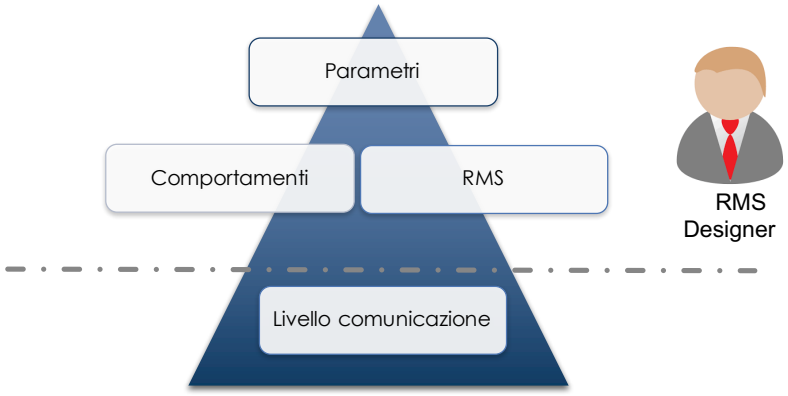

Figura 3-1 Architettura a livelli: dal livello più alto vengono richiamati metodi implementati al livello più basso.

Durante l'esecuzione, come mostrato in Figura 3-2, ciascun agente interagisce con le altre entità scambiando servizi sul piano logico (livello più alto). Al livello più basso (piano fisico), ciascun agente viene modellato attraverso un processo attivo. In questo livello sono effettivamente implementate le primitive di comunicazione e i comportamenti degli agenti. La libreria utilizzata per la programmazione distribuita, non consente la creazione di nuovi processi durante l'esecuzione del programma. Tutti i processi devono quindi essere avviati sin dall'inizio della simulazione. I processi inattivi, fino al momento in cui il *leader* non avvierà nuovi agenti, rimarranno in quiescenza (inutilizzati).

3.2 Modello basato su Agenti

Il tipo di sistemi di reputazione che vogliamo testare è un sistema distribuito in cui degli agenti interagiscono. Un sistema di questo tipo non può che essere modellato come una rete di nodi. In particolare, come sarà mostrato nei due paragrafi successivi, nella piattaforma progettata esistono tre tipi di nodi, di cui due di questi

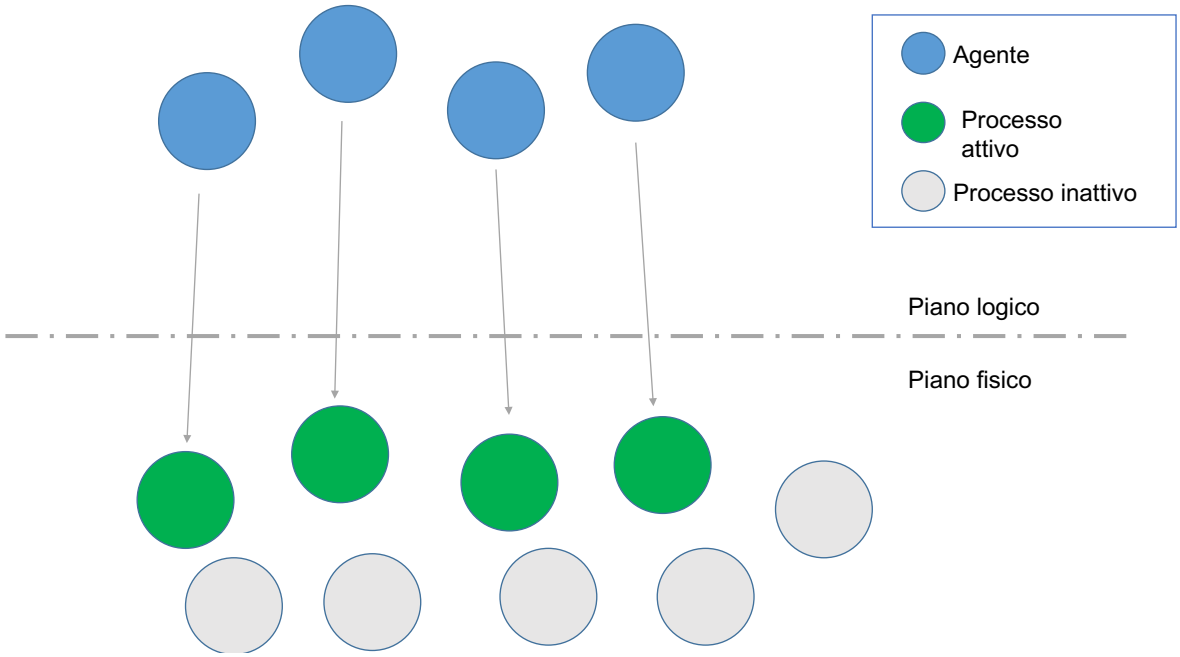

Figura 3-2 Strato fisico e strato logico

servono servono unicamente di supporto per permettere alcune delle funzionalità previste dal simulatore.

3.2.1 I nodi agente

I nodi di questo tipo (Figura 3-3) rappresentano i generici utenti di un'applicazione distribuita. Tali entità, comportandosi da consumatori richiedono servizi con un determinato tasso di *operatività,* da provider invece forniscono servizi ai consumatori da cui hanno ricevuto richieste. Essendo membri del sistema di reputazione, questi, attraverso l'algoritmo, calcoleranno i valori di reputazione dei nodi di competenza. Ciascun agente inoltre avrà un comportamento, modellato come composizione di:

- un comportamento in quanto fornitore di servizi;
- un numero arbitrario di comportamenti da membro del RMS.

In accordo con quanto descritto nel capitolo dedicato agli attacchi ai sistemi di reputazione, il comportamento in quanto *service provider* influenza il modo in cui il provider risponde alle richieste di servizio. Ad esempio il traditore alternerà risposte positive a risposte negative. Il comportamento in quanto membro del RMS invece, influenza il modo il consumatore andrà a valutare le transazioni. Ad esempio un diffamatore, qualunque sia l'esito della transazione, trasmetterà valori di reputazione alterati anche per i nodi onesti. Ovviamente, in maniera coerente con quanto indicato nella documentazione progettuale, per tutti i tipi di comportamento è previsto un elenco di obiettivi, in modo che il comportamento malevolo possa valere solo per tali nodi e che l'agente in questione possa comportarsi onestamente con i restanti nodi.

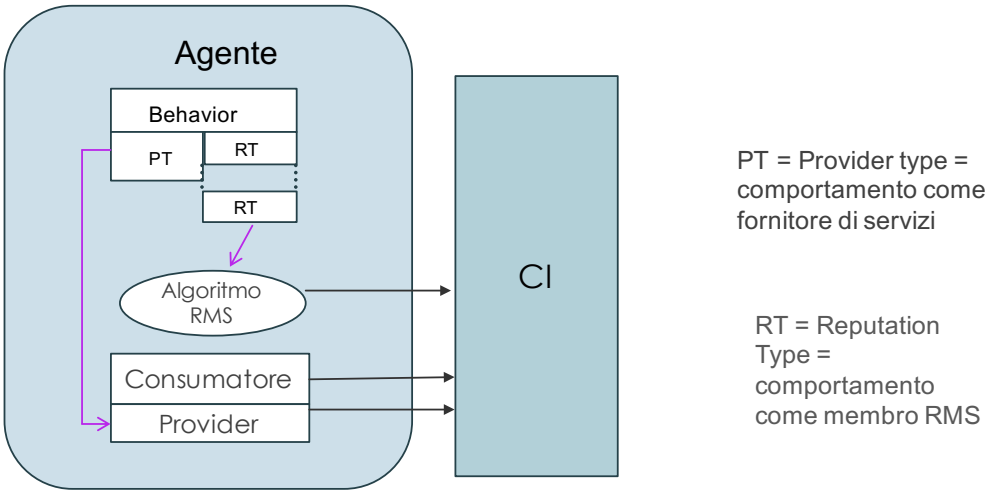

Figura 3-3 Un generico nodo della rete: i comportamenti RT influenzano il comportamento del consumatore, il comportamento PT influenza l'attività del provider. La comunicazione avviene attraverso la Communication Interface

La comunicazione tra i nodi, sia per lo scambio dei servizi sia per la trasmissione dei valori di reputazione, deve necessariamente avvenire attraverso l'interfaccia di comunicazione.

3.2.2 Il *leading agent*

Il *leading agent* è il primo dei due nodi di supporto. Tale nodo infatti, non ha un compito definito dal sistema di reputazione ma è necessario per permettere flessibilità e dinamicità alle simulazioni, e versatilità delle metriche. Uno dei requisiti di questo tipo di sistemi infatti, è la dinamicità dei nodi della rete. Ad esempio, durante la simulazione, un nuovo nodo potrebbe voler entrare nella rete, o uno già in rete potrebbe uscirne. Il *leading agent* si occupa quindi del coordinamento dei nodi generici permettendo l'ingresso di eventuali nuovi nodi. come mostra la Figura 3-4, ad ogni passo il *leader* stabilisce lo stato di ciascun nodo agente. In questo modo, i due nodi che nell'esempio mostrato al passo iniziale erano inattivi, diventano attivi dal passo n.

Un altro dei requisiti posto per il simulatore è la massima versatilità nel calcolo delle metriche. Il secondo compito del leader è quindi la raccolta delle transazioni per come effettivamente accadute (*verità assoluta*). In questo modo, il tester del simulatore può implementare anche metriche che misurino il discostamento dei valori calcolati dal comportamento effettivo dei nodi (quindi non alterato da eventuali comportamenti malevoli di tipo partecipante RMS). Come mostrato nella Figura 3-5, per assolvere questi due compiti, il leader utilizza la *communication interface*.

