Politecnico di Milano

V Facoltà di Ingegneria
Corso di Laurea in Ingegneria delle Telecomunicazioni

L'ATTACCO ECLIPSE AL PROTOCOLLO CHORD

Relatore: Prof. Giacomo Verticale

Tesi di Laurea di: Stefano Mangioni
Matr. Nr. 734610

Anno Accademico 2009-2010

''The thing you really believe in
always happens;
and the belief in a thing
makes it happen''
Frank Lloyd Wright

Indice

Abstract

P2PSIP is an IETF proposal for the realization of serverless SIP domains, thus based on P2P architectures. In P2PSIP, Chord is considered the standard protocol for resource location in the network.
Switching to a P2P paradigm introduces new types of attack, not present in centralized networks; the *Eclipse Attack*, in particular, can be considered one of the most effective attacks on P2P protocols. Using the collusion of different nodes and manipolating the routing mechanisms, it is able to carry an high quantity of keys to the attacker.

In this work we have quantitatively studied the *Eclipse Attack* to the Chord protocol and we have proposed an algorithm to check whether the network is subject to the attack, starting from locally available information.
We have principally reached four objectives.

(i) We have implemented the *Eclipse Attack* in Chord-based networks, formalizing the basic modifications to the routing and search functions.

(ii) We have verified that, increasing the percentage of malicious nodes in the overlay, the attack performance grows very rapidly; but the growth of the attack is visible also at the increasing of the overlay size and with the growth of the Join and Leave rate.

(iii) We have identified two local parameters influenced by the attack: the amount of stored keys and the number of outgoing queries. Then we have proposed two mechanisms for the evaluation of the network topology in presence of *Eclipse Attack* and we have formulated a mathematical model of the queries in transit in a node, checking its progressive lowering at the growth of the attack incidence.

(iv) We have proposed an algorithm for the detection of the *Eclipse Attack* in a network, based on the local measurement of the transit queries.

Sommario

Il P2PSIP è una proposta IETF per la realizzazione di domini SIP senza server, ovvero basati su architetture P2P. In questa architettura, Chord è considerato il protocollo standard per la localizzazione delle risorse all'interno della rete.
Passare ad un paradigma P2P, però, introduce nuovi tipi di attacchi, non presenti in reti centralizzate. L'*Eclipse Attack*, in particolare, può essere considerato uno degli attacchi più efficaci ai protocolli P2P; sfruttando la collusione di diversi nodi e manipolando i meccanismi di instradamento, infatti, riesce a convogliare una frazione elevata di chiavi verso l'attaccante.

In questo lavoro è stato studiato in modo quantitativo l'attacco Eclipse al protocollo Chord ed è stato proposto un algoritmo per valutare, partendo da informazioni disponibili localmente, la soggettività della rete all'attacco.
Sono stati raggiunti principalmente quattro obiettivi.

(i) È stata esplicitata l'implementazione dell'*Eclipse Attack* all'interno di reti basate sul protocollo Chord, formalizzando le regole basilari relative alla modifica delle funzionalità di routing e di ricerca.

(ii) È stato verificato che, all'aumentare della percentuale di nodi maligni nell'overlay, le prestazioni dell'attacco crescono molto rapidamente, ma la crescita delle prestazioni è visibile anche all'aumentare delle dimensioni dell'overlay e al crescere del tasso di Join e Leave.

(iii) Sono stati identificati parametri locali influenzati dall'attacco: le chiavi memorizzate e le ricerche effettuate. Si sono poi proposti due meccanismi per la valutazione della topologia del grafo in caso di presenza di *Eclipse Attack*, ed è stato formulato un modello matematico per la descrizione delle chiavi in transito, verificando il suo progressivo abbassamento al crescere dell'incidenza dell'attacco.

(iv) È stato proposto un algoritmo per la rivelazione dell'attacco Eclipse all'interno di una rete basandosi esclusivamente sulla misura locale delle chiavi transitanti.

Capitolo 1

Introduzione

Il P2PSIP, versione P2P del protocollo SIP utilizzato per la segnalazione di sessioni multimediali su IP, è un protocollo attualmente in fase '*work in progress*' ma che rappresenterà sicuramente una soluzione molto considerata e sviluppata nei prossimi anni.
I fondamentali pregi del P2PSIP rispetto al protocollo SIP sono l'alta flessibilità, la scalabilità e l'affidabilità tipiche delle reti P2P, oltre alle ottime prestazioni a costi decisamente contenuti, dovuti all'assenza di controllori centralizzati.
L'uso del paradigma P2P, però, porta a conseguenze che devono essere studiate a fondo per essere gestite nel modo più corretto.
Un problema introdotto dall'applicazione del P2P, ovvero dalla scomparsa di un server di gestione centralizzato, è la sicurezza: all'interno di un sistema P2PSIP, e quindi in generale in ogni sistema di tipo P2P, sarà più difficile riuscire a garantire la sicurezza a causa della nascita di nuove tipologie di attacco e dell'incremento di efficienza di attacchi già presenti nei sistemi centralizzati.

È proprio sulla sicurezza nelle reti P2P che si incentra questo lavoro di tesi: è stata effettuata un'indagine quantitativa per valutare l'impatto di un particolare attacco, noto come *Eclipse Attack*, in reti di diverse dimensioni per poi proporre un test per la rivelazione dell'attacco partendo esclusivamente da misure locali.

L'attacco Eclipse è da considerarsi uno degli attacchi più efficaci nei sistemi di tipo P2PSIP ed, in generale, in tutti i sistemi basati su di un paradigma P2P. L'idea base è quella di catturare il maggior quantitativo possibile di chiavi nel sistema controllando esclusivamente una porzione relativamente piccola di nodi per ottenere, con una spesa esigua, un ampio controllo sulla rete.
L'*Eclipse Attack* può essere visto come l'estensione di un attacco più semplice noto come attacco Sybil, che è quindi preliminare all'attacco Eclipse. Il *Sybil Attack* consiste nell'inserimento, da parte di un attaccante, di una frazione f di nodi nell'overlay necessaria per riuscire a catturare una percentuale circa pari ad f di chiavi. Partendo da questa situazione e modificando il comportamento dei nodi collusi, si può riuscire ad ottenere l'attacco Eclipse che permette il controllo di una frazione di chiavi superiore ad f.

Il P2PSIP può utilizzare differenti protocolli P2P per la costruzione dell'overlay, anche se a questo scopo è stato definito uno standard, chiamato RELOAD (*REsource LOcation And Discovery*) il quale però, oltre ad essere estremamente complesso, attualmente è ancora in una fase preliminare. Si è allora cercato un protocollo con funzionamento molto simile in modo da poter trarre delle conclusioni relative a reti P2P generiche ma coerenti anche per il P2PSIP. È stato allora affrontato lo studio dell'attacco sul protocollo Chord, che, oltre ad essere molto simile a RELOAD, rappresenta attualmente uno dei protocolli base per la gestione delle operazioni di ricerca di chiavi e di risorse all'interno di sistemi P2PSIP, ed in generale di sistemi P2P; in questo modo, lo studio sulla sicurezza effettuato ha assunto una valenza globale, non limitandosi esclusivamente alla versione P2P del protocollo SIP ma estendendosi a tutti i sistemi *Chord-based*.

Per l'implementazione dell'attacco all'interno del protocollo ci si è avvalsi di un simulatore, noto come OMNeT++/OverSim, che proponeva già l'organizzazione dell'overlay in differenti strutture, tra le quali quella ad anello propria di Chord; quindi, partendo dall'implementazione del protocollo Chord, si è introdotto l'*Eclipse Attack* basandosi su alcuni punti chiave definiti in letteratura, anche se riferiti ad overlay generici. La definizione dell'attacco ha richiesto uno studio approfondito e differenti test in quanto lo stato dell'arte non forniva informazioni dettagliate per l'implementazione dell'attacco sul protocollo considerato.
Come visto in precedenza, l'attacco Eclipse è il risultato di una modifica del comportamento dei nodi sui quali era già in atto un attacco Sybil; il primo passo è stato quindi quello di condurre un *Sybil Attack* ed utilizzarlo poi come punto di partenza per il lancio dell'attacco Eclipse, intervenendo sul comportamento dei nodi maligni tramite due differenti alterazioni, relative ad instradamento e ricerca, rispetto al comportamento standard:

- è stata modificata la funzione di routing, in modo tale da permettere l'inquinamento delle finger table e quindi favorire l'instradamento delle ricerche verso l'attaccante;
- è stata modificata l'operazione di ricerca, in modo tale da permettere a nodi collusi relativi ad un attaccante di catturare chiavi di passaggio.

Il lavoro che si è effettuato è stato diviso, dal punto di vista logico, in differenti *step*; innanzitutto è stato implementato l'attacco e ne è stata data una descrizione quantitativa, poi sono stati realizzati e validati un modello matematico ed un algoritmo per l'identificazione dell'*Eclipse Attack* basandosi esclusivamente su misure locali. Quest'ultimo obiettivo è risultato fondamentale in quanto nella realtà ogni singolo nodo non è onniscente nella rete, ma conosce esclusivamente una quantità limitata di informazioni e, di conseguenza, deve riuscire ad effettuare l'identificazione senza avere la necessità di accedere ad altre misurazioni a priori sconosciute.

Per quanto riguarda l'implementazione dell'attacco, come già detto in precedenza, si è partiti da un *Sybil Attack* per poi, tramite le modifiche prima riportate, impo-

stare un *Eclipse Attack*; per riuscire ad estrarre misure quantitative sull'attacco, poi, sono state eseguite differenti simulazioni al variare delle condizioni al contorno ed in particolare si è verificato:

- l'incremento di efficacia dell'attacco all'aumentare della percentuale di nodi maligni e delle dimensioni dell'overlay;
- la variazione di efficacia dell'attacco al variare del numero di Join e di Leave; questo è un risultato fondamentale in quanto, come in tutti i sistemi P2P, anche nei sistemi controllati dal protocollo Chord, all'aumentare del rate di Join e di Leave dei nodi, ovvero in caso di rete dinamica, aumenta l'efficienza dell'attacco.
- la variazione della ripartizione delle chiavi ad ogni singolo nodo al variare della percentuale dei nodi maligni;

In più si è cercato di studiare, utilizzando due metodi differenti, le caratteristiche strutturali del grafo in modo da riuscire ad apprezzare le modifiche imposte dall'attacco Eclipse alla topologia di rete. Questi due meccanismi hanno fornito risultati differenti in quanto sono basati sullo studio di parametri diversi: attuando la modifica nota come *Google Matrix* non si è riuscito ad apprezzare un significativo cambiamento nel grafo, mentre analizzando la matrice laplaciana si è potuto osservare l'effetto dell'attacco sulle strutture di rete.

Per quanto riguarda la realizzazione dell'algoritmo per la rivelazione dell'attacco avendo a disposizione esclusivamente misure locali, ci si è basati sull'analisi delle chiavi in transito, ovvero quelle chiavi che, dopo essere state instradate ad un nodo qualsiasi nell'overlay, vengono reinstradate al suo corretto successore in quanto non hanno raggiunto la reale destinazione. È stato scelto questo metodo di analisi, rispetto al controllo di ricerche effettuate o chiavi memorizzate in quanto, a differenza degli altri meccanismi, il controllo delle chiavi in transito assicura un'indagine più precisa e robusta. A questo scopo è stato prima formalizzato un modello matematico per la descrizione dell'operazione di ricerca nelle reti e dell'andamento delle chiavi in transito, sia in caso di overlay Chord non soggetto ad *Eclipse Attack* che in caso di presenza di nodi maligni e, successivamente, lo si è verificato tramite differenti simulazioni; poi è stato proposto un algoritmo per l'identificazione, da parte di un qualsiasi nodo dell'overlay, dell'eventuale presenza di attacco Eclipse esclusivamente attraverso l'analisi di informazioni locali e senza accedere ad ulteriori misurazioni.

Il documento è strutturato come segue:
Il secondo capitolo descrive brevemente il protocollo SIP, utile per riuscire poi a comprendere in modo più semplice le principali differenze tra esso e la versione P2P.
Il terzo capitolo descrive il protocollo P2PSIP, analizzato come possibile *usage* di RELOAD, soffermandosi sull'architettura di rete e sulle problematiche relative alla

sicurezza.
Il quarto capitolo analizza invece il protocollo Chord, ovvero il protocollo standard per la gestione dell'overlay e per la ricerca all'interno di reti di tipo P2P. In questo capitolo è riportata innanzitutto la struttura protocollare ed inoltre l'implementazione del protocollo stesso realizzata con il simulatore OMNeT++/OverSim, in modo da poter comprendere con più facilità i vari risultati pratici presenti nei capitoli successivi.
Il quinto capitolo offre una panoramica degli attacchi possibili all'interno di reti P2P, tra i quali il *Sybil Attack* e l'*Eclipse Attack*, analizzati considerando un overlay generico; uno studio di questo tipo è stato necessario per comprendere le caratteristiche degli attacchi e poterli poi analizzare e sviluppare in riferimento al protocollo Chord.
Il sesto capitolo descrive le modifiche attuate agli algoritmi per l'implementazione dell'attacco Eclipse all'interno di overlay Chord; tratta quindi l'analisi delle varie considerazioni assunte per imporre comportamenti differenti tra nodi corretti e maligni ed i vari test eseguiti per mostrare l'efficienza dell'attacco stesso.
Il settimo capitolo riguarda invece la formalizzazione del modello matematico per la descrizione dell'operazione di ricerca in overlay Chord soggetti e non ad *Eclipse Attack* e la definizione di un algoritmo per il riconoscimento della presenza dell'attacco, valutando esclusivamente i parametri locali in possesso dei singoli nodi.
L'ottavo capitolo, infine, presenta l'analisi dei risultati ottenuti e dei possibili sviluppi futuri del lavoro.

Capitolo 2

Il protocollo SIP

In questo primo capitolo verrà analizzato molto rapidamente il protocollo SIP, *Session Initiation Protocol*, in modo da poter poi comprendere con più facilità la versione P2P (*peer-to-peer*) e cogliere quindi con più semplicità le differenze e le principali difficoltà implementative.

Il protocollo SIP [1,2] è un protocollo di segnalazione che consente l'instaurazione, la modifica ed infine l'abbattimento di sessioni multimediali, notifiche di presenza e messaggi istantanei sulla rete internet; è uno standard *IETF* ed è quindi impiegato principalmente per applicazioni di telefonia su IP. Da molti è considerato un'alternativa molto interessante ad H.323 [3] (standard *ITU-T*) ma, rispetto alla tecnologia alternativa, SIP rappresenta sicuramente una soluzione più flessibile, più semplice e più facile da implementare, supportando anche meglio sia le periferiche intelligenti che l'implementazione di applicazioni avanzate. Questi fattori sono stati determinanti nello sviluppo della tecnologia poiché rappresentano le tipiche richieste fatte da ogni operatore in quanto permettono uno sviluppo veloce e semplice di apparati e di servizi.
Le principali funzionalità di SIP sono:

- localizzazione del chiamato, ovvero dell'utente destinazione della chiamata che sta per essere effettuata.
- determinazione della capacità degli utenti, impostata tramite il setting dei media e di parametri da utilizzare durante la chiamata.
- individuazione della disponibilità dell'utente nello stabilire la chiamata, tramite un meccanismo di scambio di un set di messaggi.

SIP nasce dalla necessità di creare un protocollo standard per la gestione delle molte applicazioni internet che richiedono la creazione ed il controllo di sessioni, ovvero di scambi di dati tra gruppi di partecipanti. Il problema di queste sessioni è che, nella pratica, i partecipanti possono muoversi, essere raggiungibili attraverso diversi nomi e comunicare in molteplici modi; SIP risolve questi problemi attraverso una gestione molto flessibile delle diverse applicazioni.

2.1 L'architettura SIP

SIP è un protocollo di segnalazione che regola setup, modifica ed abbattimento di sessioni multimediali. Esso, in combinazione con altri protocolli, è utilizzato per la descrizione delle caratteristiche della particolare sessione ai suoi potenziali partecipanti.
Sebbene supporti la possibilità di avvalersi dell'utilizzo di qualsiasi protocollo di trasporto per i media, solitamente si avvale di RTP (*Real Time Transport Protocol*).
La segnalazione SIP è da considerarsi separata dai media; questo è di fondamentale importanza in quanto, mentre il percorso della segnalazione potrebbe passare attraverso uno o più server, il percorso dei media potrebbe essere più diretto e quindi più rapido.

Lo standard definisce due entità di rete base ovvero i client ed i server, i quali possono essere configurati sia a livello software che hardware.

Il client, noto anche come User Agent (UA) client, è un programma applicativo che invia richieste SIP; il server, invece, è l'entità che risponde a queste richieste ed è per questo motivo che SIP può essere considerato un protocollo di tipo client-server.
Esistono quattro differenti tipologie di server.

- *Proxy server*; lavora in modo molto simile ad un server per l'accesso in una LAN ovvero, nel momento in cui il client invia richieste al proxy, esso fornirà una risposta o inoltrerà la richiesta ad un altro server, magari dopo aver attuato modifiche o traslazioni. I server che ricevono queste richieste le interpreteranno come formulate dal proxy e quindi esso dovrà saper sia ricevere che inviare ed è per questo motivo che avrà sviluppata al suo interno sia la funzionalità di client che quella di server.
  Un proxy server può essere *stateless* o *stateful*; nel primo caso processa ogni messaggio SIP basandosi esclusivamente sul suo contenuto: ogni messaggio viene analizzato, processato e trasmesso mentre non viene memorizzata nessuna informazione sul dialogo e non è possibile la ritrasmissione del messaggio stesso. Un proxy stateful, invece, tiene traccia del dialogo, usando informazioni in suo possesso per processare richieste e risposte; ad esempio, nel caso in cui debba essere inoltrata una richiesta, il proxy manterrà un riferimento temporale e, se necessario, ritrasmetterà il messaggio.

- *Redirect server*; è una particolare tipologia di server che accetta richieste SIP e fornisce in risposta al mittente, se disponibile, il nuovo indirizzo del destinatario; implementa quindi un procedimento di ridirezione della chiamata.

- *User Agent server*; ha la fondamentale funzionalità di accettare richieste SIP e di contattare l'utente; è quindi l'estremo opposto nella chiamata SIP rispetto all' UA client ed in ogni apparato SIP sono presenti sia il client, per poter effettuare una richiesta, che il server, per poter rispondere a richieste entranti.

- *Registrar server*; è il server utilizzato per tenere traccia della presenza dei vari User Agent all'interno di un dominio. Tipicamente un UA si registra su questa applicazione in modo da facilitare poi il compito di instradamento del proxy che, interrogando il Registrar, potrà riuscire a scoprire se l'utente cercato è in linea e su quale apparecchio è raggiungibile. Nella maggior parte dei casi il processo di registrazione richiede anche un'autenticazione al fine di evitare furti di identità.

2.2 Struttura dei messaggi ed indirizzamento

La segnalazione all'interno del protocollo SIP avviene attraverso l'invio e la ricezione di messaggi testuali, che vengono scambiati tra i vari apparati di rete. Quindi la struttura dei messaggi SIP è particolare e la sintassi è detta *text-based*, con una forma simile a quella utilizzata in *HTTP* (*Hypertext Transfer Protocol*). Questo consente agli operatori semplicità di implementazione ed agli utilizzatori del protocollo immediata comprensione delle funzionalità di un messaggio e delle sue caratteristiche; la struttura testuale introduce però anche delle problematiche principalmente in relazione al maggior consumo di banda rispetto, per esempio, ad H.323, caratterizzato da messaggi in forma binaria.
I messaggi SIP possono essere suddivisi in messaggi di richiesta, prodotti dal client verso il server, e messaggi di risposta, o messaggi di stato, inviati dal server verso il client. Indipendentemente dalla tipologia, comunque, ogni messaggio è composto da:

`start line`
`message header`
`CRLF`
`message body` (opzionale)

La *start line* è la linea iniziale di ogni messaggio ed è strutturata in modo differente a seconda della tipologia del messaggio, ed infatti se si tratta di un *request message* si avrà una *request line* mentre se è un *response message* sarà una *status line*. Per quanto riguarda il *message header*, esso contiene informazioni generali o più specifiche riguardo a richieste e risposte. Invece, mentre il *CRLF* è una semplice riga vuota per la separazione dell'header dall'eventuale corpo del messaggio, il *message body* è la parte opzionale del messaggio scritta in *SDP* (*Session Descrtption Protocol*) e dedicata alla negoziazione dei media che avviene, come detto, durante l'instaurazione della chiamata.

Esistono quindi, come già accennato precedentemente, due tipologie di messaggi:

- *messaggi di richiesta*, introdotti da una request line. Sono detti anche 'metodi' e hanno lo scopo di informare il ricevente riguardo ad una specifica azione da compiere; i principali messaggi di questo tipo sono `Invite`, usato per stabilire

una sessione multimediale, `Register`, che ha lo scopo di registrare un particolare client ad un Registrar server, `Ack`, per il riscontro di una risposta, `Bye`, per la chiusura di una sessione, `Option`, per interrogare un UA o un server e `Cancel`, per terminare i tentativi di chiamata non ancora conclusi.

- *messaggi di risposta*, introdotti da una status line. Essi sono generati da un UA server o da un SIP server a seguito della ricezione di un messaggio di richiesta. Vengono rappresentati tramite un codice ed una descrizione testuale e possono appartenere a sei principali classi ovvero *messaggi provisional*, messaggi facoltativi che danno informazioni sullo stato della transizione, *messaggi di conferma*, *messaggi di call forwarding*, che permettono la redirezione e *messaggi di failure* che possono essere legati alla richiesta, ovvero al client, al server o essere globali, generici.

Come in ogni protocollo di segnalazione, richieste e risposte sono sempre inviate verso un particolare indirizzo ed il SIP address è noto come *SIP URI* (*Uniform Resource Indicator*) che assume la forma `user@host`, ovvero graficamente risulta essere molto simile ad un indirizzo e-mail; in molti casi, infatti, l'indirizzo SIP di un utente può essere 'indovinato' proprio partendo dalla sua e-mail.
Sebbene i due indirizzi siano molto simili, presentano comunque delle discordanze ed infatti, mentre una SIP URI potrebbe essere `sip:user@home.net`, l'indirizzo e-mail sarà `mailto:user@home.net`, ovvero un *URL* (*Uniform Resource Locator*).
SIP è utilizzato soprattutto per sessioni multimediali, che spesso possono includere voce, e per questo motivo è necessaria la capacità di fare interworking ovvero di interfacciarsi con le reti a circuito tradizionali attraverso un gateway. Per questo motivo il SIP URI potrebbe avere la porzione `user` espressa sotto forma di numero telefonico legato ad un particolare operatore del mondo esterno; un esempio di indirizzo potrebbe quindi essere anche `sip:3344556789@telco.net`.

2.3 La chiamata base SIP

Senza addentrarsi nei particolari, non necessari ai fini di questo lavoro, stabilire una chiamata SIP è relativamente semplice e la struttura, come si può notare in Figura 2.1, è abbastanza lineare.
Il processo inizia con un messaggio di `Invite`; con questo messaggio il chiamante inviterà il chiamato a partecipare ad una sessione. La risposta a questo messaggio non è sempre univoca ma dipenderà dalla disponibilità del chiamato il quale, però, nella situazione standard, risponderà prima con un messaggio di `Ringing`, per segnalare che la richiesta ha raggiunto la destinazione desiderata, e poi con un messaggio di `Ok` per segnalare l'avvenuta accettazione. La chiamata, però, sarà realmente attiva solo nel momento in cui il destinatario riceverà un `Ack`, inoltrato da chiamante a chiamato dopo la ricezione del messaggio di `Ok` e fondamentale per rendere attivo il *dialog ID*, ovvero una terna di parametri specifici direttamente collegati alla chiamata.

Per quanto riguarda lo scambio dei media, che potrebbero essere il semplice parlato o, in aggiunta, anche il video, si ha l'utilizzo del protocollo *SDP*, il quale andrà a costituire il corpo dei messaggi SIP e permetterà quindi, in real time con l'instaurazione della chiamata, la negoziazione dei codec.
La chiusura della chiamata, invece, verrà formalizzata tramite un messaggio di `Bye` seguito da un messaggio di `Ok`, cioè di conferma.

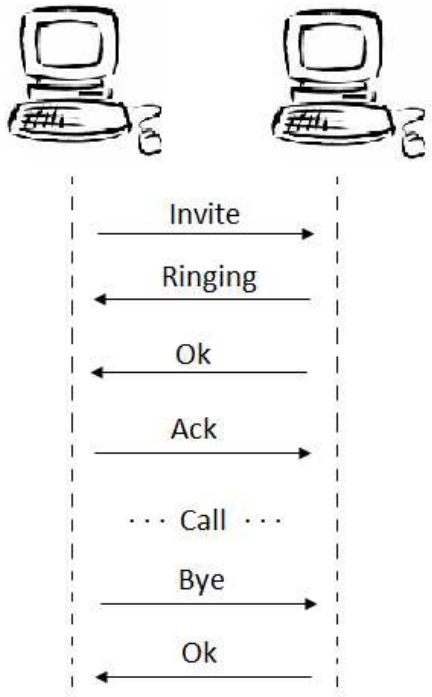


Figura 2.1: Esempio di chiamata base in *SIP*

2.4 Differenze con gli altri protocolli di segnalazione

Il protocollo SIP presenta, rispetto agli altri protocolli di segnalazione, una serie di vantaggi che l'hanno portato, come già accennato in precedenza, ad essere il protocollo di telefonia su IP più sviluppato e più utilizzato. Accanto a queste proprietà, però, sono presenti anche complessità che rendono piuttosto difficoltose le varie operazioni di sviluppo.

Innanzitutto SIP offre molta *flessibilità* alla chiamata; infatti, come tutti i protocolli di segnalazione, deve fornire dei mezzi attraverso i quali poter iniziare e poi porre fine ad una chiamata. La differenza con gli altri protocolli di segnalazione è legata al fatto che SIP fornisce queste funzionalità in modo molto flessibile andando, per esempio, a suddividere l'operazione di scambio dei media dall'operazione di setting della chiamata, che verranno eseguite contemporaneamente ma in realtà sono da

considerarsi logicamente separate, oppure anche solo permettendo un'ampia varietà di protocolli per il trasporto. Quindi SIP offre più flessibilità rispetto ad altri protocolli di telecomunicazioni e questo può essere sfruttato per abilitare con più facilità servizi o impostazioni personalizzate; questa caratteristica è fondamentale per lo sviluppo e l'utilizzo del protocollo ma il prezzo da pagare è da ricercarsi in una struttura particolare del codice che porta alla nascita di due particolari problematiche.

- *Complessità protocollare*
  Dall'analisi del protocollo SIP si può facilmente notare come le molteplici funzionalità impongano all'implementatore un difficile lavoro di documentazione sullo standard. La complessità di SIP è dovuta al fatto che esso si pone come protocollo di inizializzazione generico, in grado di poter supportare ed essere compatibile con vari protocolli odierni e, per quanto possibile, futuri. La sua grande flessibilità è quindi anche il suo principale punto debole poiché il dover essere resistente a tutte le possibili varianti provoca una complessità protocollare molto marcata.

- *Robustezza protocollare*
  Punto critico di ogni protocollo di segnalazione su IP è la robustezza protocollare in quanto il meccanismo di ideazione e di progetto presenta sempre, esplicitamente o implicitamente, debolezze protocollari. Con il termine robustezza protocollare, quindi, vengono considerati tutti gli ambiti che possono rendere sicuro o insicuro un protocollo, a partire proprio dalle sue scelte progettuali.
  Ciò che rende problematica la gestione del protocollo SIP, e più propriamente dei suoi messaggi, è la loro struttura testuale; infatti, in questo modo, si lascia grande libertà al mittente di una richiesta o di una risposta poiché potrebbe compilare il tutto nel modo che preferisce.
  Dal lato del mittente del messaggio il problema è la codifica, in quanto non è semplice codificare un messaggio SIP nel modo corretto poiché è necessaria una conoscenza approfondita dello standard. La decodifica è, invece, l'operazione in assoluto più critica in quanto in ricezione l'applicazione deve poter analizzare il messaggio ricevuto, comprenderlo ed agire di conseguenza.

I messaggi sono quindi il punto più critico del protocollo SIP. Oltre a ciò che è stato già analizzato, bisogna considerare anche il fatto che essi possono includere un ampio numero di campi opzionali che potranno essere utilizzati per contenere informazioni specifiche legate ai singoli utenti. In questo modo potranno essere condivise svariate tipologie di informazioni e questo potrà rendere il set up della chiamata anche un'operazione di scambio di informazioni, più o meno importanti, tra chiamante e chiamato. Per esempio SIP permette, tramite il messaggio di `Busy`, la comunicazione dello stato di occupazione da parte del destinatario della chiamata nei confronti del mittente ed in contemporanea il lasso di tempo di attesa dopo il quale poter riprovare ad instaurare la chiamata.

Infine, come già detto ma fondamentale poiché rappresenta la caratteristica chiave di SIP, i messaggi hanno una forma 'testuale', ovvero sono scritti in un linguaggio simile al parlato, e questo garantisce un grado di comprensione molto elevato anche se accompagnato da una complessità superiore rispetto agli altri protocolli di segnalazione su IP.

In generale, comunque, si può affermare che lo sviluppo del protocollo SIP sta progredendo molto velocemente e ad oggi rimpiazza quasi completamente il protocollo H.323 in tutte le principali applicazioni di telefonia VOIP.

Capitolo 3

Il protocollo RELOAD e il P2PSIP

Il protocollo RELOAD (*REsource LOcation And Discovery*) [4] è un protocollo di segnalazione di tipo P2P che fornisce ai suoi client servizi di storage astratto e di messaggistica avvalendosi di un set di peer coordinati che costituiscono la rete di overlay. Attualmente RELOAD non è implementato in ambito applicativo ma è ancora in una fase di *'work in progress'*; gli investimenti che si stanno effettuando, però, faranno si che nei prossimi anni possa essere realistico uno sviluppo massiccio del protocollo ed un'applicazione in prodotti con fini commerciali.
La caratteristica fondamentale di RELOAD è quella di essere stato disegnato appositamente per supportare reti P2PSIP (*peer-to-peer Session Initiation Protocol*) [5–7], anche se può essere poi utilizzato per altre applicazioni con simili richieste grazie alla possibilità di realizzare appositamente nuovi *usage* per specificare le caratteristiche base per il supporto di applicazioni specifiche. In questo capitolo verranno approfonditi gli aspetti fondamentali di RELOAD ed in particolare quelli legati all'applicazione P2PSIP, in modo da analizzare le differenze e le principali complessità in relazione al protocollo SIP standard introdotto nel capitolo precedente.

Ai fini dello sviluppo del lavoro di tesi, però, non sarà utilizzato il protocollo RELOAD a causa della sua elevata complessità, ma ci si baserà su di un protocollo più semplice ma con un comportamento molto simile, che sarà introdotto nel capitolo seguente.

3.1 Concetti preliminari

RELOAD offre un servizio di rete generico ed auto-organizzato, che permette a qualsiasi nodo di instradare in modo efficiente messaggi verso altri nodi e, successivamente, di immagazzinare e recuperare efficientemente dati dell'overlay. RELOAD offre delle funzionalità che sono fondamentali per un protocollo P2P di successo:

- *sicurezza*; una rete P2P deve stabilire connessioni tra set di nodi solitamente non autenticati. A questo scopo, RELOAD sfrutta un server centrale d'iscrizione per fornire credenzialità ad ogni peer, credenzialità che verranno poi utilizzate

per autenticare ogni singola operazione. Tramite questo meccanismo si riesce a ridurre il set di possibili attacchi.

- *modelli d'uso*; RELOAD è stato creato per supportare un'ampia varietà di applicazioni, inclusa la comunicazione P2P multimediale con il protocollo SIP che rappresenta, come detto, il principale ambito di sviluppo. Oltre ai modelli già creati e sperimentati, però, RELOAD permette la definizione di nuove applicazioni d'uso, ognuna delle quali può dichiarare le sue tipologie di dati e le regole d'uso. Questa funzionalità permette l'utilizzo di RELOAD in modo dinamico, anche se necessita una fase di documentazione sui dettagli di ogni singola applicazione.

- *attraversamento di NAT* (*Network Address Translation*); RELOAD è stato disegnato appositamente per il funzionamento in ambienti in cui la maggior parte dei nodi sono sotto il controllo di NAT o di firewall; questo è necessario in quanto in reti P2P quasi la totalità dei peer risulta essere nattata. Le operazioni di *NAT traversal* sono parte integrante del design base ed includono l'uso di ICE (*Interactive Connectivity Establishment*) per stabilire nuove connessioni o per attraversare in modo efficiente l'overlay.

- *routing ad elevate performance*; il protocollo deve garantire che i peer partecipino alle operazioni d'instradamento di messaggi nella rete verso altri nodi; questo, però, introduce ritardi sia sui peer di transito sia su quelli di destinazione in termini di banda e di potenza di processo. RELOAD è stato quindi strutturato con un header semplice e leggero in modo da minimizzare lo sforzo richiesto ai peer di passaggio.

- *algoritmi di overlay 'pluggabili'*; RELOAD è stato disegnato con un'interfaccia astratta in modo da semplificare l'implementazione di una grande varietà di algoritmi relativi ad overlay strutturati e non.