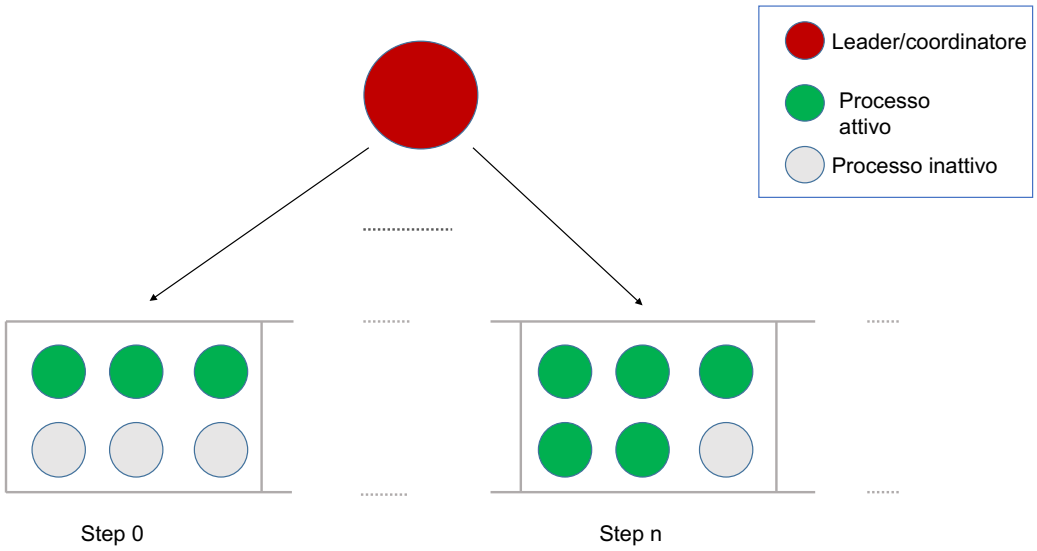

Figura 3-4 Coordinamento dei nodi agente da parte del leader

3.2.3 Il *dispatcher* dei provider

Il *dispatcher dei* servizi è l'ultimo tipo di nodi. Anch'esso non appartiene propriamente al sistema di reputazione ma, esattamente come il leader, è necessario per conferire ulteriori funzionalità al simulatore. Un altro dei requisiti infatti, è fornire la massima flessibilità nella concezione di nuovi scenari, ed in particolare nella topologia, da simulare. Una delle eventualità che si vuole permettere nel simulatore proposto è, ad esempio, che un provider possa fornire più servizi o che possa modificare i servizi offerti in un determinato istante della simulazione. Un altro caso che potrebbe verificarsi, non volendo mettere limiti al tipo di topologie che il progettista del sistema di reputazione può testare, è che un nodo della rete non conosca fornitori per un certo tipo di servizio di cui ha necessità. Il *dispatcher dei providers* è stato pensato per rispondere a queste due esigenze: in ogni istante della simulazione, ciascun nodo, grazie a questa particolare entità, conosce i fornitori per. ciascun servizio. Anch'esso utilizza la *communication interface* per comunicare con i nodi agente e assolvere la funzione indicata.

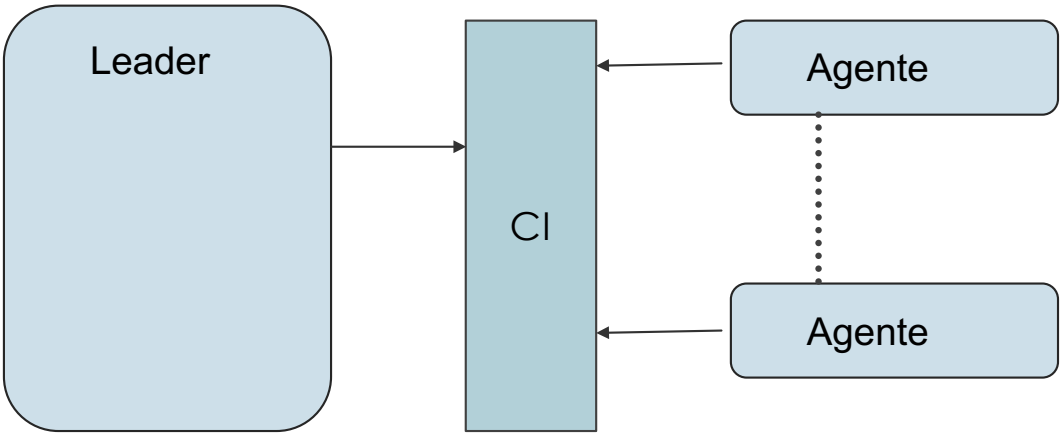

Figura 3-5 leader attraverso la CI coordinata gli agenti e si occupa della raccolta della verità assoluta e del coordinamento.

3.3 Overview

La Figura 3-6 mostra una panoramica ad alto livello del funzionamento del simulatore. Il designer del RMS da come input della simulazione la lista dei nodi con i relativi comportamenti e parametri relativi (ad esempio la cooperatività e i target), la topologia e i parametri della simulazione (ad esempio il numero di passi della simulazione). Avviata la simulazione, attraverso la *Communication Interface*, il *leader*, il *dispatcher* e gli agenti comunicano per ottemperare alle funzioni elencate in precedenza. Il risultato della simulazione è costituito dai valori di reputazione calcolato dagli agenti e dai valori di verità (ossia il rapporto transazioni andate a buon fine diviso numero di transazioni totali).

Poiché nel caso di simulazioni complesse potrebbe essere impraticabile definire manualmente l'elenco dei nodi e la topologia della rete, soprattutto nel caso di simulazioni ad elevata *casualità*, al simulatore è stato aggiunto un modulo per definizione interattiva della topologia e dei nodi della simulazione. Di tale generatore si parlerà nel paragrafo successivo.

3.4 Il generatore della topologia

Il generatore implementa le funzionalità per la creazione dei file contenenti l'elenco dei nodi con i i rispettivi comportamenti (*behaviors*) e l'elenco delle transazioni avvenute tra nodi nella "fase zero" (ossia prima dell'effettivo inizio della simulazione). Simmetricamente, il simulatore è provvisto di un *parser* per il caricamento della topologia descritta dai file generati.

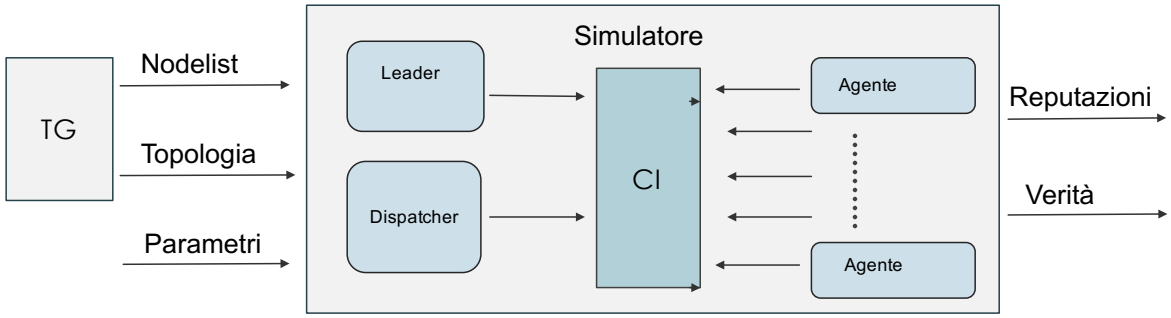

Figura 3-6 Overview sistema

3.4.1 Interazione con l'utente

Data la necessità di fornire al progettista del RMS la possibilità di creare comportamenti personalizzati, il generatore è stato fornito di un'interfaccia interattiva, in modo che l'utente possa scegliere i parametri di interesse per ogni comportamento dei nodi.

All'avvio del generatore viene richiesto all'utente di inserire il numero di nodi complessivo che faranno parte della simulazione e la percentuale (rispetto al numero complessivo) di partecipanti che ciascuno avrà come vicini (i dettagli su come questi parametri vengono usati dal generatore saranno illustrati in seguito). Successivamente il programma chiederà di inserire i comportamenti. I file generati conterranno le specifiche per il numero di nodi specificato maggiorato del venti percento. Questi nodi aggiuntivi sono i nodi quiescenti. L'idea infatti è fornire all'utente un metodo rapido per disporre di configurazioni dinamiche e ad alta casualità.

3.4.2 Comportamenti base

Nel caso in cui l'utente digiti il nome di uno dei comportamenti base (*traitor*, *promoter*, *whitewasher*, *slander*), il sistema chiederà di inserire la percentuale di nodi con il comportamento che si sta definendo, la lunghezza media dell'elenco di target e il passo di *rottura* (*beginning step* o *breaking step*).

3.4.3 Comportamenti personalizzati

Nel caso in cui invece l'utente abbia digitato un nome diverso, per prima cosa il sistema chiederà il tipo di comportamento. Successivamente, il sistema permetterà di inserire un numero variabile di parametri. I primi due saranno rispettivamente la percentuale per il tipo che il designer del RMS desidera nel sistema e la lunghezza media dell'elenco degli obiettivi per il comportamento in esame. I restanti non saranno usati dal generatore e saranno direttamente salvati sul file. In questa fase, il

sistema permette di definire parametri pseudocasuali che saranno generati dal sistema a tempo di scrittura del file.

3.4.4 Specifiche

I parametri appena discussi sono poi utilizzati dal simulatore. Nei paragrafi è spiegata brevemente la loro utilità.

3.4.5 Numero di nodi

Molto semplicemente, il numero di nodi viene utilizzato in fase di creazione del file per conoscere il numero di processi da generare. L'id di ciascun nodo sarà il "numero di creazione", ma maggiorato di due, dato che il nodo zero è il *leading agent* mentre il nodo uno è il *providers dispatcher* ed entrambi questi nodi non fanno parte della topologia del sistema. L'id di ciascun nodo può variare durante la simulazione (Es. *WhiteWasher*).

3.4.6 Percentuale di vicini

La percentuale di vicini viene utilizzata per calcolare il numero di nodi con cui ogni nodo, mediamente, ha effettuato transazioni "al passo zero". Più precisamente, il numero di nodi con cui ogni nodo ha effettuato transazioni, viene generato secondo una Gaussiana con media pari al parametro in questione e deviazione standard posta fissa al valore 2. Chiaramente il risultato viene poi arrotondato all'intero più vicino. Gli id dei vicini invece, vengono generati come interi pseudocasuali.

3.4.7 Percentuale di nodi per dato comportamento

In fase di creazione del singolo nodo, il parametro in questione indica la probabilità che il nodo abbia il comportamento a cui si riferisce il parametro. Ad esempio un valore 0.6 per il comportamento *traitor* indicherebbe in fase di creazione di un nodo nel file, una probabilità del 60 percento che il nodo abbia un comportamento *traitor*. Chiaramente, questo non esclude la possibilità che il nodo possa avere anche altri comportamento, in quanto il simulatore, come spiegato, prevede la possibilità di combinare diversi comportamenti.