3.1.1 Settaggi di base

La struttura standard di RELOAD è basata su di una *RELOAD overlay instance*, che consiste in un set di nodi arrangiati in una struttura a grafo parzialmente connesso nel quale, ad ogni nodo dell'overlay, è assegnato un ID numerico che, insieme alla specifica tipologia di rete utilizzata, determina la sua posizione nel grafo ed il set di nodi a cui è connesso.

Siccome il grafo è non pienamente connesso, quando un nodo cerca di inviare un messaggio ad un altro potrebbe avere bisogno di un meccanismo di instradamento del messaggio stesso nella rete. Nell'esempio in Figura 3.1, il nodo 10 può comunicare direttamente sia con il nodo 20 che con il nodo 40 ma non con il nodo 70. Per riuscire a mandare un messaggio a quest'ultimo, allora, il nodo in questione dovrà inviare il messaggio al nodo 40 con tutte le indicazioni necessarie ad un corretto instradamento.

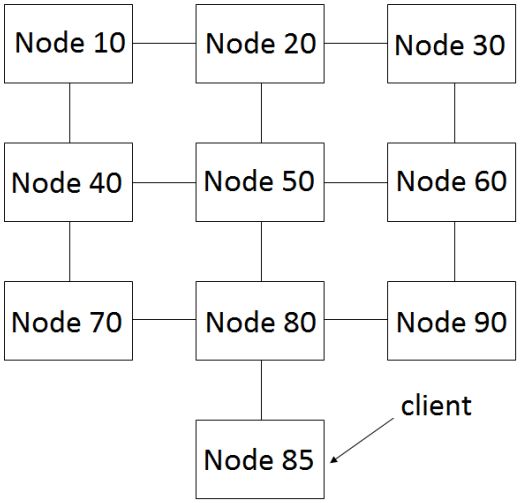


Figura 3.1: Esempio di grafo di rete nel protocollo *RELOAD*

L'algoritmo di overlay varia al cambiare della tipologia di grafo di connessione ma l'idea generale alla base di tutte le strutture di rete è quella di permettere a ciascun nodo di connettersi con tutti gli altri nodi del grafo utilizzando il minor numero possibile di hop.

La rete RELOAD, però, non è solamente una rete di messaggistica ma è anche una rete di storage; qui i dati sono memorizzati sulla base di indirizzi numerici che occupano lo stesso spazio degli identificativi di nodo. Ciascun nodo è responsabile del salvataggio di una particolare frazione di dati associata al suo identificativo ed infatti, considerando ancora l'esempio in Figura 3.1, se la regola di riferimento dice che ogni nodo è responsabile del salvataggio dei dati che hanno indirizzo minore o uguale al loro ID di nodo ma maggiore dell'ID di nodo inferiore, allora il nodo 20 sarà responsabile del salvataggio dei valori con indirizzo compreso tra 11 e 20.

RELOAD, inoltre, supporta i client, ovvero quei nodi che hanno ID di nodo ma non partecipano al routing o allo storage. Sempre nell'esempio di Figura 3.1, il nodo 85 è un client mentre i nodi che non sono definiti come client saranno semplicemente peer; esso può instradare messaggi nella rete tramite il nodo 80, ovvero il peer a cui è direttamente collegato, ma nessun altro nodo potrà instradare messaggi tramite il client. Inoltre, il nodo 90 sarà responsabile del salvataggio di tutti i valori con indirizzo compreso tra 81 e 90, in quanto a livello di storage è come se il nodo 85 non esistesse. Questa basilare differenziazione tra nodi è di fondamentale importanza per tutte le operazioni a livello applicativo e quindi anche per il SIP usage.

3.1.2 Architettura di rete

Dal punto di vista architetturale, RELOAD è suddiviso in diversi livelli, come mostrato in Figura 3.2.

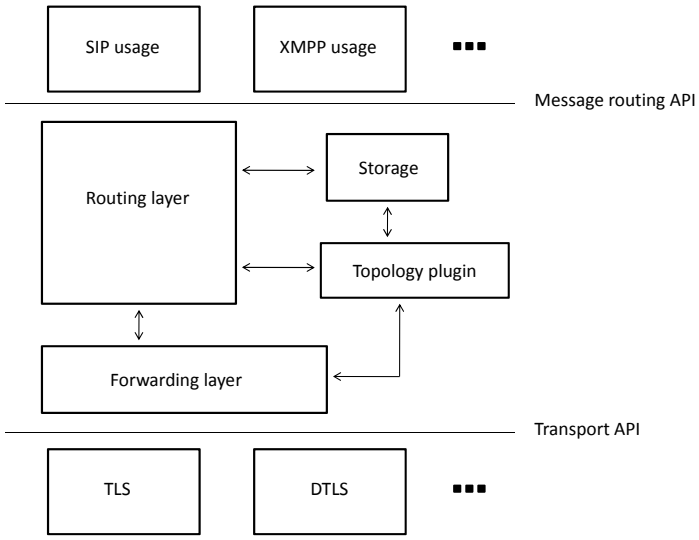


Figura 3.2: Architettura di rete del protocollo *RELOAD*

I componenti principali, ognuno con le sue caratteristiche e le sue funzionalità, sono comunque in stretto collegamento tra di loro.

- *Usage layer*; ogni applicazione definisce un *RELOAD usage* ovvero un set di tipi di dati e di comportamenti che spiegano le modalità d'uso dei servizi offerti. Tutti questi 'usi' comunicano con RELOAD attraverso una *Message Routing API* (*Application Programming Interface*) comune. Questo è il livello più alto ed il suo scopo è quindi quello di definire caratteristiche ed applicazioni specifiche dei generici servizi di overlay forniti; tramite questo livello vengono quindi definite specifiche applicazioni ed in particolare viene deciso dove mappare i dati, ovvero che forma può essere immagazzinata nell'overlay, dove salvare gli stessi, come fare queste operazioni in sicurezza ed, infine, come le applicazioni possono entrare in possesso ed utilizzare i dati.

- *Routing layer*; questo livello è responsabile dell'instradamento dei messaggi attraverso l'overlay. Inoltre gestisce gli stati di richiesta per i vari usi ed offre funzionalità di ricerca e di salvataggio. Comunica direttamente con il Topology Plugin, il responsabile dell'implementazione di topologie specifiche definite dall'algoritmo di overlay in uso. Infatti ogni peer è identificato dalla posizione

nell'overlay e dal suo ID di nodo e si ha che un componente che è client del livello di routing può eseguire due funzioni base:

  - inviare un messaggio ad un peer dato, specificato da un ID di nodo o da un ID di risorsa.
  - ricevere messaggi da altri peer, instradamento effettuato tramite un ID di nodo o un ID di risorsa di cui il peer in questione è responsabile.

  Tutti gli usi definiti sono client del livello di routing ed utilizzano i servizi di RELOAD tramite l'invio e la ricezione dei messaggi da altri peer. Il livello di routing fornisce, inoltre, un'interfaccia abbastanza generica per permettere al Topology Plugin di controllare l'overlay, le operazioni di ricerca ed i messaggi. Anche se ogni algoritmo di overlay venisse definito e funzionasse in modo differente dagli altri, un'invariante che dovrebbe essere sempre preservata è la correttezza della routing table ovvero una tabella degli altri peer che l'overlay mantiene ed usa per l'instradamento delle richieste. I componenti del livello di routing creano ed inviano richieste all'overlay per determinare il next hop, utilizzato poi per codificare ed inviare i messaggi; contemporaneamente a questa funzione l'algoritmo di overlay istituirà richieste di aggiornamenti periodici tramite i componenti, con lo scopo di mantenere corretta la routing table.

- *Storage*; questo componente è il responsabile del processing dei messaggi in funzione del salvataggio e del recupero dei dati. Comunica direttamente con il Topology Plugin e con il livello di routing in modo da riuscire ad inviare e ricevere messaggi e maneggiare repliche e migrazione di dati. È di fondamentale importanza poiché una delle principali funzioni di RELOAD è quella di permettere ai nodi di immagazzinare e recuperare dati salvati da se stessi o da altri nodi e quindi questa componente è la responsabile del processing dei dati salvati e dei messaggi di recupero.

- *Forwarding layer*; è il livello che fornisce il servizio di transito dei pacchetti tra i vari nodi ed è quindi il responsabile di invio e ricezione in accordo con i livelli di routing e di storage. Inoltre stabilisce e mantiene le connessioni di rete così come richiesto dal Topology Plugin, ed è responsabile del settaggio delle connessioni tra i peer attraverso NAT e firewall, utilizzando ICE.

- *Topology Plugin*; è il livello responsabile dell'implementazione dello specifico algoritmo di overlay in uso. Comunica direttamente con il Routing layer per inviare e ricevere messaggi di management, con la componente di storage per regolare la replicazione dei dati e con il Forwarding layer per controllare la spedizione dei messaggi hop per hop.
  Sebbene RELOAD sia stato realizzato per poter lavorare con un insieme ben definito di algoritmi di overlay, per facilitare l'implementazione di nuovi algoritmi alternativi il Topology Plugin permette l'interazione tra l'overlay, i codici

ed i protocolli centrali.
Il Topology Plugin, inoltre, è il responsabile del mantenimento della routing table che verrà poi consultata periodicamente dal livello di routing tramite messaggi d'instradamento. Inoltre, nel caso di connessioni assenti o danneggiate, il livello di transito notificherà la situazione al Topology Plugin, il quale provvederà all'aggiornamento delle routing table.
Le operazioni problematiche che possono provocare danneggiamenti alle tabelle di routing sono però l'ingresso e l'uscita di nodi in quanto possono comportare modifiche alla memorizzazione delle risorse; il Topology Plugin, allora, tiene traccia dell'elenco dei peer responsabili delle risorse ed in questo modo permetterà la migrazione delle risorse stesse in tempi limitati.
Infine questo livello è anche responsabile del mantenimento di dati ridondanti, al fine di proteggere le informazioni da attacchi o da peer compromessi o sovversivi.

3.2 Overlay management

La funzione base di RELOAD, come descritto in [4], è quella di costruire e gestire una generica rete di overlay. In questa rete i nodi devono essere in grado di entrare ed uscire a loro piacimento, devono poter creare connessioni con altri nodi ed essere in grado di instradare messaggi tramite l'overlay verso nodi con i quali non hanno una connessione diretta.

3.2.1 Identificazione e sicurezza

Ogni nodo nella rete RELOAD è caratterizzato da uno o più identificativi, contenuti in un certificato e globalmente univoci nella rete; questi identificativi sono utilizzati per indirizzare il nodo, determinare la sua posizione nell'overlay e determinare il set di risorse di cui è responsabile.
Il certificato ha vari scopi e molteplici fini:

- dà diritto all'utente di salvare dati a specifiche posizioni nell'overlay. Infatti ogni tipo di dato definisce le specifiche regole per determinare quali certificati possono accedere a quali risorse. Per esempio alcune tipologie di dati potrebbero permettere a tutti di scrivere in una data posizione, mentre altre tipologie potrebbero restringere la scrittura esclusivamente al proprietario di un particolare certificato. In questo modo, variando la posizione di salvataggio, si riescono ad irrobustire le operazioni di `storage` e `find`.

- dà diritto ad un utente di operare su di un nodo con ID contenuto nel suo certificato. In questo modo, nel momento in cui il nodo forma una connessione con un altro peer, esso può utilizzare questo certificato per raccogliere informazioni su

nodi circostanti. In più il nodo può firmare i messaggi, garantendo così integrità ed autenticità.

- dà diritto all'utente di utilizzare l'*user name* trovato nel certificato.

Se un utente ha più di una periferica, tipicamente esso avrà un certificato per ognuna di esse, in modo da permettere ad ognuna di operare come se fosse un peer indipendente.
RELOAD supporta due modelli di rilascio del certificato. Il primo modello è basato su di un processo di iscrizione centralizzato che permette l'allocazione di un nominativo univoco ed di un ID di nodo certificati con una coppia di chiavi pubblica e privata; tutti i peer in un particolare overlay utilizzano il server centralizzato come *trusted anchor* in modo da poter verificare ogni altro certificato.
In alcuni casi gli utenti vorrebbero poter settare una rete anche senza avere informazioni su attacchi di altri utenti; per gestire questi scenari RELOAD supporta l'uso di certificati auto-generati ed auto-firmati nei quali il nodo, oltre al certificato, genera il suo identificativo, digest della chiave pubblica, ed il suo username; in questo modo, però, la rete sarà sensibile a diversi attacchi (come l'attacco Sybil) ed è quindi uno scenario utilizzabile con sicurezza esclusivamente in ambienti in cui gli utenti sono mutualmente verificati.
In più RELOAD fornisce un sistema di *admission control* basato su chiavi condivise; in questo modello tutti i peer condividono una singola chiave che sarà poi utilizzata per autenticare le connessioni P2P.

3.2.2 I client

RELOAD definisce un singolo protocollo utilizzato poi sia per i peer che per i client. Questa implementazione semplificata permette alle periferiche di ricoprire entrambi i ruoli ed ai client di introdurre messaggi direttamente nell'overlay.
Con il termine *peer* si identifica un nodo nell'overlay che instrada messaggi verso altri nodi, oltre che verso quelli a cui è direttamente connesso, e che ha tipicamente responsabilità sul salvataggio dei dati. Il termine *client*, invece, è riferito a nodi che non hanno responsabilità sull'instradamento o il salvataggio dei dati.

Il meccanismo di gestione dei client in RELOAD permette a nodi che non partecipano alla rete come peer di utilizzare la stessa implementazione e gli stessi benefici derivati dal meccanismo di sicurezza, come se fossero peer. In questo modo i client possiedono ed utilizzano certificati per l'autorizzazione dell'utente a salvare dati in una specifica posizione dell'overlay; a questo scopo l'ID di nodo del certificato è utilizzato per identificare il client come membro dell'overlay e poter così autenticare i suoi messaggi.
È importante analizzare le modalità con le quali RELOAD supporta le funzionalità dei client, in termini di instradamento e di comportamento.

- *Client routing*; ci sono due opzioni di instradamento tramite le quali un client può essere localizzato nell'overlay:

  - stabilire una connessione con il peer responsabile dell'ID di nodo del client nell'overlay; in questo modo sarà questo peer a maneggiare lo step finale dell'instradamento verso il client considerato.
  - stabilire una connessione con un peer qualsiasi nell'overlay, basandosi per esempio sulla prossimità o sull'incapacità di instaurare una connessione con il peer responsabile; in questo caso il client deve fare affidamento su di una *destination list* per garantire la raggiungibilità. Nello scenario principale il client può iniziare la richiesta ed ogni nodo nell'overlay, che conosce la list verso la sua locazione corrente, può raggiungerla; nel caso, però, in cui esso non sia direttamente raggiungibile utilizzando esclusivamente il suo ID di nodo, la destination list necessita anche altre informazioni dipendenti dall'overlay e dall'implementazione.

- *Client behavior*; ci sono differenti motivazioni per le quali un nodo può comportarsi da client oltre che da peer e sono quindi possibili diversi scenari applicativi in cui il client deve modificare le sue capacità ed il suo comportamento.
  Un particolare nodo può essere forzato a comportarsi da client invece che da peer quando:

  - non ha un'appropriata connettività di rete, tipicamente a causa di un NAT molto restrittivo o di una connessione a banda molto bassa;
  - non ha risorse sufficienti in termini di potenza di processo, spazio per il salvataggio dei dati o potenza energetica;
  - l'algoritmo di overlay impone richieste specifiche per la selezione dei peer come ad esempio la partecipazione nell'overlay per determinare l'attendibilità di altri nodi o il controllo dei peer nella rete per ridurre il numero di hop nei percorsi di instradamento;

  Se un nodo non dovesse possedere le caratteristiche per essere un peer non è detto sia garantita la sua classificazione come client, in quanto ci sono anche delle funzionalità minime richieste per comportarsi da client:

  - deve implementare il meccanismo di gestione della connessione, per poter stabilire connessioni con altri peer;
  - deve implementare i meccanismi di salvataggio dei dati ed i metodi di recupero con funzionalità di client;
  - deve essere in grado di calcolare l'ID di risorsa utilizzato dall'overlay;
  - deve possedere i requisiti di sicurezza richiesti dall'overlay di cui fa parte;

  Un client, come già detto precedentemente, comunica attraverso lo stesso protocollo dei peer, conosce il modo con cui calcolare l'ID di risorsa e firma le sue richiste nella stessa modalità dei peer; a differenza dei peer, però, non richiede

una piena implementazione dell'algoritmo di overlay.
RELOAD, quindi, non supporta un differente protocollo per i client anche se a livello applicativo si avranno implementazioni differenziate per nodi peer e nodi client in modo da poter garantire i diversi comportamenti specificati e le differenti funzionalità richieste.