3.4.8 Lunghezza media elenco degli obiettivi

In maniera perfettamente analoga a quanto avviene per la definizione del numero di vicini, questo parametro, generato secondo una normale con media pari al parametro e deviazione standard fissata a 2, indica il numero di target di ciascun nodo per determinato comportamento. Anche qui, gli id dei target vengono generato come interi pseudocasuali.

3.4.9 Beginning Step

Il *beginning step* è trattato come un intero che indica il passo a partire dal quale il nodo inizierà a comportarsi diversamente (nella maniera che è definita all'interno delle classi *behavior* del simulatore). Questo parametro non è utilizzato dal generatore, ma semplicemente memorizzato su file per essere usato successivamente nelle simulazioni.

3.4.10 Tipo del comportamento

Dopo aver inserito il nome del comportamento, se diverso dai comportamenti basici, il sistema chiederà di inserire una stringa tra "-rt" e "-pt" che indicano rispettivamente un comportamento come membro del RMS (**R**ms **T**ype) o come fornitore di servizi (**P**rovider **T**ype). In realtà questo parametro è presente anche nel caso di comportamenti basici, ma è gestito automaticamente dal generatore. In particolare, come si può notare dal file creato, nel caso di *traitor* o *whitewasher* avremo un flag "-pt", nei restanti due invece avremo il flag "-rt", esattamente in accordo con quanto indicato a proposito degli attacchi ai sistemi di reputazione. Questo parametro è utile al simulatore per la creazione dell'oggetto corretto del tipo opportuno.

3.4.11 File creati

Dopo la fase interattiva con l'utente, il sistema (a meno di errori) creerà due file. Il primo dei due, "*nodelist.txt*", contiene l'elenco dei nodi con i rispettivi comportamenti. Il secondo, "*topology.txt*", contiene invece, per ciascun nodo, l'elenco delle transazioni.

3.4.12 Nodelist.txt

L'elenco dei nodi sarà composto di righe secondo uno schema fisso. Il primo parametro sarà l'id. Il secondo parametro consiste di una coppia di interi non negativi. In particolare, il primo dei due interi indica il passo di ingresso del nodo, cioè l'istante a partire dal quale il nodo sarà *attivo*; il secondo, al contrario, indica l'istante a partire dal quale il nodo sarà *inattivo*. Successivamente, è possibile leggere i parametri relativi al comportamento. In particolare, tra le parentesi tonde, è possibile leggere rispettivamente il nome del comportamento, il grado del comportamento (generato come numero pseudocasuale tra 0.0 e 1.0) e lo *step* di rottura. Tra le parentesi angolari invece sono contenuti gli id dei target.

3.4.13 Topology.txt

Anche nel file *topology.txt* il primo parametro è l'id del nodo. In questo caso, l'id può essere letto come id del nodo "consumatore". In seguito, è possibile leggere un numero indicante un servizio (un double) e, tra parentesi angolari, rispettivamente l'id (che possiamo vedere come id del nodo fornitore) e i numeri di transazioni positive e negative (questi ultimi due sono salvati tra parentesi tonde).

Come si può notare, i file contengono una serie di caratteri non indispensabili ai fini della memorizzazione (ad esempio le parentesi). Si è scelto di inserire caratteri di questo tipo nei file per permettere, qualora il tester ne avesse la necessità, di individuare parametri e modificarli come necessario. Cosa che sarebbe impensabile senza caratteri di delimitazione (quanto meno visiva).

4 Esperimenti

In questo capitolo vengono mostrati i risultati di alcune delle simulazioni effettuate con l'ambiente di simulazione presentato. Poiché la trattazione risulterebbe eccessivamente ampia, vengono presentati solo gli esperimenti più utili alla comprensione o più significativi. Come accennato in precedenza, gli esperimenti condotti hanno come oggetto i due sistemi di reputazione presentati nel secondo capitolo. In più, del sistema di reputazione assunto come caso di studio sono state testate tre versioni. Prima di mostrare i risultati delle simulazioni, sono discussi l'hardware utilizzato e alcuni esempi di metriche.

4.1 Hardware delle simulazioni

L'ambiente dei test è costituito da un cluster di 16 macchine virtuali identiche. Ciascuna macchina ha le seguenti specifiche tecniche:

- CPU: Intel Xeon E3-12xx v2 @ 2,70 GHz;
- Numero socket: 2;
- Numero thread per core: 1;
- Numero core per socket: 1;
- L1d cache: 32K;
- L1i cache: 32K;
- L2 cache: 4096K;
- L3 cache: 16384K;
- RAM: 4GB.

La prima macchina del cluster è costituita da due interfacce di rete. La prima gli consente di comunicare con la rete pubblica e quindi di essere raggiunta dall'esterno ad esempio tramite SSH per i comandi da remoto, tramite i quali sono state eseguite le simulazioni. La seconda interfaccia invece, presente anche sulle rimanenti macchine, la espone verso la rete interna, tramite cui comunicano le macchine del cluster.

Per comodità di lavoro, anziché creare una copia dei file necessari per l'esecuzione del simulatore, si è scelto di creare una cartella condivisa tra le macchine. Per far ciò si è utilizzato NFS o *Network File System.* NFS permette alle

macchine di una rete di condividere file, directory o interi File Systems attraverso un'architettura di tipo client-server. Il protocollo prevede che un dispositivo della rete, nel nostro caso la prima macchina, si comporti da server, condividendo la risorsa di interesse. A questo punto le macchine client possono accedere alla risorsa dichiarando un punto di montaggio. Dopo aver effettuato l'accesso i client possono accedere alle cartelle condivise in maniera assolutamente trasparente. Questo ci permette anche di reperire facilmente i log a partire da qualunque macchina.

4.2 Esempi di metriche

Uno dei requisiti richiesti riguardava la flessibilità nella scelta delle metriche da calcolare da parte del RMS designer. È stata discussa la funzione "oracolo" del leader, ossia la raccolta da parte di quest'ultimo della verità sulle transazioni. Avendo a disposizione i valori esatti delle transazioni e i valori calcolati dai nodi, il progettista dispone di ampia libertà nella definizione di metriche personalizzate. In più, avendo il leader raccolto tutte le transazioni, è anche possibile calcolare i valori restituiti da *Eigentrust* nella versione centralizzata e confrontare i risultati con l'algoritmo distribuito sotto osservazione. Quindi il progettista potrebbe ad esempio crearsi delle metriche che misurino lo scarto dai valori calcolati da *Eigentrust*, che come è stato già detto, è uno degli algoritmi più impiegati. Nei paragrafi seguenti sono illustrati un paio di esempi di metriche personalizzate impiegate sulle nostre simulazioni.

4.2.1 Reputazione media

Se indichiamo con N il numero di nodi "recensori", possiamo calcolare la *reputazione media* di un generico nodo *i* calcolando la media delle recensioni:

$$\bar{r}_i = \frac{\sum_{j \in N} r_{ji}}{N}$$

Una metrica di questo tipo permette di annullare o affievolire l'effetto di eventuali recensori mendaci, purché siano in numero trascurabile rispetto al numero di recensori totali.

4.2.2 Errore su singolo nodo ed errore totale medio

Se indichiamo con R_{0i} il valore riportato dal nodo leader per il generico nodo *i*, cioè il rapporto tra il numero di transazioni soddisfate dal nodo *i* come fornitore e il numero totale di transazioni, sia fruttuose che infruttuose riguardanti il nodo *i* in veste di provider, possiamo definire *l'errore su singolo nodo* E_i:

$$E_i = R_{0i} - \bar{r}_i = \frac{\sum_{j \in N} sat(j,i)}{\sum_{j \in N}\left(sat(j,i) + unsat(j,i)\right)} - \frac{\sum_{j \in N} r_{ij}}{N}$$

In sintesi, questa metrica misura lo scarto della reputazione media dal valore esatto fornito dal leader.

A partire dall'errore su singolo nodo è possibile calcolare *l'errore totale medio*:

$$E_{tot} = \frac{\sum_{j \in N} |E_i|}{N}$$

Quest'ultima quantità corrisponde alla media aritmetica dei valori assoluti degli errori su singolo nodo.

4.3 Il caso di studio

I primi esperimenti presentati hanno come oggetto il sistema di reputazione del caso di studio. Per dare un senso ai parametri indicati nelle simulazioni, riportiamo qui le formule impiegate dall'algoritmo:

$$lt_{ij} = \frac{sat(i,j)}{sat(i,j) + unsat(i,j)}$$

$$lr_{ij} = \alpha * lt_{ij} + (1 - \alpha) * r_{ij}(t-1)$$

$$r_{ij}(t) = (1 - \beta) * lr_{ij}(t) + \beta \frac{\sum_{k \in K} r_{ij}(t-1) * r_{kj}(t-1)}{\sum_{k \in K} r_{ik(t-1)}}$$

dove abbiamo chiamato lt_{ij} local trust, lr_{ij} local reputation, r_{ij} global reputation e α β sono parametri compresi tra 0 e 1, che servono per pesare rispettivamente le nuove informazioni nell'aggiornamento della reputazione locale e le informazioni ricevute dai vicini nel calcolo della reputazione globale. Nel prossimo paragrafo sono illustrati i risultati ottenuti dai quattro attacchi discussi all'inizio della trattazione. Nelle nostre simulazioni, per mettere in evidenza gli effetti degli attacchi, si è scelto di far interagire ciascun nodo con tutti i rimanenti ad ogni passo.

4.3.1 Setting sperimentale

Nella tabella che segue sono riassunti i valori dei parametri per le simulazioni che saranno mostrate in seguito.

Parametro	Valore
α	0.1
β	0.1
τ	0.4
finestra	30

Attacco	Coop attaccante	Coop Vittima	Step inizio
Slandering	1	1	50
Promoting	0.20	1	50
Traditore	Varia con lo step	Irrilevante	0
Whitewashing	0	Irrilevante	0

4.3.2 Slandering

Il primo caso illustrato riguarda l'attacco del diffamatore. In questa simulazione non sono presenti elementi di casualità, in quanto sia l'attaccante sia la vittima si comportano in maniera deterministica. Ed in particolare, tutti gli agenti rispondono positivamente nel soddisfacimento delle richieste. Come ci si potrebbe aspettare, per un nodo onesto e non vittima dei diffamatori, la reputazione convergerà al valore massimo, che nel caso dell'algoritmo in esame è 1. E questo avviene qualunque sia il valore dei parametri.