3.2.3 Il routing

Le capacità di instradamento di RELOAD devono essere definite in modo tale da rispettare alcune richieste fondamentali, in parte già introdotte precedentemente.

- *Attraversamento di NAT*; RELOAD deve supportare la stabilizzazione e l'utilizzo di connessioni tra nodi separati da uno o più NAT e deve permettere quindi la localizzazione di peer anche se situati in zone protette.

- *Promozione di client*; RELOAD deve supportare la richiesta da e verso client, anche se questi non partecipano attivamente all'instradamento. Inoltre deve garantire ad ogni client la possibilità di diventare un peer durante il tempo di vita della rete.

- *Gestione di topologie instabili*; è possibile che, durante l'evoluzione della rete, alcuni nodi abbiano informazioni inconsistenti riguardo la connettività del grafo di instradamento. Il protocollo RELOAD deve gestire questi casi in modo da assicurare che la risposta ad una richiesta finisca sempre al nodo che l'ha effettuata e non ad altri.

Per riuscire a rispettare questi vincoli il protocollo di routing di RELOAD si affida a due meccanismi base.

- *Via lists method*; l'header di invio usato da tutti i messaggi RELOAD contiene due liste fondamentali. Innanzitutto vi è una lista che mantiene le informazioni di come, hop per hop, i messaggi vengono instradati attraverso l'overlay; questa è chiamata *Via list*. In più vi è una lista contenente informazioni sulle capacità di routing per i messaggi di richiesta ed i percorsi di instradamento per i messaggi di risposta; questa è invece chiamata *Destination list*.
  Tramite queste due liste i nodi riescono a mantenere funzioni di fondamentale importanza per la buona riuscita dell'instradamento.

- *Route query method*; questo metodo permette invece ad un nodo di interrogare un peer per determinare il prossimo hop che utilizzerà poi per instradare un messaggio.

Il meccanismo base di instradamento utilizzato in RELOAD è il routing simmetrico ricorsivo, il quale presenta vantaggi in termini di prestazioni rispetto alle altre tecniche di routing.

Il *Symmetric Recursive Routing* richiede che un messaggio segua un percorso, attraverso l'overlay, verso la destinazione, allontanandosi sempre di più dal nodo d'origine; in questo modo ogni peer avvicinerà sempre più il messaggio alla sua destinazione. Inoltre il percorso di ritorno della risposta verso il mittente sarà lo stesso percorso determinato per l'instradamento del messaggio ma preso in senso opposto.

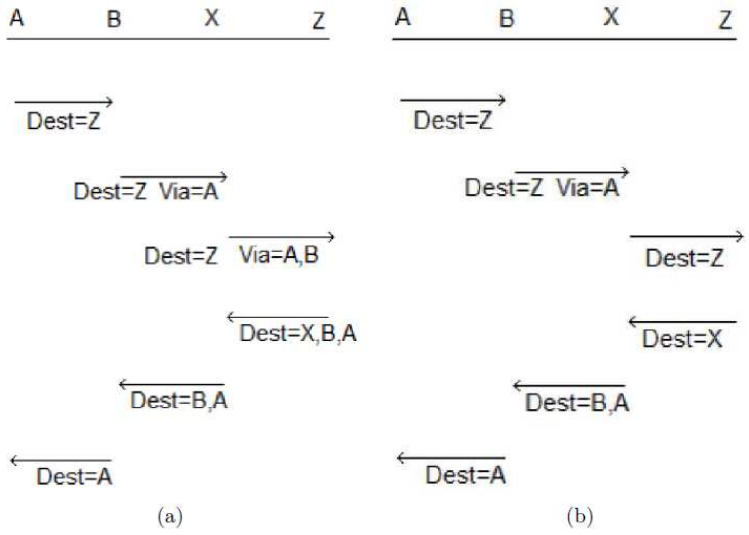


Figura 3.3: Esempio di *Symmetric Recursive Routing*

In Figura 3.3(a) è visualizzato un esempio di instradamento di un messaggio che ha come sorgente il nodo A e come destinazione il nodo Z e transita attraverso i nodi B ed X; in questo esempio il nodo A non è definito ne come client ne come peer, instrada un messaggio verso B, ed il messaggio di risposta sarà trattato poi nello stesso modo indipendentemente dalle caratteristiche di A.
L'esempio mostra quindi l'applicazione di una Via list tramite gli intermediari B ed X. Se B e X sono in uno stato di store, allora possono troncare la lista, salvare le informazioni interne e far tornare il messaggio di risposta lungo il percorso dal quale è stato ricevuto il messaggio. Queste opzioni richiedono un maggior numero di stati per i peer intermediari, ma permettono l'uso di un minor quantitativo di banda e riducono la richiesta per modificare i messaggi.
L'applicazione di questa modalità operativa o di una differente è una scelta individuale del peer e le varie tecniche utilizzabili sono comunque interoperabili.
La Figura 3.3(b), invece, utilizza ancora una Via list completa, ma X la tronca e salva lo stato internamente. Per scopi di debugging questo meccanismo necessita di un attributo di route log che salva informazioni sullo stato di ogni peer durante l'instradamento del messaggio.

RELOAD supporta inoltre una modalità instradamento detta *Iterative Routing*, dove i peer intermediari forniscono semplicemente una risposta indicando l'hop suc-

cessivo, senza però instradare direttamente il messaggio al nodo successivo.
Questa tecnica ha diversi vantaggi rispetto al Symmetric Recursive Routing in quanto il debug è più semplice, vi è un minor consumo di risorse sui peer intermediari e permette ai peer interrogati di individuare eventuali peer con comportamenti anomali e di conseguenza evitare di utilizzarli per l'instradamento. Il problema di questa tipologia di routing è, però, che in presenza di NAT risulta essere estremamente costosa in quanto deve stabilire ad ogni hop una nuova connessione, tramite l'utilizzo di ICE. È quindi impensabile pensare di utilizzarlo in reti P2P, dove la presenza di NAT e di firewall è molto frequente.
Di conseguenza l'algoritmo di routing scelto per il P2PSIP, che è anche l'algoritmo standard per RELOAD, è il Symmetric Recursive Routing, il quale presenta, oltre alle proprietà già analizzate, anche una buona stabilità, utile per non intaccare le prestazioni della rete.

3.3 L'implementazione P2PSIP

Il *SIP usage* di RELOAD, come descritto in [4,6,7], permette ad agenti SIP di fornire un servizio di telefonia P2P caratterizzato quindi dall'assenza di un proxy permanente o di un server di registrazione in quanto sarà RELOAD ad occuparsi delle ordinarie funzioni di registrazione e di connessione solitamente associate ai server.

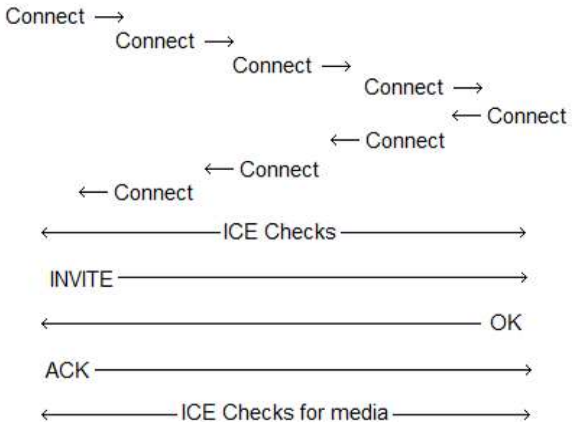


Figura 3.4: Esempio di connessione P2P tra due utenti *SIP*

La funzione base dell'applicazione SIP di tipo P2P, come è mostrato in Figura 3.4, è quella che permette al mittente, ad esempio Alice, di instaurare una connessione con una SIP URI, che potrebbe essere per esempio `sip:bob@dht.example.com`, in modo che il SIP UA di Alice e il SIP UA di Bob riescano a scambiarsi messaggi SIP. I vari passi che permettono l'instaurazione di questo canale comune sono:

1. Bob salva un'associazione tra il suo URI e il suo ID di nodo nell'overlay. Nel caso in esempio questa relazione sarà `sip:bob@dht.example.com` → 1234.

Al termine di questa operazione l'URI di Bob sarà associato ad un identificativo di nodo.

2. Alice decide di chiamare Bob. Allora richiede l'ID di nodo associato all'identificativo SIP di Bob. Nel caso in esame, in cui Alice è associata all'ID di nodo 5678, essa ricercherà `sip:bob@dht.example.com` e le verrà fornito l'ID di nodo 1234, ovvero quello che era stato associato al passo precedente.

3. Alice utilizzerà l'overlay per instradare un messaggio di connessione al peer di Bob; all'arrivo di questo messaggio Bob risponderà con un messaggio di accettazione della connessione e nel momento in cui questo messaggio raggiungerà il peer di Alice la connessione sarà settata.

È importante notare che l'unico ruolo di RELOAD, in tutte le operazioni di gestione di chiamate di tipo P2PSIP, è quello di settare la connessione diretta tra Alice e Bob: non appena il check ICE verrà completato e la connessione sarà stabilita, verrà utilizzato il SIP ordinario.
Ci sono quindi alcune operazioni del P2PSIP che in realtà non differiscono dal SIP standard; in particolare la stabilizzazione del media channel per la chiamata telefonica viene effettuata appunto mediante il meccanismo SIP standard, senza coinvolgere RELOAD e questo accade in quanto i media non devono passare attraverso l'overlay. Naturalmente, dopo lo scambio dei messaggi di SIP, i peer interessati interverranno dando il via al controllo della connessione per i media tramite ICE.
Oltre a permettere una connessione tra gli AOR (*Address of Record*) e i rispettivi ID di nodo, il SIP usage gestisce anche la connessione tra differenti AOR. Per esempio, se Bob volesse un inoltro temporaneo di tutte le sue telefonate verso Charlie, può salvare la relazione `sip:bob@dht.example.com` → `sip:charlie@dht.example.com`; in questo modo, nel momento in cui Alice cerca di chiamare Bob, riceverà l'indicazione relativa all'inoltro temporaneo ed allora per l'instaurazione della connessione cercherà l'ID di nodo legato all'AOR di Charlie.

Il SIP usage, quindi, permette all'overlay di RELOAD di essere utilizzato come rete SIP distribuita con funzioni di registrazione. Questo necessita però il verificarsi di tre operazioni fondamentali.

- *Registrazione di ogni AOR all'interno dell'overlay.*
  Nel SIP ordinario, un UA registra il suo AOR e la sua posizione tramite una funzione di Registrar. In RELOAD questa funzione è fornita dall'intero overlay in quanto, per registrare la sua posizione, un peer salva una struttura di registrazione SIP collegata al suo AOR; questo meccanismo si basa sulla tipologia dell'ID mentre i GRUU (*Globally Routable UA URI*), che verranno analizzati in seguito, sono trattati tramite un meccanismo separato.

Come già detto i peer possono salvare due tipologie di mapping SIP:

- da un AOR verso una destination list (come ad esempio un ID di nodo) con lo scopo di comunicare che, per l'instaurazione di una comunicazione con quel determinato utente, bisognerà contattare quel particolare nodo.
- da un AOR verso un altro AOR con lo scopo di inoltrare temporaneamente le chiamate verso un altro riferimento.

Sono definite due modalità per la registrazione di queste informazioni:

- se la registrazione è di tipo *SIP registration URI* allora il contenuto sarà una stringa 'opaca' contenente l'URI.
- se la registrazione è di tipo *SIP registration route* allora il contenuto sarà una stringa contenente le *contact preference* ed una destination list per il peer.

RELOAD supporta le registrazioni multiple per un singolo AOR e tutte le informazioni legate ad una particolare registrazione sono salvate in un dizionario, rintracciabili tramite l'ID di nodo che funge, quindi, da chiave.
Per prevenire il dirottamento delle richieste, le registrazioni sono soggette a regole di controllo d'accesso e si ha che, prima di permettere il salvataggio, il peer interessato deve verificare che:

- il certificato contenga un *username* corrispondente al SIP AOR.
- il certificato contenga un ID di nodo corrispondente alla chiave di ricerca nel dizionario collegato.

Queste regole, sempre riferendosi allo scenario di riferimento considerato precedentemente, permettono ad Alice di instradare chiamate verso Bob senza il suo permesso, ma non permettono ad Alice di instradare chiamate di Bob verso se stessa.

- *Ricerca di un dato AOR all'interno dell'overlay.*
Quando un utente RELOAD decide di chiamare un altro utente ed ha a disposizione un AOR non GRUU deve seguire una particolare procedura in quattro step.

  1. Controlla se l'AOR con cui vuole collegarsi coincide con un altro appartenente all'overlay di cui è membro. Se così non fosse l'AOR sarebbe da considerarsi esterno e di conseguenza dovrà essere bloccata la chiamata e iniziata una procedura SIP standard per cercare di diventare membro dell'overlay a cui appartiene l'AOR che si vuole chiamare.
  2. Esegue una *Fetch* per la registrazione SIP all'ID di risorsa corrispondente all'AOR. Questa operazione potrebbe non indicare nessuna chiave del dizionario, ed in tal caso si inizierà il recupero di tutti i valori memorizzati.

3. Se qualcuno dei risultati dell'operazione di Fetch è un AOR non GRUU, allora dovrà essere ripetuto il primo step per quell'AOR.
4. Nel momento in cui rimangono una sola GRUU ed una sola destination list allora il peer rimuoverà dalla lista i duplicati ed il GRUU e quindi instaurerà una connessione SIP con il peer appropriato. Inoltre, nel caso ci fossero anche AOR esterni, il peer dovrà seguire la procedura appropriata per riuscire a contattarli.

- *Realizzazione di una connessione diretta con un peer dato.*
  Una volta che il peer è riuscito a tradurre l'AOR in un set di destination list, utilizzerà l'overlay per l'instradamento di messaggi di connessione verso ognuno dei peer indicati; in questi messaggi il campo applicazione dovrà essere 5060 per indicare il protocollo SIP. In questo instradamento vi sarà autenticazione basata su di un certificato in quanto il peer destinatario sarà obbligato a presentare un certificato contenente le corrispondenze tra ID di nodo ed entry del terminale nella lista di instradamento. È comunque possibile che i peer abbiano già una connessione RELOAD tra di loro, ma questa non verrà utilizzata per i nuovi messaggi SIP; nel caso, invece, in cui esistesse già una connessione SIP, verrà riutilizzata anche per i nuovi messaggi.
  Nel momento in cui la connessione avviene con successo, il peer invierà messaggi SIP all'interno di essa, come in un normale SIP.

Come già citato nella registrazione dell'AOR all'interno dell'overlay, i GRUU vanno trattati separatamente; naturalmente bisogna comprenderne, prima di analizzarne il meccanismo di trattamento, il reale significato.
Molte applicazioni di SIP richiedono un UA per la costruzione e la distribuzione di un URI che possa essere poi utilizzato da tutti per l'instradamento di una chiamata verso una specifica istanza UA. Un URI che effettua l'instradamento verso una specifica istanza UA è chiamato GRUU.
I GRUU non richiedono immagazzinamento di dati nell'istanza di overlay, ma sono costruiti tramite l'incorporazione di una lista di destinazioni all'interno del parametro `gr`; questa lista è codificata in `base 64`, ovvero è realizzata con un alfabeto specifico legato a *RFC 4648*, con l'eccezione del simbolo $\sim$ utilizzato al posto di $=$. Un esempio di GRUU è

`sip:alice.dht@example.com; gr=MDEyMzQ1Njc4OTAxMjM0NTY3ODk~`

Nel caso in cui un peer necessiti l'instradamento di un GRUU verso un'altra rete P2P uguale a quella di appartenenza, utilizzerà semplicemente la destination list e si connetterà al peer corrispondente; siccome i GRUU contengono una destination list potrebbero avere lo stesso contenuto di quella salvata nel dizionario delle ricerche.
I GRUU anonimi, invece, sono realizzati nello stesso modo ma richiedono o che il server di iscrizione istituisca un differente ID di nodo per ogni GRUU anonimo richiesto, oppure che la destination list usata includa peer che comprimano la loro destination list per evitare che gli ID di nodo siano rivelati.

3.4 Considerazioni sulla sicurezza

RELOAD, come già introdotto in precedenza, fornisce un servizio generico di storage, propriamente disegnato per il P2PSIP.
In un'istanza di overlay, ogni utente dipenderà da un numero di peer senza relazioni ben definite a parte quella di essere tutti membri dello stesso overlay; il problema è che tutti questi nodi potrebbero essere con uguale probabilità corretti o maligni e di conseguenza nessun sistema di sicurezza potrà garantire perfetta protezione in un ambiente così incerto; lo scopo della sicurezza in RELOAD è quindi quello di garantire alcune invarianti anche nel caso in cui ci sia un alto numero di nodi maligni, ed il corretto funzionamento dell'overlay nel caso in cui ci sia solamente un modesto numero di *malicious node*.