Diverso è chiaramente l'andamento dei valori di reputazione nel caso dei nodi vittima della diffamazione. La Figura 4-1 mostra l'andamento del valore di reputazione di un nodo vittima onesto al variare della percentuale di diffamatori. Al fine di renderne più evidente l'effetto, si è scelto di far iniziare l'attacco allo *step* 50. Si nota dunque che, fino al *beginning step* dei diffamatori, la reputazione delle future vittime è corretta e converge al valore unitario. Dall'istante in cui l'attacco di diffamazione parte, il valore di reputazione decresce con una rapidità e ad un valore finale che dipendono dal numero di attaccanti: in particolare la rapidità cresce all'aumentare del numero di attaccanti, il valore finale decresce al crescere del numero di collusi.

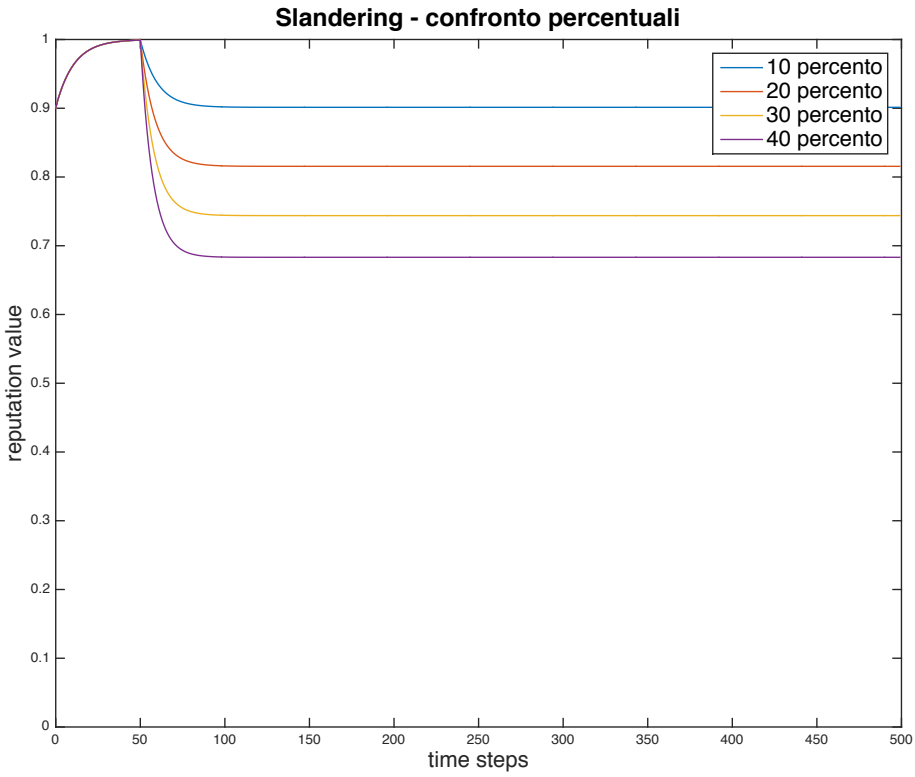

Figura 4-1 Confronto valori reputazione al variare della percentuale di diffamatori

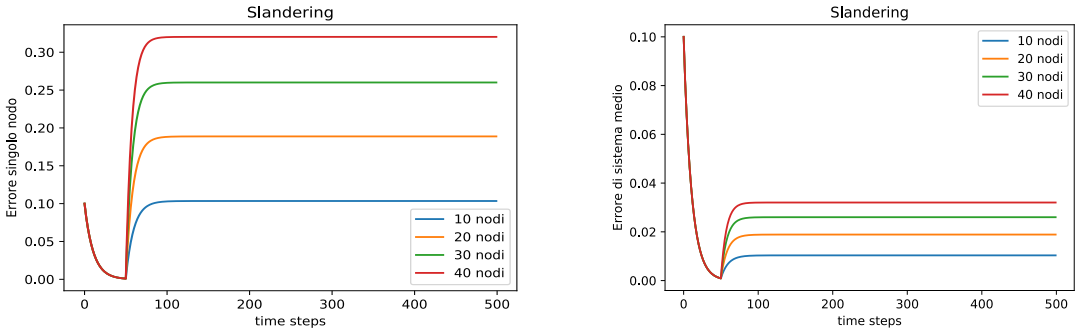

Figura 4-2 Errore su singolo nodo ed errore totale medio per un attacco di slandering

La **Errore. L'origine riferimento non è stata trovata.**Figura 4-2 mostra rispettivamente l'errore su singolo nodo e l'errore totale medio calcolati su un nodo onesto. Come si potrebbe immaginare, entrambi crescono al crescere dei diffamatori, mentre nell'intervallo precedente all'azione dei nodi collusi i due errori tendono a 0. L'errore sul singolo nodo è di entità maggiore rispetto all'errore totale medio, il quale è mitigato dalla media sugli N nodi recensori

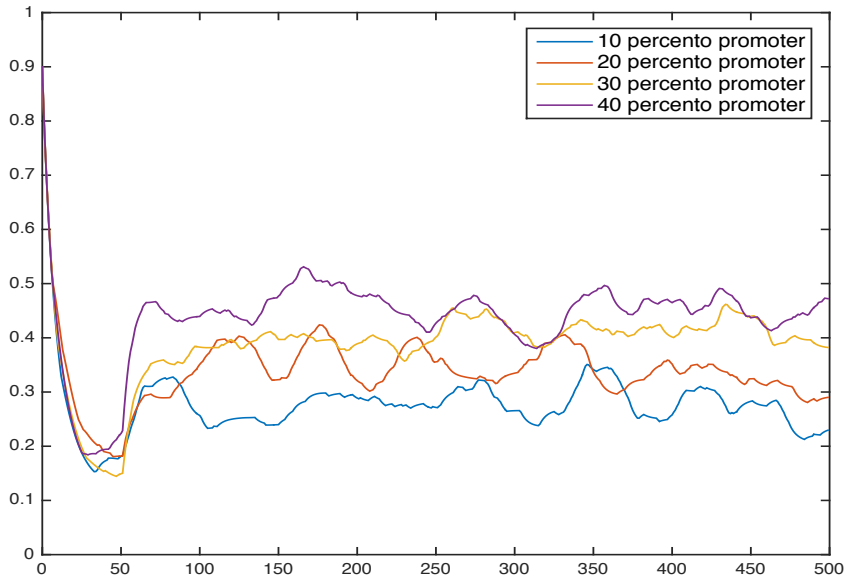

Figura 4-3 Valore di reputazione di un nodo beneficiario al variare della percentuale di promotori

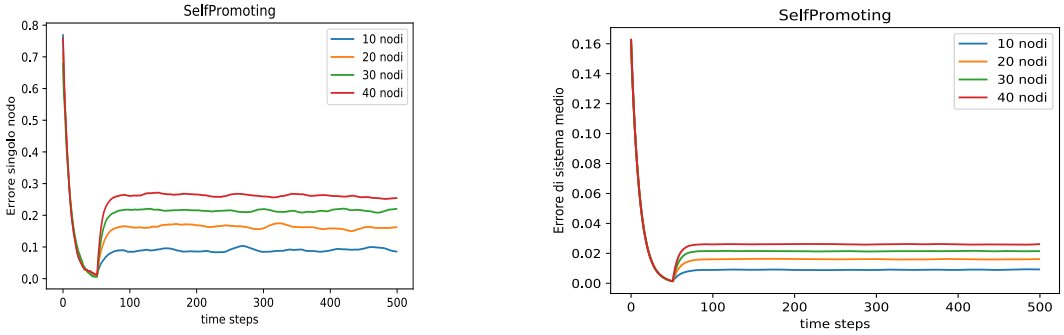

Figura 4-4 Errore su singolo nodo ed errore totale medio su nodo beneficiario

4.3.3 Promoting

L'attacco di *promoting* ha ragione di esistere nel caso in cui l'obiettivo dei promoter (che nel caso dell'attacco in esame è un *beneficiario*) non sia totalmente onesto. Nell'esperimento in analisi, il nodo beneficiario ha una *cooperatività* del venti percento, il che equivale a dire che solo due volte su dieci risponderà alle richieste in maniera corretta. L'inserimento di questo elemento di casualità produce l'effetto più oscillante alle curve discusse precedentemente. Dalla prima delle figure indicate (Figura 4-3), si nota che, come intuitivamente si potrebbe immaginare, al crescere del numero di promotori, il valore di reputazione del nodo beneficiario si assesta mediamente ad un valore più alto. Considerazioni simili possono farsi dalle curve rappresentanti i due errori, in Figura 4-4. Come nel caso dell'attacco di *slandering* prima dell'inizio dell'attacco i due errori tendono a zero. Iniziato l'attacco, i due errori crescono e si attestano ad un valore più alto al crescere della percentuale di

collusi. Dall'ultima figura si può inoltre notare come il mediare l'errore singolo sugli N nodi vada ad ammortizzare l'effetto della casualità.

4.3.4 Attacco del traditore

In tutte le simulazioni mostrate, l'agente traditore alterna cento passi rispondendo alle richieste in maniera onesta e cinquanta passi rispondendo in maniera antisociale verso i nodi target. Dalla Figura 4-5**Errore. L'origine riferimento non è stata trovata.** si evidenzia che, al variare del numero di nodi traditori, non cambia la reputazione calcolata di un singolo nodo traditore. Questo è dovuto al metodo di aggregazione del sistema di reputazione in analisi, che in fase di aggregazione calcola una media pesata dei valori ricevuti. D'altra parte, il sistema di reputazione del caso di studio è un sistema di reputazione assoluto e di conseguenza i valori di reputazione di ciascun nodo non variano al variare del numero di nodi antisociali nel soddisfacimento delle richieste. La stessa considerazione può essere essere fatta osservando l'andamento degli errori su singolo nodo al variare del numero di traditori, in Figura 4-6. Anche in questo caso le curve si sovrappongo perfettamente. Ciò non si verifica invece nel secondo tipo di errore, in cui le curve sono distinte e, come facilmente intuibile, al crescere della popolazione dei traditori, l'errore aumenta. Nei grafici relativi ai due errori, possiamo notare che i massimi coincidono con i momenti in cui il traditore passa dalla modalità malevola alla modalità onesta, vale a dire ai passi 150, 300 e 450. Simmetricamente, i minimi coincidono invece con i passaggi dalla modalità onesta alla modalità malevola.