Le specifiche del P2PSIP richiedono l'abilità di autenticare sia i peer che le risorse (e quindi gli utenti) senza necessitare la presenza di un'entità di autenticazione nel sistema; a questo scopo sono possibili due meccanismi: il primo, di default, è basato su certificati a chiave pubblica ed è disponibile per distribuzioni generiche, mentre il secondo, che è stato definito e viene supportato come 'alternativo', è basato su di un'ampia chiave simmetrica condivisa, ma è disponibile solo per distribuzioni limitate in cui la relazione tra i peer ammessi non risulta contraddittoria.

Le due funzioni base fornite dai nodi dell'overlay sono lo storage ed il routing; Vengono allora definite due grandi categorie di nodi: alcuni sono i responsabili del salvataggio dei dati dei peer e permettono ai peer stessi di individuare l'esatta locazione di alcuni set di dati richiesti; altri nodi sono invece responsabili dell'instradamento dei messaggi da e verso nodi di storage.
Gli overlay P2P, non solo quelli P2PSIP, sono soggetti ad attacchi da parte di nodi sovversivi che cercano di danneggiare l'instradamento, di corrompere o rimuovere la registrazione di utenti o di intervenire nella segnalazione. È quindi utile andare a realizzare degli algoritmi di sicurezza in modo da evitare, o in qualche modo limitare, l'influenza dei nodi maligni all'interno dell'overlay.

3.4.1 Certificate-based security

L'algoritmo di sicurezza basato sul certificato, algoritmo base per tutte le trasmissioni su RELOAD, è stato creato per garantire che instradamento e registrazione degli utenti avvengano senza problemi. Per riuscire a proteggere la segnalazione da attaccanti che si fingono peer onesti, il meccanismo da applicare è quello di garantire che tutti i messaggi siano ricevuti da membri autorizzati dell'overlay; per questo motivo RELOAD trasporta tutti i messaggi su canali sicuri in modo da garantire appunto integrità ed autenticità. In più, quando viene utilizzato il sistema di sicurezza basato su certificato, i messaggi e i dati vengono firmati digitalmente mediante la chiave privata del chiamante, in modo da rendere sicura la comunicazione end-to-end.

La rete richiede quindi, come detto, una soluzione per garantire la sicurezza dei dati e l'instradamento nell'overlay. Entrambe le misure di sicurezza, all'interno di questo algoritmo, sono basate sulla richiesta che ogni entità nel sistema, sia peer che utente, si autentichi crittograficamente utilizzando una coppia di chiavi asimmetriche, in modo da formare un certificato.
Quando un utente entra nell'overlay esso richiede, o gli viene assegnato automaticamente, un nome univoco, come ad esempio `sip:alice.dht@example.com`; questi nominativi sono scelti appositamente per essere facilmente comprensibili dall'utente umano. All'utente sono anche assegnati uno o più ID di nodo dall'autorità di iscrizione centrale. Nome e ID di nodo sono posti, insieme alla chiave pubblica dell'utente, all'interno di un certificato.
Ogni certificato permette ad un'entità di operare in due modalità:

- come utente, immagazzinando dati verso specifici ID di risorsa nell'overlay corrispondenti ad un nome utente.
- come peer, con l'ID inserito in una lista all'interno del certificato.

Si noti che, siccome solo gli utenti di questo overlay richiedono e necessitano di un certificato, non si richiede una PKI (*Public Key Infrastructure*); invece i certificati vengono firmati da un'autorità centrale di iscrizione che occupa il ruolo di autorità di certificazione per l'overlay e firma il certificato di ogni peer. Siccome ogni peer possiede il certificato dell'autorità centrale, che riceve al momento dell'iscrizione, esso può verificare i certificati delle altre entità nell'overlay senza ulteriori comunicazioni.

In caso di utilizzo di certificati auto-firmati, la sicurezza decresce sensibilmente in quanto gli attaccanti possono dare il via ad attacchi molto pericolosi (come l'attacco Sybil); in più gli attaccanti non possono verificare i nomi utente nei certificati, anche se possono comunque verificare l'ID di nodo perché sono crittograficamente verificabili.
Questo tipo di schema è appropriato solo per piccole reti, come uffici oppure overlay costruiti ad hoc per meeting e spesso, per aumentare la sicurezza, viene implementato un meccanismo di controllo dell'accesso a chiave segreta condivisa.
Siccome tutti i dati salvati sono firmati dal loro possessore, i peer che devono memorizzarli possono verificare l'autorizzazione riguardo a quell'ID di risorsa ed in più possono permettere ai consumatori di verificare provenienza ed integrità dei dati.

3.4.2 Shared-secret security

Questo schema di sicurezza con controllo dell'accesso basato su chiave segreta condivisa è un meccanismo, come già detto, non di default in RELOAD, ma disponibile per fornire ulteriore protezione alla rete. È basato su di una singola chiave condivisa da tutti i membri dell'overlay ed è appropriato per applicazioni in cui si ha un piccolo gruppo che cerca di formare una rete senza molte complessità.

In questo meccanismo di sicurezza tutti i peer condividono una singola parola simmetrica, utilizzata come chiave in un meccanismo TLS-PSK (*Transport Layer Security-Pre Shared Key Ciphersuites*), ovvero un set di protocolli crittografici che garantisce una comunicazione sicura basandosi su chiavi condivise predefinite. In questo modo un peer che non conosce la chiave non può formare una connessione TLS con nessun altro peer e non può entrare nell'overlay.

Un naturale approccio dello schema a chiave segreta condivisa è quello di utilizzare come chiavi delle password immesse dagli utenti, anche se così, soprattutto in caso di password dal significato banale o dalla struttura non abbastanza articolata, il sistema può essere soggetto all'attacco del dizionario; un'alternativa meno vulnerabile agli attacchi si basa sempre sulle password immesse dagli utenti, ma utilizzate poi come base per la creazione di una chiave condivisa in modo tale da rendere il TLS-PSK una 'scelta superiore' e quindi meno soggetta ad attacchi del dizionario.

3.4.3 Sicurezza nella fase di storage

Quando viene utilizzato un algoritmo di sicurezza basato sul certificato, ogni coppia `ID di risorsa/ID di tipo` fornita, viene limitata da alcuni set ridotti di certificati. Per scrivere i dati in uno slot il possessore deve fornire la prova di essere in possesso della chiave privata di uno di questi certificati; in più tutti i dati salvati vengono firmati dal certificato che autorizza l'immagazzinamento. Questo set di regole rende le richieste di autorizzazione e di integrità dei dati relativamente semplici.

Quando viene utilizzato un algoritmo di sicurezza a chiave segreta condivisa, ogni peer verifica tutti gli altri, garantendo così il rispetto delle credenzialità per poter effettuare l'ingresso nel sistema.

Quindi, mentre nel secondo meccanismo di sicurezza prima introdotto non ci sono molti problemi riguardo a verifiche ed a correttezza nei meccanismi di scambio di informazioni tra i vari peer, nel primo meccanismo bisogna valutare attentamente le varie fasi che compongono lo storage.

- *Autorizzazione*
  Se un client volesse salvare un particolare valore in uno slot, dovrebbe innanzitutto firmare il valore con la sua chiave privata e quindi mandare una `Store Request`, contenente sia il valore che la firma, verso il peer dedicato all'immagazzinamento. Nel momento in cui il peer riceve la richiesta dovrà determinare se il client è autorizzato a salvare dati in quel particolare slot; per fare questo eseguirà un algoritmo di tipo *Resource Name Construction*, utilizzando come parametri il tipo del valore da salvare e le informazioni del certificato dell'utente. Al termine di questa operazione verrà calcolato l'ID di risorsa tramite il Resource Name e si verificherà la corrispondenza con lo slot nel quale l'utente ha deciso di immagazzinare l'informazione. Se questa è verificata allora si può concludere che l'utente sarà autorizzato a scrivere in quel particolare slot, altrimenti l'autorizzazione verrà negata e la richiesta sarà immediatamente bloccata.

- *Distribuzione di quote dei dati*
  Essere un peer all'interno di una rete di overlay comporta la responsabilità di immagazzinare dati per una data regione dell'overlay. Tuttavia, se i client fossero autorizzati a salvare quantità di dati illimitate, questo potrebbe portare il peer responsabile ad avere oneri inaccettabili ed essere, di conseguenza, molto esposto a varie tipologie di attacchi.
  RELOAD risolve questo problema imponendo come caratteristica base di ogni usage una dimensione massima di dati immagazzinabili per ogni tipologia di dati. Tentativi di salvataggio di valori che eccedono questi limiti vengono respinti e nel caso in cui il peer risultasse inconsistente la rete è in grado di intervenire con un meccanismo tramite il quale viene cambiata la zona di responsabilità ed altri peer possono diventare responsabili dei dati troppo grandi; inoltre, siccome ogni slot è anche limitato ad un ristretto set di certificati, queste restrizioni sulla dimensione dei dati creeranno un meccanismo di distribuzione delle quote dei dati amministrato dal server di iscrizione centrale.
  La caratteristica di permettere a differenti tipi di dati di avere restrizioni proprie sulle dimensioni conferisce agli usage la flessibilità di poter limitare la forma dei propri dati senza dover imporre un limite generale.

- *Controllo della correttezza*
  Siccome ogni valore immagazzinato viene firmato, è banale per ogni peer verificarne l'integrità. Servono però altri interventi per prevenire attacchi di *Rollback*, che possono portare all'annullamento di tutti gli aggiornamenti effettuati nel corso della transazione. Questa tipologia di attacchi sullo storage viene evitata tramite l'uso di tempi di salvataggio e la creazione di *lifetime value* in ogni singola operazione di immagazzinamento. Un lifetime rappresenta l'ultimo istante di tempo di validità del dato e la sua presenza riesce a limitare, anche se non previene completamente, attacchi di Rollback.
  Per riuscire, invece, a prevenire l'attacco direttamente al momento della Store Request servirà un tempo di storage monotonicamente crescente; in questo modo gli storing peer potranno riuscire ad annullare tutte le richieste con tempo di storage minore o uguale a quello che stanno immagazzinando. In più, con questa modifica, se un nodo dovesse recuperare un valore e ne ricevesse uno con tempo di storage maggiore di quello relativo ad una ricerca precedente, riconoscerà un attacco di Rollback e permetterà quindi alla rete di isolare il nodo maligno.

I meccanismi descritti garantiscono un alto livello di sicurezza, ma rimangono possibili comunque alcuni attacchi.
Un esempio di attacco molto semplice, ma devastante per l'efficienza della rete, si basa sul fatto che rimane comunque possibile per un nodo rifiutare l'immagazzinamento di un valore. Inoltre un nodo potrebbe fingere di non conoscere valori precedentemente accettati e salvati.

Lo schema di autenticazione basato su certificato offre l'importante funzionalità di prevenire che un singolo peer sia in grado di realizzare dati indicando come autori altri peer; in questo modo un peer sovversivo non potrà immettere in rete dati forgiati da lui in quanto sarà sprovvisto dell'autenticazione per la registrazione, anche se potrà lo stesso danneggiare il sistema rifiutandosi di restituire dati dei quali è responsabile. Per cercare di aggirare questi attacchi la rete potrebbe immagazzinare più repliche dello stesso valore e poi, in fase di ricerca, andare a ricercarne più di una; l'affidabilità di questo meccanismo è statistica in quanto, in caso di bassa percentuale di peer compromessi, basterà solo un'esigua percentuale di repliche per rendere affidabile la rete, mentre in caso di molti nodi maligni collusi servirà un alto numero di repliche. Perciò questo procedimento risulta essere innanzitutto molto costoso ed inoltre un client non potrà sapere a priori quando comportarsi nella modalità standard e quando introdurre o ricercare le repliche e quindi bisognerebbe introdurre anche questo grado di conoscenza.
Inoltre, in caso di dati multivalore, come per esempio un array, il nodo che li immagazzina potrà fornire solamente alcuni sottogruppi di valori, polarizzando così le sue risposte. Questo può essere contrastato utilizzando valori singoli piuttosto che set, rendendo così il coordinamento tra salvataggi multipli più complesso; questo è comunque un costo ineliminabile e, per incrementare le prestazioni relazionate al costo computazionale, è necessaria una valutazione in fase di design scegliendo il trade-off desiderato.

3.4.4 Sicurezza nella fase di routing

Il sistema di sicurezza nella fase di storage garantisce, entro certi limiti, l'integrità dei dati immagazzinati. Il sistema di sicurezza nella fase di routing, invece, si concentra sul bloccaggio degli attaccanti in riferimento ad attacchi di tipo DOS (*Denial of Service*) sul sistema, che si basano su di un reinstradamento, naturalmente scelto dall'attaccante, dei messaggi e quindi non corretto per il sistema.
Ci sono alcune semplici osservazioni legate a questo attacco:

- è facile 'imbrogliare' il sistema facendogli credere che un attaccante sia un peer valido;
- nel caso in cui una larga percentuale di peer nell'overlay è corrotta, con molta probabilità sarà impossibile proteggere perfettamente il sistema da attacchi di tipo DOS;

I meccanismi di sicurezza nella fase di routing del protocollo RELOAD sono stati disegnati per contenere più che per eliminare attacchi, ed è quindi comunque possibile per un attaccante effettuare un'ampia varietà di attacchi. Lo schema di sicurezza basato sul certificato rende sicuro il *namespace*, ma se un peer nell'overlay fosse compromesso o l'attaccante ottenesse un certificato, il sistema potrebbe essere irrimediabilmente corrotto.

In generale, gli attacchi al routing di tipo DHT (*Distributed Hash table*) sono attuati dall'attaccante in modo da instradare il traffico attraverso i nodi da lui controllati; questi particolari attacchi, come analizzato in [8], sono particolarmente efficienti in sistemi P2PSIP, e devono quindi essere analizzati molto attentamente. Gli attacchi di questo tipo più dannosi sono l'attacco *Sybil* e l'attacco *Eclipse*. Come verrà trattato in dettaglio nei capitoli seguenti, in un *Sybil Attack* l'attaccante registra un ampio numero di nodi ed in questo modo può essere in grado di catturare un ampio volume di traffico; in un *Eclipse Attack*, invece, l'attaccante, tramite la manomissione delle tabelle di instradamento, può riuscire ad aumentare il volume di traffico controllato ovvero rispetto all'attacco Sybil, che rappresenta comunque un punto di partenza, riuscirà a catturare una percentuale di chiavi maggiore con un costo computazionale meno elevato. Questi attacchi sono limitati dall'utilizzo di un controllo di ammissione centralizzato e basato su certificati, anche se queste soluzioni allontanano il sistema dal paradigma P2P puro.

Controllo di Ammissione

L'ammissione, nel protocollo RELOAD, è controllata tramite la richiesta che ogni peer abbia un certificato contenente il suo peer ID; questa richiesta è rinforzata dall'utilizzo di un'autenticazione mutua basata su certificati ad ogni connessione. In questo modo, ogni volta che un peer si connette ad un altro, ognuna delle due parti potrà controllare automaticamente che l'altra abbia un certificato adatto.
Questi *peer ID* vengono assegnati casualmente dal server centrale di iscrizione e questo porta a due benefici:

- permette al server di iscrizione di limitare il numero di peer ID assegnati allo stesso utente fornendo protezione contro l'attacco Sybil grazie alla limitazione della capacità del singolo nodo;
- evita che l'attaccante possa scegliere peer ID specifici e questo riesce a limitare l'effetto dell'attacco Eclipse, anche se non a prevenirlo completamente, in quanto l'attaccante non potrà porsi in posizioni strategiche all'interno dell'overlay;

Identificazione ed autenticazione di peer

In generale, non appena un peer inizia un'attività che può portare alla modifica della routing table, deve stabilire la sua identità. Nel momento in cui il peer stabilisce una connessione diretta con un altro, dovrà autenticarsi tramite un processo mutuo basato su certificati. In questo modo si viene a formare un canale protetto sul quale viaggeranno poi tutte le informazioni tra i peer ed uno qualsiasi potrà verificare la provenienza e la correttezza dei valori senza necessitare di meccanismi di cifratura.
In alcuni casi, però, potrebbe essere necessario stabilire l'identità dei peer con i quali non si hanno connessioni dirette. Questa situazione, per esempio, potrebbe verificarsi

nel momento in cui un peer aggiorna il suo stato; in questo caso, gli altri peer potrebbero necessitare un aggiornamento della visualizzazione dell'overlay ma dovrebbero verificare che il messaggio di aggiornamento provenga dal peer e non da un attaccante; per questo motivo tutti i messaggi di routing dell'overlay vengono firmati dal peer che li ha generati.