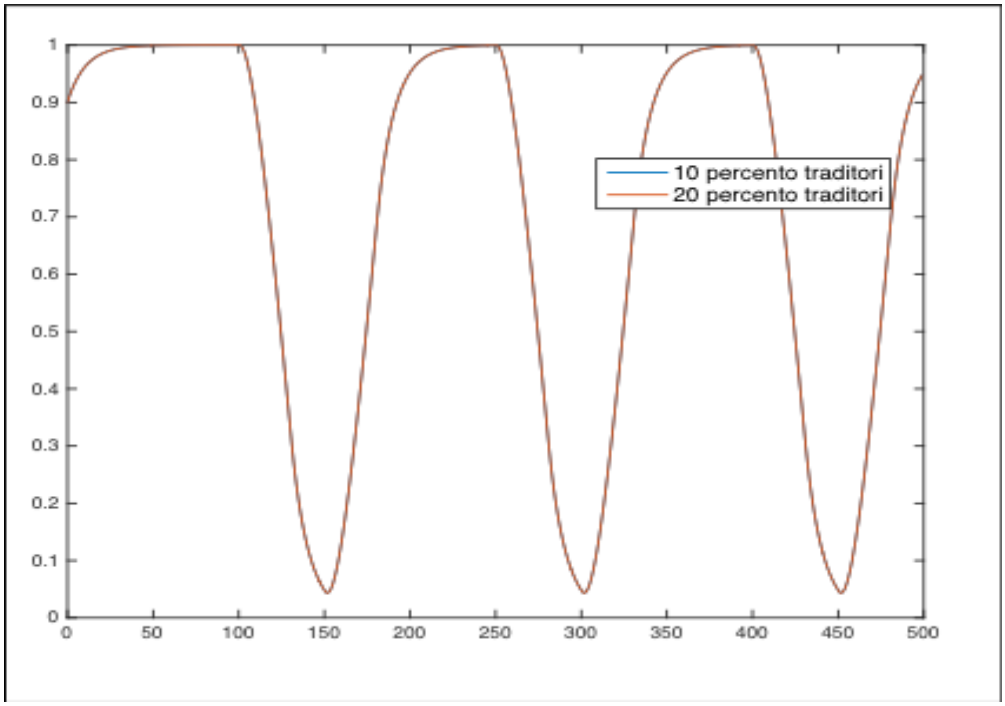

Figura 4-5 Reputazione nodo traditore. Non si registrano differenze al variare del numero di collusi

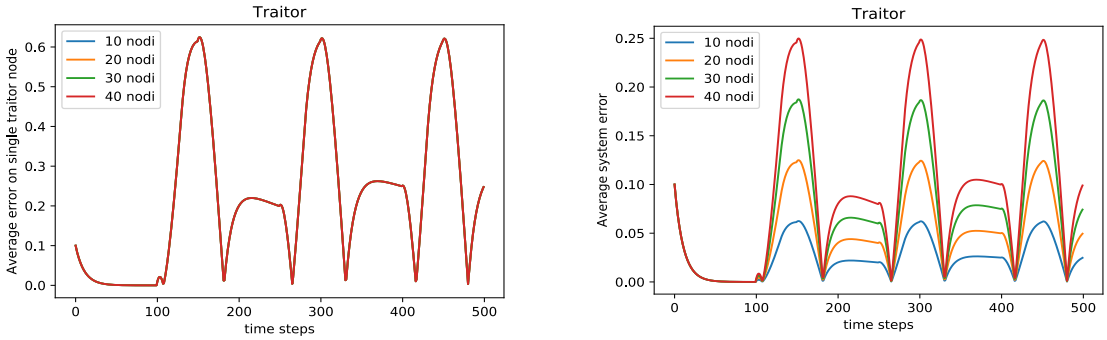

Figura 4-6 Errore su singolo ed errore di sistema per un attacco del Traditore. Le curve del primo grafico si sovrappongono

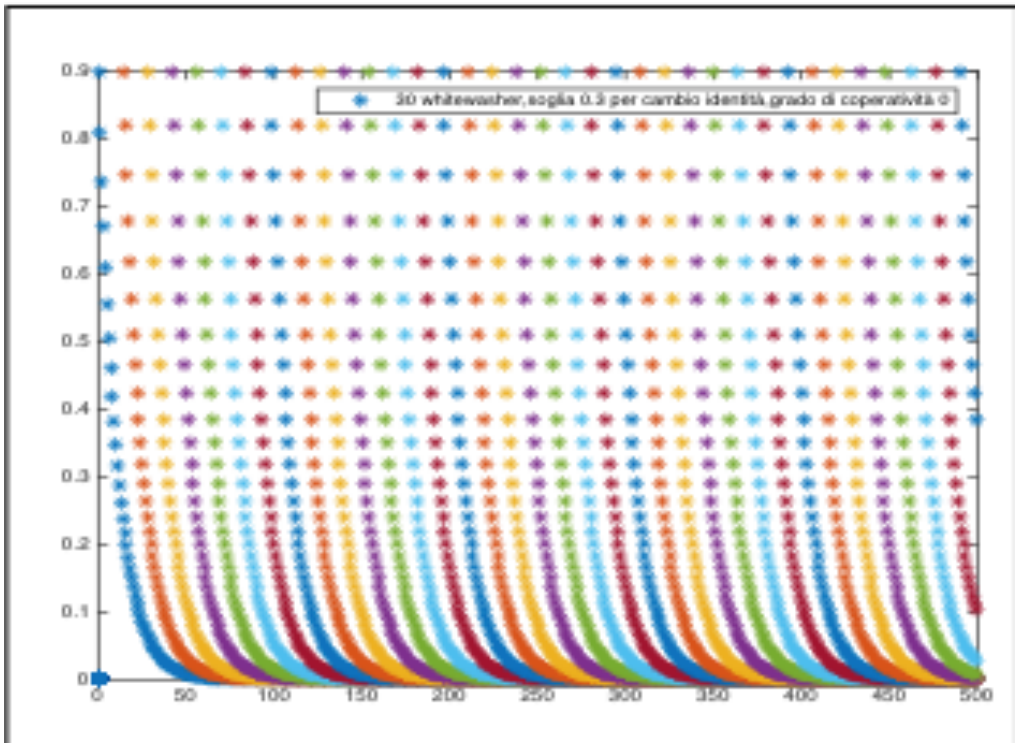

Figura 4-7 Simulazione whitewashing, singoli alias con coop. 0 e soglia fissata a 0.3

4.3.5 Whitewashing

Un attaccante di tipo *Whitewashing*, come spiegato in precedenza, abusa del sistema finché il suo valore di reputazione non scende al di sotto di una determinata soglia di reputazione, momento in cui l'attaccante esce dal sistema e rientra con una nuova identità e una nuova reputazione. Il numero di identità presenti nel sistema può divenire molto elevato se la soglia per il cambio di identità è alta, se la cooperatività è molto bassa o nulla (il provider malevolo risponde quasi sempre o sempre in maniera anti sociale) e se il numero di *whitewasher* è elevato. Infatti, nel caso in cui la soglia affinché l'agente si duplichi sia fissata a metà del valore massimo di reputazione, quando manifesta un grado di cooperatività pari a 0, in 500 passi ciascun attaccante assume circa 55 alias. Se il numero di collusi è 40, nel nostro sistema sono presenti 55 x 40 = 2200 alias per i soli malevoli. Per quanto spiegato in precedenza sull'impossibilità di avviare nuovi processi durante la simulazione, ciò vorrebbe dire effettuare l'intera simulazione con 2200 + 60 processi, dove i 60 processi sono necessari per i nodi onesti. In termini percentuali vorrebbe dire un *overhead* del duemila percento. In altre versioni discusse in seguito, la percentuale di alias è addirittura più alta. Ciò sarebbe computazionalmente inefficiente con sistemi di reputazione meno onerosi come quello in analisi e infattibile con algoritmi particolarmente complessi. Da qui l'esigenza di cambiare approccio per simulazioni di Whitewashing. L'approccio utilizzato prevede l'impiego di una funzione che mappi la nuova identità sul processo liberato dal nodo uscente. Nel caso di

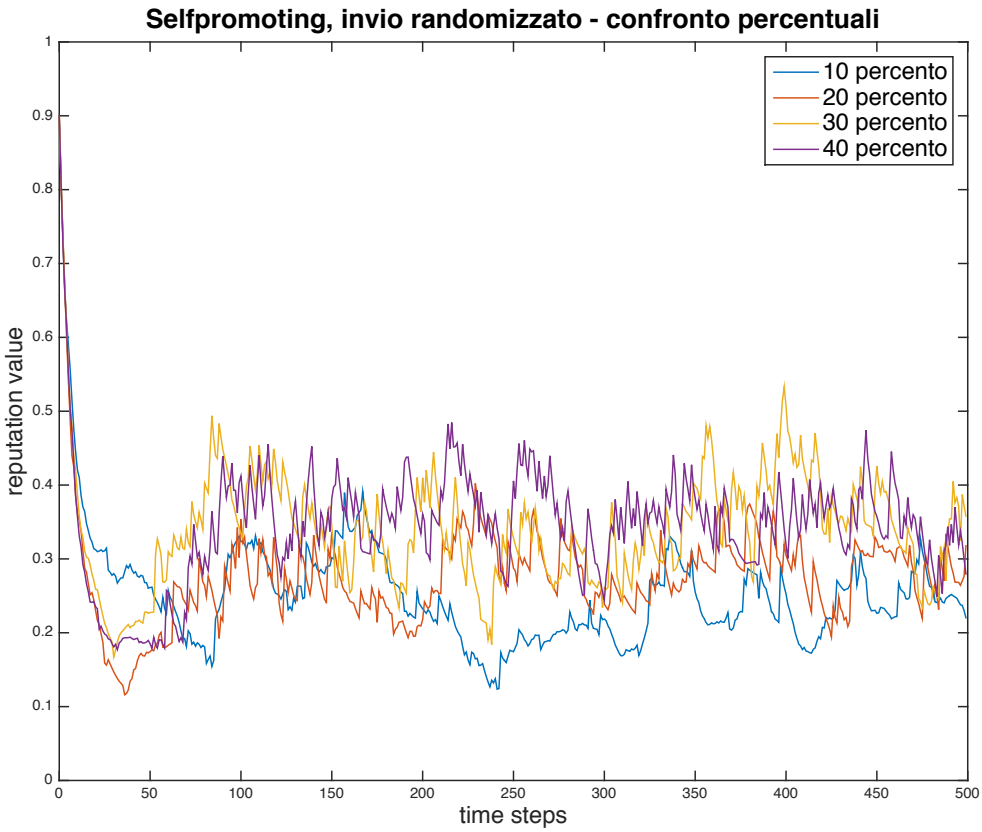

Figura 4-8 Confronto reputazioni nodi beneficiario del promoting nella versione randomizzata in invio

Whitewashing infatti, il numero di agenti partecipanti si conserva (a meno di altri attacchi in corso o di altri comportamenti dinamici).

La Figura 4-7 mostra un esperimento analogo a quello accennato. Per una migliore visibilità infatti è stato riportato qui il grafico con i singoli alias di un nodo *whitewasher* la cui risposta è sempre antisociale. La soglia per il cambio di identità nell'esperimento mostrato è stata fissata a 0.3, il che comporta un ritardo nel cambiamento di identità e quindi un minore numero di alias rispetto all'esempio menzionato prima per il calcolo degli alias.

Come ci si aspetta, dalla figura notiamo che, quando un alias raggiunge il valore 0.3, una nuova curva parte dal valore iniziale con andamento identico, data l'assenza di elementi casuali. Le evidenze sperimentali hanno mostrato che ad esempio con cooperatività del 45 percento e soglia di reputazione a 0.5, in molti casi non si sono verificati cambi di identità perché il valore soglia non viene mai raggiunto.

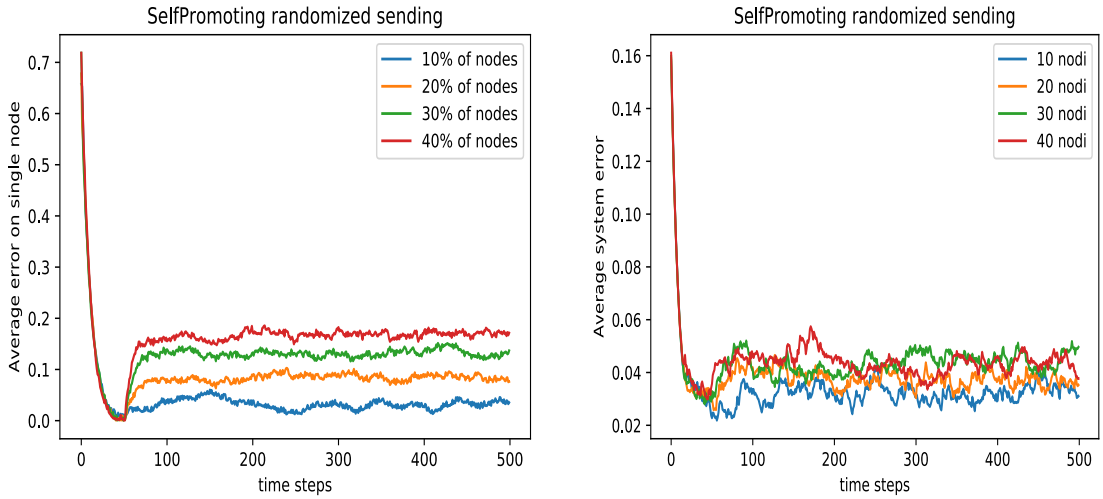

Figura 4-9 Andamento dei due errori in un attacco di Self Promoting nella variante con invio casuale

4.4 Caso Studio con filtro in invio pseudocasuale

La prima variante dell'algoritmo caso di studio che abbiamo testato si differenzia dall'algoritmo originale per l'insieme dei vicini a cui i valori di reputazione locale calcolati sono trasmessi. In particolare, in questa prima variante l'insieme dei vicini a cui trasmettere i valori viene estratto in maniera pseudocasuale. Nell'esperimento mostrato, ciascun agente invia al 10 % dei suoi vicini. Per non estendere eccessivamente la trattazione, per ciascuna delle tre varianti è presentato un solo attacco. I parametri dell'algoritmo, ossia α, β, e τ sono rimasti invariati rispetto alle simulazioni precedenti per consentirne il confronto a parametri non alterati. Da un rapido confronto dei grafici nelle Figura 4-8 e Figura 4-9 con i grafici della versione standard nelle Figura 4-3 e Figura 4-4 , relativi ad un attacco di *Self Promoting* ,si nota che le curve dei valori di reputazione ottenuti con la seconda versione presentano molte più sovrapposizioni. L'errore su singolo nodo obiettivo è più basso mentre quello totale medio è più alto. Quindi la versione randomizzata in invio al 10 %, nello scenario simulato, ha ottenuto valori più corretti per i nodi beneficiari (l'errore su singolo nodo è più basso), ma più lontani dalla realtà sui nodi onesti. Questo comporta valori di errore totale medio più alti.

4.5 Caso studio con filtro in ricezione pseudocasuale

La seconda variante dell'algoritmo del caso di studio è esattamente speculare alla versione appena illustrata. In questa versione dell'algoritmo infatti, l'insieme dei

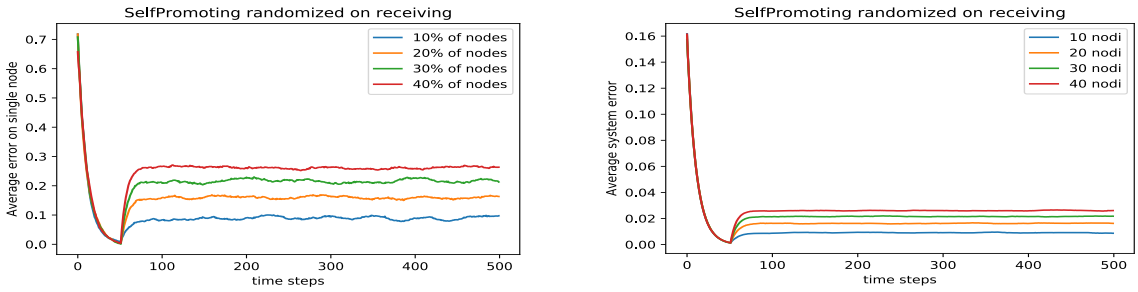

Figura 4-10 Errore su singolo nodo ed errore totale medio per la variante con ricezione casuale

sui valori di reputazione per un attacco di *promoting* sono molto simili a quelli ottenuti dalla versione tradizionale, per cui il grafico non è mostrato. Dai grafici in Figura 4-10 invece è possibile notare che i risultati registrati dall'errore su singolo nodo beneficiario, sono nettamente migliori dalla versione con invio randomizzata, in quanto l'errore si attesta su valori più bassi. Il discorso si inverte invece analizzando le curve degli errori totali.

4.6 Caso studio con soglia in invio

L'ultima versione dell'algoritmo prevede anch'esso una selezione del sottoinsieme dei vicini a cui spedire. Questa volta però il criterio non è casuale ma deterministico. Infatti, tra tutti i vicini a cui un nodo può inviare i propri valori di reputazione locale, vengono scelti quelli il cui valore di reputazione è superiore ad una determinata soglia. Sono state effettuate diverse simulazioni con diversi valori di soglia ma per brevità ne viene mostrata solo una delle più significative. L'attacco mostrato in questo caso è lo *slandering* e il valore della soglia è fissato a 0.7. Il primo grafico, in Figura 4-11, mostra i valori di reputazione di un nodo vittima di diffamazione. Si nota che, a parte la curva relativa al 40 percento di *slander*, il grafico corrisponde a quello mostrato per la versione tradizionale dell'algoritmo. Degna di nota invece è soglia per l'esperimento in analisi. Si rileva che nella curva della versione con soglia appaiono delle piccole oscillazioni. Queste piccole oscillazioni producono l'effetto ondulato visibile nei grafici dell'errore su singolo nodo e totale medio mostrati in Figura 4-12, e sono dovute al fatto che, avvicinandosi al valore 0.7, i nodi vittima di *slandering* usciranno e rientreranno dal sottoinsieme dei vicini a cui ciascun nodo invia i valori locali.

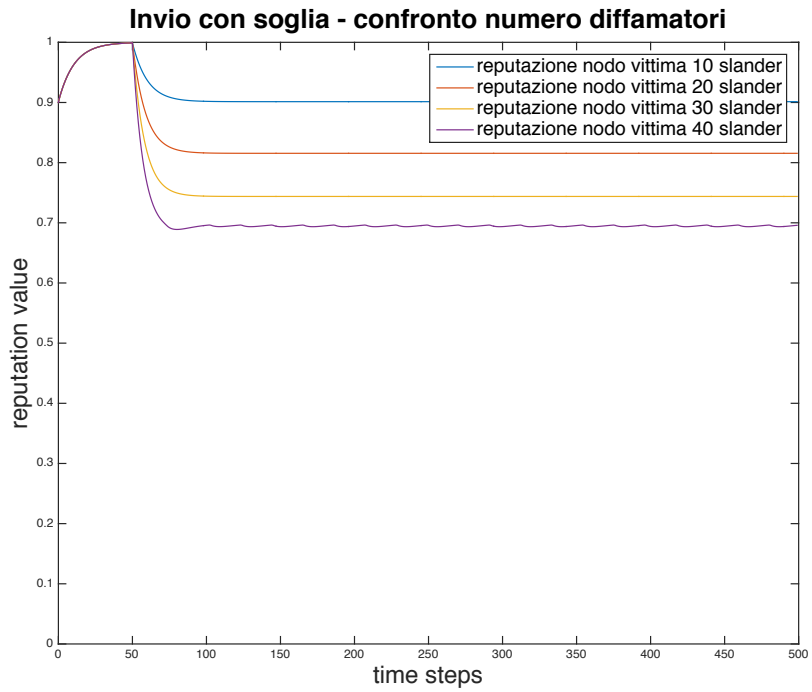

Figura 4-11 Confronto valori di reputazione, versione soglia in invio a 0.7

4.7 Secure Eigentrust

In questo paragrafo vengono illustrate alcune simulazioni relative ad Eigentrust. Quest'ultimo è un algoritmo molto diverso dai quattro algoritmi proposti nel paragrafo precedentemente e, a rigore, non dovrebbe essere vulnerabile ad attacchi di tipo *Slandering* o *Promoting*. Tuttavia, volendo forzare un confronto tra l'algoritmo del caso di studio ed Eigentrust, è stata implementata una particolare variante che sarà discussa nei paragrafi successivi. Nel primo paragrafo sono illustrati i risultati ottenuti con un attacco da parte di traditori. Anche per questi esperimenti, la simulazione prevede ad ogni passo l'interazione tra tutti i nodi