Protezione della segnalazione

Lo scopo della protezione della segnalazione è quello di non permettere ad un attaccante di scoprire il mittente ed il destinatario della segnalazione stessa, e possibilmente anche il contenuto.
Un attaccante potrebbe però essere in grado di osservare le attività di un gruppo di peer ed inoltre, siccome i messaggi potrebbero essere instradati utilizzando esclusivamente le informazioni del campo header, il corpo del messaggio RELOAD potrebbe essere cifrato durante la trasmissione.
Nascono allora due linee di difesa per controbattere questo problema; queste strategie non sono esclusive ma possono essere integrate per ottenere un miglior livello di sicurezza.

- Utilizzo di TLS (*Transport Layer Security*) o DTLS (*Datagram TLS*) per ogni link di comunicazione tra peer; questo garantisce protezione contro attaccanti che non sono membri dell'overlay, ovvero attaccanti esterni.

- Utilizzo l'algoritmo di sicurezza *certificate-based*, ed in questo modo ogni messaggio può essere firmato; questo permetterà la protezione dei messaggi dalla modifica da parte di peer avversari, anche se fossero situati sul percorso di routing del messaggio stesso.

Tramite questi meccanismi di sicurezza, quindi, il protocollo RELOAD può riuscire a gestire varie problematiche legate principalmente all'implementazione della versione P2P di SIP.
Come detto già in precedenza, però, lo studio di questo protocollo e di conseguenza di tutto il P2PSIP usage, è ancora oggi ad uno stadio iniziale e quindi serviranno alcuni anni di sviluppo per riuscire ad osservare miglioramenti generali sul funzionamento e sullo sviluppo di tecniche di sicurezza.

Capitolo 4

Il protocollo Chord

Il protocollo Chord [9, 10] è un protocollo scalabile per la ricerca in sistemi P2P dinamici con frequenti arrivi ed uscite di nodi; esso supporta fondamentalmente un'operazione: data una chiave la assegna ad un nodo e permette poi la ricerca ottimizzata del nodo responsabile di una particolare chiave; inoltre, a seconda dell'applicazione che utilizza Chord, il nodo potrebbe essere responsabile della memorizzazione di un valore associato alla chiave.
In questo capitolo verrà analizzato nel dettaglio il protocollo Chord in quanto è considerato attualmente lo standard per la gestione del P2PSIP ed il suo funzionamento può essere visto come riferimento per uno scenario tipico d'utilizzo del protocollo RELOAD; è stato quindi interessante riuscire ad analizzare un protocollo semplificato rispetto alla struttura di *usage* classica che, come detto nel capitolo precedente, non è ancora stata standardizzata, in modo tale da riuscire ad effettuare varie misurazioni ricavando risultati coerenti anche per il protocollo RELOAD e quindi per la struttura di gestione del P2PSIP che diventerà lo standard nei prossimi anni.

4.1 Concetti preliminari

Il protocollo Chord semplifica il design di reti P2P generiche e di sistemi basati su di esse tramite l'indirizzamento di alcuni problemi complessi.

- *Bilanciamento del carico*; Chord fornisce una computazione veloce e distribuita di una hashing function per l'assegnamento delle chiavi ai nodi responsabili; con alta probabilità questa funzione di hash bilancia il carico, ovvero tutti i nodi ricevono all'incirca lo stesso numero di chiavi, ed inoltre, nel momento in cui l'N-esimo nodo entra (o esce) dalla rete, viene mossa esclusivamente una frazione $O(1/N)$ di chiavi in una diversa locazione, ovvero il minimo necessario per mantenere il bilanciamento del carico.

- *Decentralizzazione*; Chord è un protocollo pienamente distribuito ovvero nessun nodo risulta avere più importanza di un altro. Questa caratteristica assicura robustezza e rende Chord appropriato ad applicazioni *P2P loosely organized*.
- *Scalabilità*; il costo della ricerca in Chord cresce come il logaritmo del numero dei nodi e questo rende disponibili ed efficienti ricerche anche all'interno di reti molto grandi.
- *Disponibilità*; Chord aggiorna automaticamente le sue tabelle interne per riflettere in tempo reale e senza particolari problemi ingressi o uscite di nodi, garantendo così in ogni momento la ricerca veloce del nodo responsabile di una chiave.
- *Nominativi flessibili*; Chord non impone vincoli sulla struttura delle chiavi di ricerca, ovvero lo spazio delle chiavi è uniforme. Questa caratteristica garantisce all'applicazione alta flessibilità sul mappaggio dei nomi nelle chiavi.

4.2 Definizione del protocollo

Chord specifica i meccanismi legati alla ricerca delle locazioni delle chiavi e alle modalità con cui i nodi entrano o escono dal sistema.
Queste operazioni si basano su di un paradigma fondamentale, noto come *Consistent Hashing*, che viene utilizzato per assegnare le chiavi ai nodi; il Consistent Hashing tende a bilanciare il carico, dato che ogni nodo controlla circa lo stesso numero di chiavi, e di conseguenza ne coinvolge un piccolo quantitativo durante le operazioni di ingresso o di uscita dal sistema. È per questo motivo che ogni nodo mantiene informazioni su circa $\log_2 N$ altri nodi e per la ricerca richiede un massimo di $O(\log_2 N)$ messaggi.
Nei sistemi precedenti si assumeva che ogni nodo conoscesse informazioni relative alla maggior parte degli altri nodi del sistema, rendendo così impraticabile la scalabilità nel caso di reti di dimensioni elevate. Nel caso di Chord, invece, ogni nodo gestisce informazioni di routing riguardanti solamente un piccolo numero di altri nodi ed in questo modo tutte le operazioni di gestione risultano semplificate.
Per comprendere a fondo il funzionamento del protocollo bisogna analizzare separatamente le varie strutture che costituiscono Chord sia in caso di rete statica che nel caso dinamico di ingresso ed uscita di nodi.

4.2.1 Consistent hashing

La funzione di Consistent Hashing, come già introdotto precedentemente, assegna ad ogni nodo e ad ogni chiave un identificativo di m bit usando solitamente come funzione di hash *SHA-1* (*Secure Hash Algorithm*). L'identificativo di un nodo viene scelto calcolando l'hash dell'indirizzo IP del nodo stesso, mentre quello di una chiave è

prodotto calcolando l'hash del valore della chiave. La lunghezza m dell'identificativo deve essere abbastanza elevata in modo tale da garantire che la probabilità che due nodi o due chiavi abbiano lo stesso hash sia molto bassa; questa lunghezza, naturalmente, dipende dalla rete in esame anche se solitamente gli hash calcolati con SHA-1 prevedono m pari a 160 bit.
Il Consistent Hashing, inoltre, ordina gli ID in una struttura nota come *identifier circle* modulo 2^m e la chiave k verrà assegnata al primo nodo il cui identificativo è uguale o maggiore di k nello spazio degli identificativi. Questo nodo viene chiamato *successor node* della chiave k e, rappresentando gli ID come un cerchio di numeri da 0 a $2^m - 1$, il *successor(k)* sarà il primo nodo in senso orario partendo da k.

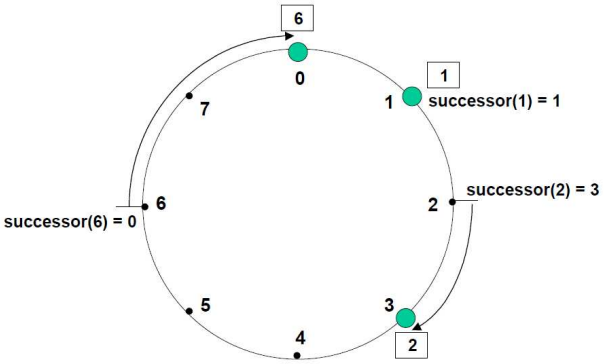


Figura 4.1: Esempio di *identifier circle* formato da 3 nodi

La Figura 4.1 mostra un identifier circle semplificato con $m = 3$ e dove la rete rappresentata ha solamente 3 nodi (0, 1 e 3). Il successore dell'identificativo 1 sarà il nodo 1 e quindi la chiave 1 dovrebbe essere allocata al nodo 1, mentre il successore dell'identificativo 2 sarà il nodo 3 ed è proprio li che andrà allocata la chiave 2; similmente la chiave 6 verrà allocata al nodo 0.

Il Consistent Hashing è stato progettato appositamente per permettere ai nodi di entrare ed uscire dalla rete con il minimo disordine; per mantenere l'assegnazione effettuata da questo meccanismo, nel momento in cui un nuovo nodo entrerà nella rete alcune chiavi assegnate al suo successore dovranno entrare in suo possesso mentre se un nodo dovesse lasciare la rete tutte le chiavi in suo possesso dovranno essere trasferite al suo successore.

4.2.2 Locazione scalabile delle chiavi

Per implementare il Consistent Hashing in un ambiente distribuito è necessario un numero molto esiguo di informazioni di routing: ogni nodo ha la necessità di conoscere esclusivamente il suo successore nell'identifier circle in modo da far scorrere lungo

l'anello le richieste per un dato ID.
Una porzione del protocollo Chord mantiene questi puntatori ai successori, in modo da garantire la correttezza di ogni operazione di ricerca, ma questo schema di risoluzione è inefficiente in quanto potrebbe richiedere l'attraversamento di tutti i nodi per trovare l'assegnazione adeguata.

Per accelerare questo processo, Chord mantiene allora un'informazione di routing addizionale, non essenziale per la correttezza, che viene comunque garantita nel momento in cui viene conservata correttamente l'informazione sul successore di ogni nodo.
Questa informazione addizionale prende il nome di *finger table* ed ogni nodo n mantiene questa tabella di routing caratterizzata al più da m elementi con l'i-esima finger contenente l'identità del primo nodo s che succede n di almeno 2^{i-1} posizioni nell'identifier circle, ovvero $s = successor(n + 2^{i-1})$ (tutto modulo 2^m). Questo nodo s verrà chiamato i-esimo finger, o i-esima entry della finger table, del nodo n.
Una qualsiasi entry della finger table includerà il *Chord identifier*, l'indirizzo IP ed il numero di porta del nodo rilevante; naturalmente la prima finger di un generico nodo n sarà l'immediato successore di n all'interno dell'identifier circle.
Quindi gli elementi della generica entry k della finger table del nodo n sono:

- `finger(k).start`, pari a $(n + 2^{k-1}) \mod 2^m$
- `finger(k).interval`, pari a $[finger(k).start; finger(k+1).start)$, intervallo della particolare entry.
- `finger(k).node`, primo nodo con ID maggiore o uguale a *finger(k).start*.

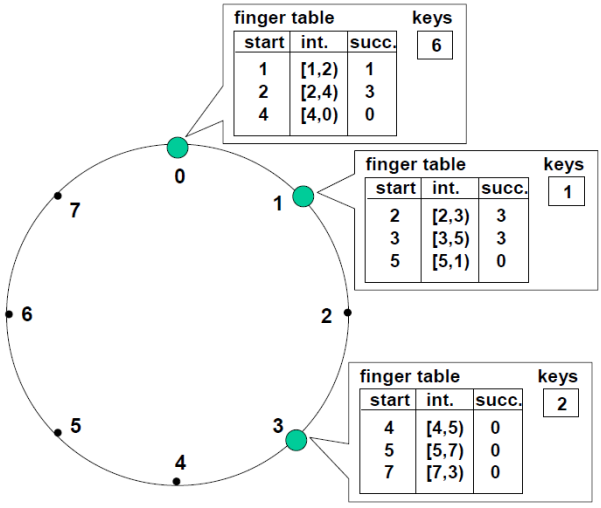


Figura 4.2: Finger table e locazione delle chiavi in un *identifier circle* formato da 3 nodi

Nell'esempio mostrato in Figura 4.2, la finger table del nodo 1 punta ai nodi successori degli identificativi

$$\begin{aligned}(n + 2^0) \mod 2^3 &= 2\\ (n + 2^1) \mod 2^3 &= 3\\ (n + 2^2) \mod 2^3 &= 5\end{aligned}$$

che sono rispettivamente il nodo 3, ancora il nodo 3 ed il nodo 0.
Questo schema ha due importanti caratteristiche:

- ogni nodo mantiene informazioni riguardo un piccolo numero di altri nodi, e conosce di più in riferimento ai nodi più prossimi che lo seguono nell'identifier circle, piuttosto che su nodi molto lontani.
- la finger table di un nodo generalmente non contiene abbastanza informazioni per determinare il successore di un'arbitraria chiave k. Infatti, considerando ancora l'esempio in Figura 4.2, il nodo 3 non conosce il successore di 1, e il successore del nodo 1 non appare nella *finger table* del nodo 3.

Ma cosa succede nel caso in cui un nodo n non conosce il successore di una chiave k? Se n può trovare il nodo il cui ID è più vicino del proprio a k, allora tale nodo conoscerà di più riguardo all'identifier circle nella regione di k di quanto sappia n. Quindi n cercherà nella sua finger table un nodo p il cui ID precede k più da vicino, ed inoltrerà a p la richiesta. Ripetendo questo processo iterativamente n acquisirà informazioni riguardo ai nodi vicini a k.
Per questa operazione viene definita la funzione `find-predecessor` la quale ha il compito di fornire l'immediato nodo predecessore dell'identificativo desiderato il cui successore è anche il successore dell'identificativo. Quando un nodo eseguirà la ricerca di un predecessore l'obiettivo della funzione sarà quello di muoversi, attraverso le entry delle finger table, lungo l'anello Chord, avvicinandosi sempre di più all'ID cercato. Se il nodo dovesse contattare un nodo n' con identificativo posizionato tra n' ed il suo successore allora la funzione avrà raggiunto il suo scopo e quindi restituirà al nodo chiamante n'; altrimenti interverrà la ricorsività della funzione in quanto il nodo n' interrogherà il nodo nella sua finger table che precede l'ID più da vicino ed esso ripeterà le stesse operazioni. L'algoritmo, quindi, farà sempre progressi verso il predecessore dell'ID, rendendo l'operazione di *lookup*, che in [11] è analizzata nel dettaglio, anche se in relazione al P2PSIP, estremamente efficiente.

Nell'esempio in Figura 4.2, supponendo che il nodo 3 voglia trovare il successore dell'identificativo 1, bisognerà effettuare una ricerca proprio partendo dal nodo 3 come riferimento. Siccome 1 appartiene all'intervallo circolare [7,3), allora apparterà a `3.finger[3].interval`, dove `finger[i].interval` non è altro che l'intervallo `[finger[i].start,finger[i+1].start)`; il nodo 3 quindi verificherà il terzo elemento nella sua tabella, che è 0. Dato che 0 precede 1, 3 chiederà al nodo 0 di trovare il successore di 1; siccome 0, analizzando la sua finger table, scoprirà che il successore

dell'ID 1 è il nodo 1 stesso, restituirà come risposta al nodo 3 il nodo 1. In questo modo il nodo 3 capirà che il responsabile dell'identificativo 1 è il nodo 1.

Quindi la topologia della rete è una struttura ad anello in cui ogni nodo prenderà una posizione determinata dall'hashing del suo indirizzo costituito da indirizzo IP più porta; ogni nodo Chord ha allora quattro elementi fondamentali:

- *Successor*, il nodo che segue il nodo dato nell'identifier circle.
- *Predecessor*, il nodo che precede il nodo dato nell'identifier circle.
- *Finger table*, una tabella di massimo m elementi con lo scopo di aumentare l'efficienza dell'algoritmo.
- *Key vector*, insieme delle chiavi di cui il nodo è responsabile.

4.2.3 Ingresso ed uscita di nodi

In una rete dinamica, i nodi possono entrare e lasciare il sistema in ogni momento e senza nessuna indicazione preventiva. Il principale cambiamento a livello di funzionamento di Chord nel caso di implementazione di queste operazioni è il preservare l'abilità di allocamento di ogni chiave nella rete; Chord deve quindi essere un protocollo robusto ad ingressi ed uscite.
Per ottenere questo obiettivo, Chord avrà bisogno di preservare due invarianti, ovvero ha bisogno di mantenere sempre corrette due informazioni:

- ogni nodo dovrà mantenere correttamente il suo successore.
- per ogni chiave k dovrà sempre esistere *successor(k)* e dovrà sempre essere il corretto responsabile.

In più, per rendere la ricerca veloce, è anche desiderabile che le finger table siano tenute corrette.

Per semplificare i meccanismi di ingresso ed uscita, come descritto in [9, 10], ogni nodo dovrà mantenere un puntatore al suo predecessore; il puntatore al predecessore di un nodo conterrà il Chord identifier e l'indirizzo IP dell'immediato predecessore di tale nodo e potrà essere utilizzato per percorrere in senso antiorario l'identifier circle.

Un nuovo nodo n, nel momento in cui fa un'operazione di *Join*, dovrà quindi imparare diverse informazioni relative alla rete e per farlo si avvarrà di un nodo casuale n' già presente nell'overlay. Il nodo n, quindi, userà il nodo n' per inizializzare il suo stato ed entrare quindi a far parte della rete Chord esistente. Per realizzare questa fondamentale operazione e per preservare le due invarianti prima introdotte, il nodo entrante dovrà eseguire tre fasi.

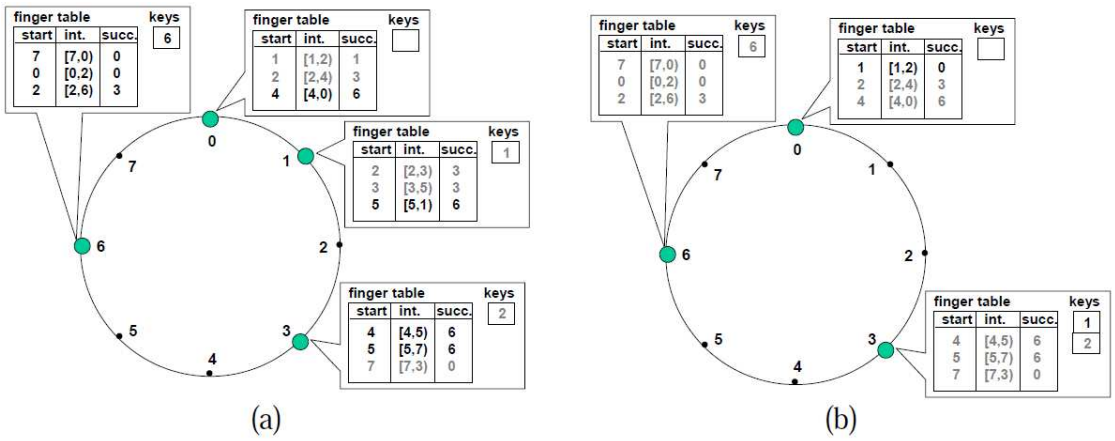


Figura 4.3: (a) Finger table e locazione delle chiavi dopo l'ingresso del nodo 6. (b) Finger table e locazione delle chavi dopo l'uscita del nodo 3. Le entry cambiate sono visualizzate in nero, quelle invariate in grigio

1. Inizializzare il predecessore e le entry della finger table. A questo scopo il nodo n chiederà ad n' di fare una ricerca; da una prima e sommaria analisi risulterebbe necessario eseguire un'operazione di `find node` per ognuno degli m elementi della finger table, per un tempo di esecuzione totale pari a $O(m \log_2 N)$. In realtà durante queste operazioni il nodo n controlla se sono valide sia l'i-esima entry che l'i+1-esima entry della finger table; questo avviene nel caso in cui `finger[i].interval` non contiene alcun nodo e quindi `finger[i].node`>=`finger[i+1].start`. Con questa modifica il tempo di esecuzione totale si ridurrà fino a $O(\log_2 N)$.

2. Aggiornare le finger table ed i predecessori dei nodi esistenti in modo da riflettere l'aggiunta del nuovo nodo n. Nell'esempio in Figura 4.3 il nodo 6 diventa la terza entry per i nodi 0 e 1 e la prima e seconda entry per il nodo 3.
   La funzione di update della finger table ha la funzione di aggiornare le tabelle esistenti secondo un principio molto semplice ovvero il nodo n diventerà l'i-esima finger del nodo p se e solo se p precede n di almeno 2^{i-1} e l'i-esima finger del nodo p succede n; quindi il primo nodo p che unirà queste due condizioni sarà l'immediato predecessore di $n - 2^{i-1}$. In questo modo, per un dato nodo n, l'algoritmo si muoverà in senso anti-orario sull'identifier circle partendo dall'i-esima entry della finger table di n, fino a che non incontrerà un nodo il cui i-esimo finger precede n.

3. Notificare a livello software in modo da poter trasferire lo stato associato alle chiavi di cui il nodo n diventa responsabile. Questa è l'ultima operazione che deve essere eseguita al momento di una Join e consiste nello spostamento verso il nuovo nodo di tutte le chiavi di cui risulta essere il nuovo successore. Cosa

comporta questa operazione dipenderà dal software di più alto livello di Chord ma tipicamente potrebbe riguardare lo spostamento dei dati associati ad ogni chiave che cambia proprietà verso il nuovo nodo. Naturalmente il nodo n potrà diventare successore esclusivamente delle chiavi di cui era precedentemente responsabile il nodo che ora è diventato suo predecessore e così, per portare a termine questa delicata operazione, dovrà contattare esclusivamente un nodo per poter trasferire la responsabilità per tutte le chiavi rilevanti.

Per quanto riguarda l'operazione di *Leave* la procedura sarà del tutto simile a quella di ingresso e le fasi saranno quindi le stesse descritte in precedenza. In realtà la Leave di un nodo potrebbe anche essere trattata in modo totalmente differente, come se il nodo fosse ancora presente in rete ma ignorasse tutte le richieste reindirizzandole al suo successore; questo meccanismo semplifica il processo ma porta ad una riduzione di efficienza dell'algoritmo. Quindi l'operazione di Leave di un nodo, come si vede in Figura 4.3, consisterà in:

1. garantire la correttezza della rete andando a cancellare il nodo appena uscito dal suo successore e dal suo predecessore.

2. nel momento in cui si è collegati al suo successore, trasferire ad esso tutte le chiavi di cui il nodo uscente era responsabile.

3. aggiornare le finger table nello stesso modo visto in precedenza.

Per rendere più semplici queste operazioni di Leave il protocollo Chord dovrà garantire che ogni nodo mantenga non uno, ma una lista di successori all'interno dell'identifier circle; grazie a questo espediente, nel momento in cui un nodo lascia il sistema il suo predecessore sceglierà come successore il secondo elemento della sua successor list, senza dover andare a chiedere informazioni al nodo uscente o senza dover andare a cercare il nodo in questione.

4.2.4 Stabilization

L'algoritmo di Join discusso in precedenza porta inevitabilmente ad una conservazione che può essere definita 'aggressiva' delle finger table dei nodi durante l'evoluzione della rete. Si ha però un problema nel caso di ingressi multipli in quanto rischiano di venire a meno la correttezza e le performance del protocollo.
Viene allora utilizzato un protocollo di stabilizzazione con lo scopo di mantenere aggiornati i puntatori ai successori dei nodi, funzionalità che garantisce la correttezza del *lookup*; successivamente questi puntatori verranno utilizzati per aggiornare le finger table e rendere di nuovo il procedimento, oltre che corretto, anche veloce. Tramite questa modifica il tempo di aggiornamento del sistema a seguito di una Join o di una Leave risulterà essere $O(\log_2{}^2 N)$.

Il protocollo di stabilizzazione, analizzato in [12] in riferimento a reti P2PSIP, si inserirà all'interno dell'algoritmo di funzionamento di Chord immediatamente dopo

l'ingresso di un nodo: quando un nodo n entrerà nel sistema si collegherà immediatamente ad un nodo n' chiedendogli il valore del suo successore in modo che potrà posizionarsi dopo un lasso di tempo molto basso all'interno dell'identifier circle. Subito dopo questa operazione si scatenerà la funzione di *Stabilize* che verrà poi ripetuta da ogni nodo della rete periodicamente; con questa funzione il nodo n in questione chiederà al suo successore informazioni riguardo al suo predecessore p e deciderà poi se quel nodo può essere considerato anche il suo successore. In questo modo si riuscirà a gestire un eventuale ingresso non riscontrato precedentemente, in quanto accaduto molto recentemente. In più la funzione notificherà al successore di n l'effettiva presenza del nodo n stesso, dandogli così la possibilità di modificare il suo predecessore (questa modifica verrà fatta se il successore di n non conosce nessun predecessore a lui più vicino di n).

Il problema si potrebbe avere nel momento in cui l'ingresso di nodi dovesse interessare più regioni dell'anello Chord; in questo caso una ricerca che avviene prima della fine della stabilizzazione potrà provocare comportamenti discordanti.

- Nel caso generico, tutte le entry delle finger table interessate alla ricerca saranno ragionevolmente corrette, ed allora la ricerca troverà il *successor(k)* in un massimo di $O(\log_2 N)$ step.

- Un secondo caso è invece caratterizzato dai puntatori ai successori corretti, ma da tabelle inaccurate. In questo caso la ricerca sarà corretta anche se potrebbe avvenire in un lasso di tempo maggiore.

- Infine si ha lo scenario in cui i nodi della regione affetta dagli ingressi multipli hanno puntatori ai successori non corretti o le chiavi non sono ancora migrate verso i nuovi nodi entrati; in questo caso la ricerca fallirà. Il software di livello alto di Chord dovrà gestire questo fallimento ed allora, nel momento in cui verrà informato che il dato desiderato non è stato trovato, ci sarà una ripetizione della ricerca dopo una pausa che sarà tanto più breve tanto più il protocollo di stabilizzazione sarà veloce nel correggere i puntatori ai successori.

4.3 Implementazione di Chord

Come già accennato in precedenza, si è deciso di utilizzare l'overlay Chord come protocollo di riferimento per la fase simulativa, che verrà poi analizzata nel dettaglio nei capitoli seguenti. Per riuscire ad implementarla si è utilizzata una particolare realizzazione del protocollo appena introdotto, sviluppata tramite la piattaforma *OMNeT++/OverSim*.
È quindi necessario, ai fini della comprensione del lavoro svolto, analizzare il simulatore utilizzato ed in particolare la sua implementazione del protocollo Chord.

4.3.1 Il simulatore OverSim

OverSim è un simulatore flessibile per overlay di reti basato su OMNeT++ [13], un sistema di simulazione ad eventi discreti.
OverSim è stato disegnato, come descritto in [14, 15], per adempiere ad un numero di richieste che, come suggerito dagli autori, erano state parzialmente trascurate dalle piattaforme di simulazione già esistenti; presenta quindi tutte le possibili qualità che un simulatore di rete dovrebbe avere, ovvero:

- *scalability*; il simulatore è in grado di effettuare simulazioni con overlay di dimensioni estese in un lasso di tempo adeguato.
- *flessibility*; il simulatore è in grado di semplificare l'utilizzo sia di overlay strutturati che non strutturati e di permettere un utilizzo molto flessibile, tramite dei file di configurazione, contenenti parametri base, facilmente modificabili dall'utilizzatore.
- *underlying network modeling*; il simulatore presenta la possibilità di intercambiare il modello di rete di underlay in modo da riuscire ad adattare il più possibile la simulazione allo scenario reale.
- *reuse of simulation code*; il simulatore è realizzato in modo tale da permettere l'intercambiabilità di parti del codice con altre piattaforme e questo garantisce ampi margini di crescita e di sviluppo.
- *statistics*; il simulatore è in grado di collezionare dati statistici come il traffico di rete inviato, ricevuto o instradato da ogni nodo, come i pacchetti consegnati con o senza successo e come il numero di hop effettuati per portare a termine una consegna.
- *documentation*; il simulatore è supportato da un'ampia documentazione riguardante i vari spezzoni di codice e le varie funzionalità.
- *interactive visualizer*; per riuscire a semplificare l'utilizzo e l'efficacia informativa del simulatore, i dati di output sono visualizzabili tramite un'interfaccia grafica customizzabile.

OverSim include alcuni protocolli P2P strutturati e non strutturati come *Chord*, *Kademlia* e *Gia* e le loro implementazioni possono essere utilizzate sia per simulazioni che anche per un approccio relativo a reti reali, tramite l'estensione *INET-OverSim* che, per lo sviluppo di questo lavoro di tesi, non è stata utilizzata. In più OverSim prevede un insieme di *common function*, come un generico meccanismo di lookup per reti P2P strutturate ed un'interfaccia RPC (*Remote Procedure Call*), per facilitare l'implementazione di protocolli addizionali e renderli tra loro comparabili.
Il simulatore prevede anche, come già accennato, diversi modelli di rete di underlay

intercambiabili per consentire la simulazione di reti complesse ed eterogenee e per attuare semplificazioni in caso di simulazioni in larga scala.

In conclusione si può quindi dire che la struttura del simulatore OverSim è molto semplice ed è funzionale in quanto è un'architettura modulare, composta da vari spezzoni di codice, alcuni anche riutilizzabili per più scopi, ma comunque legati all'implementazione del singolo overlay.

4.3.2 L'implementazione di Chord in OverSim

Tra i vari modelli realizzati in OverSim, vi è una serie di moduli atti all'implementazione del protocollo Chord, in accordo con le specifiche descritte in [9, 10].
Accanto a questi vi sono moduli generici per la gestione sia delle strutture di overlay che della fase applicativa che, nel caso di Chord, consiste nella ricerca del responsabile di una particolare chiave. Per comprendere meglio i risultati ottenuti durante le varie fasi simulative, che saranno illustrati in seguito, bisogna allora analizzare nel dettaglio sia i moduli per l'implementazione di Chord che i moduli applicativi utilizzati nella fase di ricerca.

L'overlay Chord

L'implementazione del protocollo Chord in OverSim è caratterizzata dalla suddivisione del codice in tre moduli base che prendono il nome di `Chord`, `ChordSuccessorList` e `ChordFingerTable`; l'unione delle varie procedure contenute in questi moduli consente al simulatore di implementare tutte le funzionalità del protocollo descritte precedentemente.
Questi tre moduli presentano caratteristiche e funzionalità differenti ma sono tutti ugualmente importanti per il funzionamento dell'applicazione ed inoltre necessitano dell'appoggio di alcuni modelli generici di OverSim, in comune tra i diversi overlay implementati.

Il modulo `Chord` è il modulo principale dell'overlay e consente l'organizzazione dei vari terminali nella classica struttura ad anello, come mostrato in Figura 4.4. Le procedure implementate consentono l'ingresso e l'uscita dei nodi ed il mantenimento, anche dopo le operazioni di Join e di Leave, della struttura ad anello tramite l'utilizzo di funzioni puramente grafiche per una rappresentazione, come mostrato sempre in Figura 4.4, visivamente molto chiara.
Sovraintende inoltre anche alle varie operazioni di assegnamento delle chiavi, cioè di ricerca dei nodi responsabili delle chiavi stesse. È da considerarsi il modulo principale in quanto è in stretto contatto sia con gli altri moduli specifici di Chord che con le procedure generiche di OverSim ed infatti si ha che:

- per quanto riguarda i rapporti con i moduli generici di OverSim, `Chord` è legato principalmente ad operazioni di modifica dell'overlay e di ricerca. Per quanto riguarda la modifica dell'overlay si ha che, in caso di Join o di Leave, i moduli

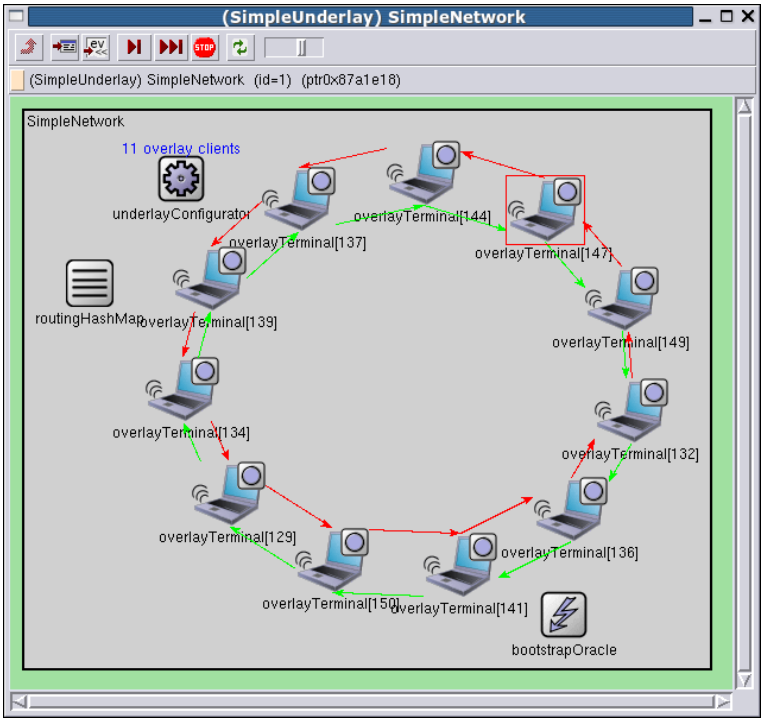


Figura 4.4: Esempio di visualizzazione grafica di un overlay Chord in *OMNeT++/OverSim*

generici, che gestiscono queste particolari operazioni, comunicheranno a `Chord` le informazioni relative al cambiamento ed esso provvederà alla modifica dell'anello tramite le interazioni con i moduli atti alla gestione della lista di successori e della finger table. Per quanto riguarda invece le operazioni di assegnamento delle chiavi al corretto nodo responsabile, `Chord` è in stretto contatto con i moduli legati all'applicazione in modo da fornire, tramite operazioni di ricerca nell'anello, il desiderato responsabile di una certa chiave.
Riguardo l'interazione tra `Chord` ed i moduli generici, è necessario fornire una descrizione di due procedure utili per la gestione dell'algoritmo che in [9,10] non vengono analizzate nel dettaglio; per comprendere le funzionalità, allora, ci si è basati sulla loro implementazione all'interno del simulatore.