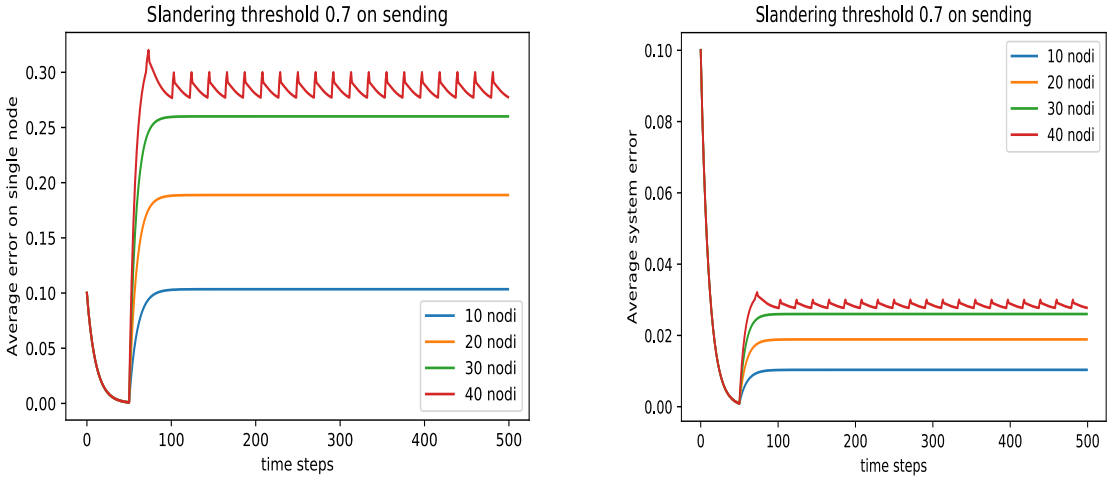

Figura 4-12 Andamento degli errori nella variante con soglia in invio

.

4.7.1 Attacco del traditore

Il grafico mostrato in **Errore. L'origine riferimento non è stata trovata.** mostra l'andamento dei valori di reputazione di un nodo traditore al variare della percentuale di nodi attaccanti presenti nel sistema. Ad un primo sguardo, potrebbe apparire poco comprensibile il fatto che, al crescere del numero di traditori, crescano anche i valori di reputazione. In realtà tale andamento trova spiegazione nel calcolo dei valori c_{ij} secondo la formula:

$$c_{ij} = \begin{cases} \dfrac{\max(s_{ij}, 0)}{\sum_j \max(s_{ij,}0)} & se \quad \sum_j \max(s_{ij}, 0) \neq 0 \\ p_j & altrimenti \end{cases}$$

Notiamo che, al crescere del numero di fornitori che rispondono in modo non cooperativo, il denominatore decresce e di conseguenza il valore del rapporto cresce. Anche il secondo dei due grafici(Figura 4-13b) si spiega con lo stesso ragionamento. I risultati ottenuti dalle simulazioni mostrate confermano la natura relativa dell'algoritmo: preso un valore di reputazione calcolato attraverso Eigentrust, non è possibile stabilire il grado di cooperatività del nodo a cui questo si riferisce. È sensato però confrontare i valori di reputazione di due agenti e, ad esempio, o scegliere il fornitore a cui è associato il valore più alto o scegliere con probabilità proporzionale ai valori. Questo dovuto al fatto che è un algoritmo relativo

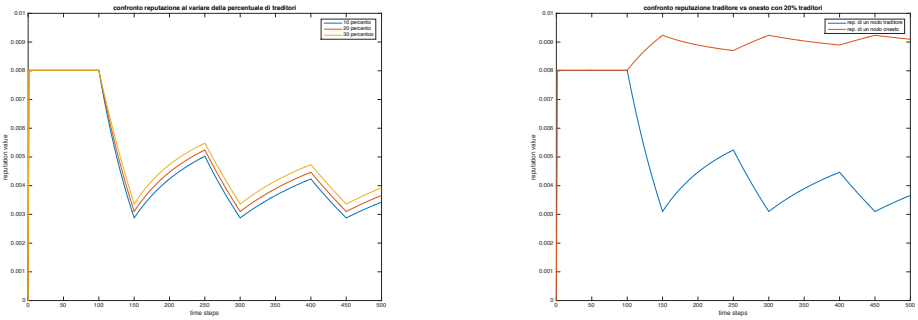

Figura 4-13 a) Valori di reputazione di un nodo traditore al variare della percentuale; b) Adamento dei valori di reputazione di un nodo onesto e di un nodo traditore

4.7.2 Slandering e Promoting

Gli attacchi di *slandering* e *promoting* su cui è stato valutato Eigentrust sono leggermente diversi rispetto alla versione di attacco utilizzata per l'algoritmo del caso di studio. Le specifiche dell'algoritmo infatti, prevedono che la somma dei valori c_{ij} per ogni consumatore sia pari ad uno. Qualora un attaccante diffamatore o promotore dunque volesse alterare i valori di reputazione di un agente, dovrebbe alterare anche i valori relativi ai nodi neutri. In particolare, nella nostra versione adattata di diffamazione, il nodo attaccante assegna un valore nullo alle vittime ed un valore pari al reciproco del numero di vicini neutri ai restanti. Al contrario, un promotore assegna un valore pari al reciproco del numero complessivo di vicini. In questo modo viene garantita la somma unitaria. Va precisato, che l'attacco non viene condotto da parte degli *score managers*, i quali infatti, per la proprietà di anonimità dell'algoritmo, non conoscono l'identità del nodo valutato, ma da parte dei consumatori che hanno effettuato transazioni con il fornitore in oggetto e che devono trasmettere tali informazioni allo score manager opportuno.

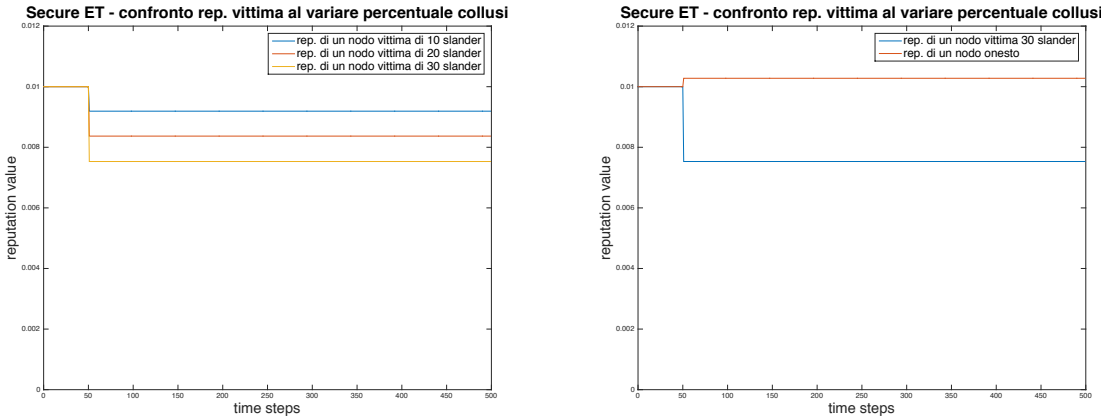

Figura 4-14 Andamento dei valori di reputazione durante un attacco di diffamazione

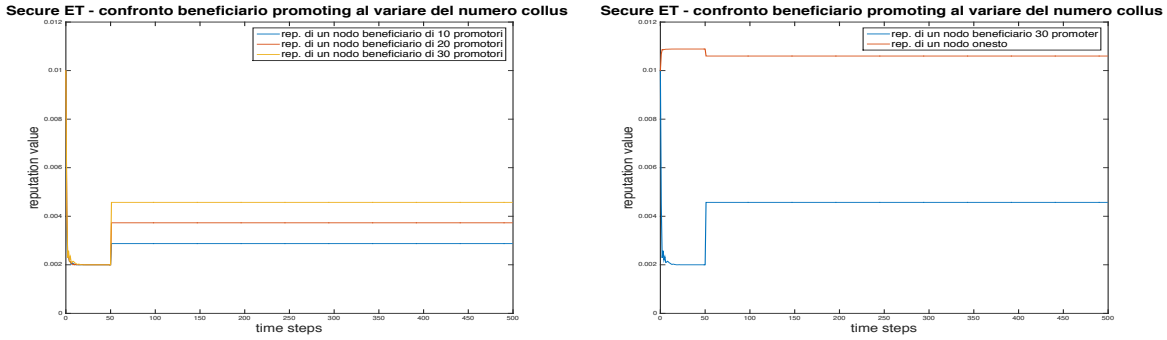

Figura 4-15 Andamento dei valori di reputazione durante un attacco di promoting

Conclusioni

Come esposto nel primo capitolo della trattazione, i sistemi di reputazione hanno un ruolo fondamentale per molte delle applicazioni distribuite che utilizziamo tutti i giorni. È stato anche osservato che progettare un algoritmo di reputazione è tutt'altro che un compito semplice e che di conseguenza diventa fondamentale per i progettisti disporre di simulatori che ne valutino le performance. Attraverso un approfondito studio della letteratura, si è constatata l'assenza di piattaforme o strumenti di questo tipo, e analizzate le caratteristiche desiderate da un tale ambiente di simulazione, si è provveduto prima ad una attenta fase di progettazione e poi allo sviluppo di una piattaforma parallela che permetta la simulazione di diversi tipi di sistemi di reputazione, con diversi scenari simulativi e con una versatilità nelle metriche tale da consentire il confronto anche di RMS di tipo diverso. Sono state effettuato numerose simulazioni per valutare la vulnerabilità di alcuni algoritmi di reputazione presentati in letteratura ai più importanti attacchi a cui sono esposte le applicazioni distribuite. Le simulazioni effettuate dimostrano che il simulatore proposto soddisfa i requisiti richiesti e non necessità di hardware particolarmente prestante. La piattaforma proposta contiene in libreria alcuni sistemi di reputazione già implementati e si presta alla implementazione rapida di nuovi algoritmi e di nuovi comportamenti. Se occorre, il progettista può estendere le classi della piattaforma o implementare le opportune interfacce invece che sviluppare ex novo algoritmi o comportamenti. Per fare ciò, non è richiesto che il programmatore del RMS abbia particolari abilità di programmazione o che addirittura sia un esperto di programmazione che sfrutti il paradigma di comunicazione tra processi a passaggio di messaggi, concetto fondamentale su cui si basa il progetto della tesi. I dati salvati sul file poi, offrono grande versatilità nella realizzazione di metriche calcolabili. Nel nostro caso infatti, come illustrato nel capitolo precedente, abbiamo potuto confrontare l'algoritmo del caso di studio con Eigentrust, possibilità tutt'altro che scontata dato che come è stato spiegato i due algoritmi sono molto differenti. Per aiutare il progettista del sistema di reputazione nella definizione di scenari particolarmente complessi o con elevate casualità, l'ambiente di simulazione è stato anche dotato di un modulo per la generazione degli ambienti di simulazione (nodi e topologia). Si ritiene dunque che la piattaforma proposta offra una soluzione versatile, scalabile e semplice da utilizzare.

Indice delle figure

Figura 2-1 Relazione tra gli agenti 24
Figura 2-2 Gerarchia tipi di comportamento.....25
Figura 2-3 Gerarchia comportamenti di tipo membro RMS.....26
Figura 2-4 Gerarchia comportamenti di tipo fornitori.....27
Figura 2-5 Classe Comportamento come aggregazione delle due classi specifiche.28
Figura 2-6 Diagramma finale semplificato 29
Figura 2-7 Ciclo di vita di un generico nodo 30
Figura 2-8 Overview nodo singolo step.....31
Figura 2-9 Overview singolo step del leadingAgent.....32
Figura 2-10 Overview singolo providersDispatcher 33
Figura 2-11 Esempio di calcolo valori di reputazione.....34
Figura 2-12 Attacco da parte di membri RMS.....35
Figura 2-13 Attacco del traditore 36
Figura 2-14 Attacco del Whitewasher 37
Figura 3-1 Architettura a livelli: dal livello più alto vengono richiamati metodi implementati al livello più basso. 42
Figura 3-2 Strato fisico e strato logico 43
Figura 3-3 Un generico nodo della rete: i comportamenti RT influenzano il comportamento del consumatore, il comportamento PT influenza l'attività del provider. La comunicazione avviene attraverso la Communication Interface 44
Figura 3-4 Coordinamento dei nodi agente da parte del leader 45
Figura 3-5 leader attraverso la CI coordinata gli agenti e si occupa della raccolta della verità assoluta e del coordinamento. 46
Figura 3-6 Overview sistema 47
Figura 4-1 Confronto valori reputazione al variare della percentuale di diffamatori 55
Figura 4-2 Errore su singolo nodo ed errore totale medio per un attacco di slandering 55
Figura 4-3 Valore di reputazione di un nodo beneficiario al variare della percentuale di promotori.....56
Figura 4-4 Errore su singolo nodo ed errore totale medio su nodo beneficiario 56
Figura 4-5 Reputazione nodo traditore. Non si registrano differenze al variare del numero di collusi 58
Figura 4-6 Errore su singolo ed errore di sistema per un attacco del Traditore. Le curve del primo grafico si sovrappongono.....58
Figura 4-7 Simulazione whitewashing, singoli alias con coop. 0 e soglia fissata a 0.3 59

Figura 4-8 Confronto reputazioni nodi beneficiario del promoting nella versione randomizzata in invio.....60
Figura 4-9 Andamento dei due errori in un attacco di Self Promoting nella variante con invio casuale 61
Figura 4-10 Errore su singolo nodo ed errore totale medio per la variante con ricezione casuale.....62
Figura 4-11 Confronto valori di reputazione, versione soglia in invio a 0.7.....63
Figura 4-12 Andamento degli errori nella variante con soglia in invio 64
Figura 4-13 a) Valori di reputazione di un nodo traditore al variare della percentuale; b) Adamento dei valori di reputazione di un nodo onesto e di un nodo traditore 65
Figura 4-14 Andamento dei valori di reputazione durante un attacco di diffamazione 66
Figura 4-15 Andamento dei valori di reputazione durante un attacco di promoting 66

Bibliografia

[1] V. Agate, A. De Paola, S. Gaglio, G. Lo Re e M. Morana, «A Framework for Parallel Assessment of Reputation Management Systems,» in *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016, CompSysTech '16* , New York, 2016.

[2] V. Agate, A. De Paola, G. Lo Re e M. Morana, «A Simulation Framework for Evaluating Distributed Reputation Management Systems,» in *Distributed Computing and Artificial Intelligence, 13th International Conference*, Cham, 2016.

[3] V. Agate, A. De Paola, G. Lo Re e M. Morana, «Vulnerability Evaluation of Distributed Reputation Management Systems,» in *The 10th EAI International Conference on Performance Evaluation Methodologies, VALUETOOLS2016*, 2017.

[4] C. Crapanzano, F. Milazzo, A. De Paola, G. Lo Re, *Reputation management for distributed service-oriented architectures,* In Proceedings of the Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010, pp.

[5] A. De Paola, A. Tamburo, *Reputation management in distributed systems,* In Proceedings of the 3rd International Symposium on Communications, Control and Signal Processing, 2008. ISCCSP 2008, pp. 666-670.

[6] F. Concone, A. De Paola, G. Lo Re, M. Morana, *Twitter Analysis for Real-Time Malware Discovery,* F. Concone, A. De Paola, G. Lo Re, M. Morana. Twitter Analysis for Real-Time Malware Discovery. In Proceedings of the International Annual Conference of AEIT (2017).

[7] S. Gaglio, G. Lo Re, M. Morana, *A framework for real-time Twitter data analysis,* In Journal of Computer Communications, Elsevier, ISSN 0140-3664.

[8] S. Gaglio, G. Lo Re, M. Morana, *Real-Time Detection of Twitter Social Events from the User's Perspective,* In Proceedings of the 2015 IEEE International Conference on Communications (ICC2015).

[9] E. Koutrouli e A. Tsalgatidou, «Reputation Systems Evaluation Survey,» *ACM Computing Surveys (CSUR),* vol. 48(3), n. 35, 2016.

[10] K. Aberer e Z. Despotovic, «Managing Trust in a Peer-2-Peer Information System,» in *Proceedings of the tenth international conference on Information and knowledge management*, Atlanta,Georgia,USA, 2001.

[11] L. Xiong e L. Liu, «PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities,» vol. 16, n. 7, pp. 843-857, 2004.

[12] R. Sherwood, S. Lee e B. Bhattacharjee, «Cooperative peer groups in NICE,» in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies* , San Francisco, 2003.

[13] B. Lagesse, M. Kumar, J. M. Paluska e M. Wright, «DTT: A Distributed Trust Toolkit for pervasive systems,» in *2009 IEEE International Conference on Pervasive Computing and Communications*, Galveston, 2009.

[14] G. Suryanarayana, M. H. Diallo, J. R. Erenkrantz e R. N. Taylor, «Architectural Support for Trust Models in Decentralized Applications,» in *Proceedings of the 28th international conference on Software engineering*, Shanghai, China, 2006.

[15] Y. Zhang, W. Wang e S. Lü, «Simulating Trust Overlay in P2P Networks,» in *Computational Science – ICCS 2007* , 2007.

[16] K. k. Fullam, T. Klos, G. Muller e L. Vercouter, «The Agent Reputation and Trust (ART) testbed,» in *Trust Management: 4th International Conference, iTrust*, Pisa, 2006.

[17] R. Kerr e R. Cohen, «TREET: The Trust and Reputation Experimentation and Evaluation Testbed,» *Electronic Commerce Research,* vol. 10, n. 3-4, pp. 271-290, 2010.

[18] A. Salehi-Abari e W. Tony, «DART: A DISTRIBUTED ANALYSIS OF REPUTATION AND TRUST FRAMEWORK,» *Computational Intelligence,* vol. 28, n. 4, pp. 642-682, 2012.

[19] K. Hoffman, D. Zage e C. Nita-Rotaru, «A survey of attack and defense techniques for reputation systems,» *ACM Computing Surveys (CSUR),* vol. 42, n. 1, 2009.

[20] L. Xiong, L. Liu e M. Ahamad, «Countering Sparsity and Vulnerabilities in Reputation Systems,» 2005.

[21] M. Srivatsa, L. Xiong e L. Liu, «TrustGuard: countering vulnerabilities in reputation management for decentralized overlay networks,» *Proceeding WWW '05 Proceedings of the 14th international conference on World Wide Web,* pp. 422-431, 2005.

[22] H. Yu, Z. Shen, C. Leung, C. Miao e V. R. Lesser, «A Survey of Multi-Agent Trust Management Systems,» *IEEE Access,* vol. 1, pp. 35-50, 2013.

[23] A. Josang e R. Ismail, «The Beta Reputation System,» in *In Proceedings of the 15th bled electronic commerce conference*, 2002.

[24] U. Kuter e J. Golbeck, «Using probabilistic confidence models for trust inference in Web-based social networks,» *ACM Transactions on Internet Technology ,* vol. 10, 2010.

[25] S. D. Kamvar, M. T. Schlosser e H. Garcia-Molina, «The EigenTrust Algorithm for Reputation Management in P2P Networks,» in *Proceeding WWW '03 Proceedings of the 12th international conference on World Wide Web*, Budapest, 2003.