- `find node`; è una procedura non specifica di Chord, ma che può essere utilizzata anche in caso di overlay simulativo differente. Consiste nella ricerca del nodo successore di un identificativo dato e si basa sulla creazione di una lista di nodi presenti all'interno dell'overlay in ogni istante di tempo; alla chiamata della funzione, allora, tramite il confronto tra l'identificativo fornito e quelli contenuti nella lista, tutti derivati dall'indirizzo IP tramite la funzione di hash SHA-1, viene fornito l'ID del nodo più vicino.
- `fix finger`; è una procedura necessaria nella fase di *stabilize* del protocol-

lo Chord ed il suo compito è quello di eseguire periodicamente un *refresh* delle entry delle finger table. La sua importanza è molto elevata in quanto, soprattutto in caso di frequenti Join e Leave, le informazioni necessarie per il routing potrebbero cambiare ed in particolare potrebbe variare la corrispondenza tra `finger[i].start` e `finger[i].node`. Questa procedura, allora, controlla se, per ogni entry della finger table, la corrispondenza `.start`/`.node` è corretta ed in caso contrario provvede alla ricerca del giusto successor, tramite la funzione `find successor`, in modo da avere in ogni istante di tempo informazioni coerenti.

- per quanto riguarda i rapporti con `ChordFingerTable` e `ChordSuccessorList` si ha che la creazione di un elenco di vicini o quella di un set di entry della finger table non può avvenire casualmente, ma deve essere regolata da una serie di funzioni dedicate allo scopo. `Chord` implementa procedure con la funzionalità di andare ad imporre una modifica alle due liste fondamentali per mantenere le corrette relazioni di successorship tra i nodi e le corrette finger in ogni tabella e facilitare così le successive operazioni di ricerca.

Il modulo `ChordSuccessorList` è il modulo responsabile per il mantenimento della corretta struttura ad anello dell'overlay Chord. Esso permette la gestione, per ogni nodo presente nell'overlay, di una lista di successori; in questo modo si riesce a mantenere sempre corretta la principale invariante di Chord: in ogni istante ogni nodo deve sempre avere il corretto predecessore ed il corretto successore in modo tale da riuscire a costruire l'identifier circle.
Le procedure implementate da questo modulo permettono quindi la creazione, la modifica e l'analisi di questa lista in modo tale da permettere al modello `Chord` di sapere, in ogni istante di tempo e dopo una generica operazione di Join o di Leave, il corretto successore di ogni nodo considerato. Quella dei successori è una lista proprio per riuscire a gestire al meglio le operazioni di ingresso e di uscita di nodi; in particolare, in caso Leave, il predecessore del nodo considerato, semplicemente eliminando la prima entry dalla successor list e senza ulteriori operazioni, riuscirà ad avere il corretto riferimento al suo nuovo successore, velocizzando così le operazioni.

Il modulo `ChordFingerTable`, infine, è il responsabile nella gestione delle finger table di tutti i nodi dell'overlay. Il mantenimento di questa tabella è un'operazione fondamentale poiché, come detto precedentemente, avere entry corrette permette ricerche più veloci ed efficaci. `ChordFingerTable` implementa quindi procedure per la gestione della finger table ed, in particolare, per l'aggiunta di una nuova entry, per la cancellazione di una entry relativa ad un nodo uscito dall'overlay e per la modifica di una entry nel caso in cui l'ingresso o l'uscita di un nodo impongano particolari variazioni.

Il singolo elemento della tabella risulta essere formato principalmente dal campo noto come `finger(k).first` che corrisponde al campo `.node` descritto in [9,10]. Per quanto riguarda il campo `.start`, esso viene semplicemente calcolato dalla relazione

$$(n + 2^{k-1}) \mod 2^m$$

già descritta in precedenza, mentre il campo `.interval` può essere facilmente dedotto dalla conoscenza di due entry consecutive.
Naturalmente è fondamentale che le relazioni tra questo modulo ed il modulo `Chord` siano corrette in quanto l'aggiunta, la modifica e la cancellazione di una entry della finger sono possibili esclusivamente tramite comandi imposti dal modulo principale che risulta quindi essere il controllore, oltre che della successor list, anche della finger table.

La fase di ricerca del nodo responsabile

L'operazione di ricerca del `successor(k)` è da considerarsi una fase successiva a quella della creazione dell'anello precedentemente analizzata.
Per riuscire a gestire le due operazioni separatamente ed in modo semplice, OMNeT++/OverSim definisce tre particolari fasi simulative che intervengono in modo differente nella gestione dell'overlay e sulla fase di ricerca. La sequenza di queste fasi e la loro durata, così come anche altri parametri base per la simulazione, possono essere facilmente impostati e modificati dall'utilizzatore all'interno di un file di configurazione, `omnetpp.ini`. La Figura 4.5 mostra un esempio di parametri di configurazione per una rete statica; alcuni parametri hanno un significato facilmente intuibile, come per esempio `**.overlayType`, che naturalmente indica Chord (ma che potrebbe indicare un qualsiasi altro overlay disponibile in OverSim), e `*.underlayConfigurator.churnGeneratorTypes` che, indicando `noChurn`, rappresenta proprio lo scenario di rete statica, senza Join e Leave dei nodi. Gli altri parametri assumono invece un significato specifico all'interno delle tre fasi che compongono la simulazione.

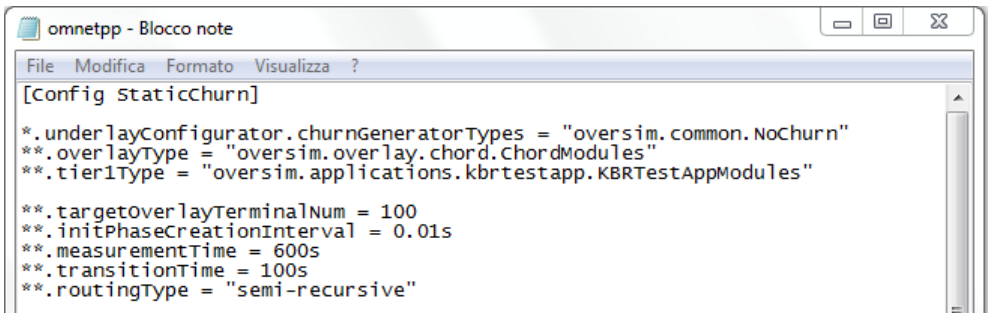


Figura 4.5: Esempio di *configuration file* per una rete statica in *OMNeT++/OverSim*

La prima fase, nota come *init phase*, è la fase dedicata alla creazione dell'anello. Basandosi sui parametri

`**.targetOverlayTerminalNum` e `**.initPhaseCreationInterval`

contenuti nel file di configurazione, l'utente può decidere la grandezza dell'overlay ed il tempo medio di creazione dell'anello. Si ha infatti che, sempre in riferimento al file di configurazione in Figura 4.5, in questa prima fase verranno generati 100 nodi con intervallo medio di generazione tra uno e l'altro pari a $0{,}01s$; al termine della init phase, che durerà circa $100 \cdot 0{,}01$ secondi, si avrà solitamente un numero di nodi nell'overlay pari a quelli richiesti, collegati tra di loro in modo coerente all'interno della struttura ad anello.

Per evitare, però, possibili errori dovuti al fatto che i riferimenti temporali all'interno del file di configurazione sono tempi medi, viene impostata anche una *transition phase*, ovvero una fase in cui il simulatore non fa nulla se non continuare le operazioni di creazione dell'anello iniziate nella init phase. Questa fase, la cui durata viene decisa dall'utente tramite il parametro `**.transitionTime`, è necessaria per avere la certezza che all'inizio della fase finale di misurazione siano terminate le varie operazioni di creazione dell'overlay; questo, naturalmente, è vero se si sta considerando una rete statica, mentre andranno fatte ulteriori considerazioni in caso di rete dinamica.

La fase finale è nota invece come *measurement phase* ed è la fase di misurazione, ovvero la fase principalmente relativa alle operazioni di ricerca dei responsabili. In questa fase, la cui durata è impostata tramite il parametro `**.measurementTime`, possono essere utilizzati differenti moduli applicativi, tramite l'utilizzo del campo `**.tier1Type` nel quale verrà indicato il nome corretto del livello applicativo che si appoggerà poi all'overlay selezionato sottostante.
Nel caso di Chord l'applicazione standard utilizzata per la fase di ricerca, come mostrato nell'esempio in Figura 4.5, è nota come `KBRTestApp`, la quale consiste nella generazione di un valore di key casuale ed una successiva operazione di ricerca del nodo responsabile, partendo da un nodo generico dell'overlay e muovendosi secondo la struttura di Chord lungo l'anello. Quindi, nel momento in cui l'applicazione scatena l'i-esima ricerca, verrà creato un particolare messaggio applicativo che, passando di nodo in nodo seguendo la struttura della finger table raggiungerà proprio il nodo responsabile della chiave.

Nel caso invece in cui si volesse simulare un overlay dinamico, caso di interesse in quanto le reti P2P reali sono caratterizzate da continue operazioni di Join e Leave, bisognerebbe impostare correttamente alcuni parametri, sempre contenuti nel file di configurazione, legati direttamente alla dinamicità della rete.
La differenza tra rete statica e dinamica è semplicemente legata al fatto che, nel secondo caso, alla fine della fase di transizione e nel momento in cui inizia la fase di misurazione, non si avrà un overlay fisso ma la sua struttura varierà; infatti, mentre il programma applicativo effettuerà le varie operazioni di ricerca e collezionerà i dati statistici, si scateneranno continue Join e Leave e quindi la struttura dell'anello e le

varie entry delle finger table verranno continuamente modificate in modo da essere comunque coerenti alla struttura dinamica dell'overlay. Naturalmente, avendo fissato il numero medio di nodi nell'overlay, la dimensione della rete manterrà sempre un valore circa pari a quello impostato, ma l'elenco dei nodi considerati varierà continuamente nel tempo.
I parametri da aggiungere o da modificare nel file di configurazione mostrato precedentemente, per la creazione di un overlay Chord dinamico, sono quelli relativi alla gestione delle operazioni di ingresso ed uscita dei nodi:

- `*.underlayConfigurator.churnGeneratorTypes` = `oversim.common.LifetimeChurn`; tramite questo parametro la tipologia di *Churn* non sarà più statica ma sarà di tipo *Lifetime*. Questa tipologia di churn, che non è l'unica ma è stata quella poi applicata alle simulazioni effettuate, rappresenta un modello di rete nel quale ogni nodo, dopo un tempo medio di vita fissato, effettua una Leave ed esce dal sistema; questa operazione, siccome il simulatore deve mantenere overlay di dimensioni circa pari a quella fissata in ogni istante di tempo, scatenerà una successiva operazione di Join.

- `**.lifetimeMean` = $500s$; tramite questo parametro viene imposto il tempo medio di vita di ogni nodo della rete, in questo caso pari a $500s$, necessario per regolare il rate di Join e di Leave.

Capitolo 5

Attacchi sulle reti P2P

Le reti P2PSIP, ed in generale tutte le reti basate su di un paradigma P2P, sono soggette ad un'ampia varietà di attacchi atti a minare il loro funzionamento. Questa particolare fragilità è proprio legata alla struttura P2P che rende si la rete più scalabile e più flessibile, ma aumenta le tipologie di attacchi e le loro percentuali di successo.
In questo capitolo verranno analizzati alcuni attacchi, considerando lo scenario applicativo, l'effetto che hanno sul sistema ed i meccanismi base per la difesa.

A causa dell'ampia varietà di protocolli P2P esistenti, ci si riferirà, come in [16], ad un modello astratto di overlay P2P, strutturato per rendere la discussione indipendente dal particolare protocollo; nei capitoli seguenti, invece, ci si riferirà con precisione all'implementazione dell'attacco Eclipse sul protocollo Chord, che rappresenta lo scenario simulativo considerato.

5.1 Modello di riferimento

Nel modello astratto considerato, ad ogni nodo partecipante è assegnato un identificatore random uniforme (*nodeID*) estratto da uno spazio degli ID. Agli oggetti delle specifiche applicazioni, invece, sono assegnati identificatori univoci, chiamati *key*, estratti dallo stesso set di valori. Ogni chiave è mappata dall'overlay ad un unico nodo centrale chiamato *key root* ed il protocollo instrada messaggi con una data chiave verso il root associato.
Per instradare i messaggi in modo efficiente, ogni nodo mantiene una routing table, contenente l'ID e l'indirizzo IP associato degli altri nodi, ed in più un nodo potrebbe mantenere un neighbor set, ovvero una lista dei suoi vicini.
Per riuscire a tollerare i guasti, infine, ogni oggetto applicativo viene immagazzinato in più di un nodo nell'overlay ed a questo scopo esiste una *replica function* per riuscire a mappare la chiave di un oggetto in più repliche, ognuna delle quali seguirà poi un percorso di instradamento differente.

Il sistema lavora su un set di N nodi, che costituiscono l'overlay, tra i quali si assume una frazione f di nodi maligni; ognuno di questi nodi non opera necessariamente in modo solitario, ma potrebbero esserci set di nodi che collaborano per arrecare maggior danno al sistema; allora, chiamata c la dimensione di ogni partizione di nodi maligni disgiunta, si ha che il caso peggiore sarà quello in cui $c = f$ in quanto tutti i nodi colluderanno per causare danni al sistema.

I nodi del sistema comunicano su connessioni internet e si possono distinguere due tipologie di livelli di comunicazione:

- *Network level*, dove i nodi comunicano direttamente senza l'instradamento attraverso l'overlay. Per i messaggi inviati a questo livello di comunicazione il sistema descritto in [16] impone l'uso di tecniche crittografiche per prevenire che nodi maligni riescano ad effettuare modifiche o anche solo ad osservare le informazioni. Questo però non esclude la possibilità che i nodi maligni entrino in possesso di messaggi inviati a questo livello, in quanto le connessioni da e verso nodi maligni a livello rete sono completamente disponibili.

- *Overlay level*, dove i messaggi sono instradati attraverso l'overlay; è il livello di comunicazione attraverso il quale i nodi maligni possono cercare di effettuare attacchi per riuscire a ridurre la sicurezza nella rete.

5.2 Secure routing

La sicurezza nella fase di routing attraverso l'overlay è una condizione di importanza vitale per tutti i sistemi P2P ed è quindi fondamentale riuscire a definire delle primitive di routing sicure che possano essere combinate con le esistenti tecniche per rendere le applicazioni più sicure.
Con overlay composti da nodi maligni, i messaggi che circolano in rete possono essere corrotti, eliminati o diventare di proprietà proprio di nodi maligni. Per risolvere questi problemi viene definita una *Secure Routing Primitive* che garantisce che, nel momento in cui un nodo non corrotto invia un messaggio ad una chiave k, il messaggio ricercherà con alta probabilità tutti i membri non corrotti nel set delle repliche R_k, definito come il set di nodi che contiene, per ogni membro del set delle chiavi di replica associate a k, il nodo principale responsabile della chiave replicata.

Il Secure Routing assicura che:

1. il messaggio venga consegnato alla sua reale destinazione, nonostante ci siano nodi che possano corromperlo, eliminarlo o cambiare il suo instradamento;

2. il messaggio venga consegnato a tutte le radici delle repliche della chiave legittimata, nonostante ci siano nodi che cercano di impersonare radici non legittime;

L'implementazione di queste primitive richiede la soluzione di tre problemi.

- *Assegnamento sicuro di ID ai nodi*, che assicura che un attaccante non possa scegliere il valore dell'ID di nodo. Senza questo controllo l'attaccante potrebbe scegliere ID in posizioni strategiche in modo da intercettare tutto il traffico da e verso altri nodi.
- *Mantenimento sicuro delle routing table*, che assicura che la frazione di nodi corrotti all'interno di una qualsiasi routing table non superi, in media, la frazione di nodi maligni nell'intero overlay. Senza questo controllo un attaccante potrebbe impossessarsi di gran parte delle routing table anche con una piccola percentuale di nodi maligni.
- *Invio sicuro dei messaggi*, che assicura che almeno una copia del messaggio inviato ad una chiave riesca a raggiungere, con alta probabilità, ogni corretta radice.

Verranno ora analizzate, come riportato in [16–18], alcune soluzioni a questi fondamentali problemi di sicurezza nelle reti P2P.

5.2.1 Assegnamento sicuro di ID ai nodi

Le performance e la sicurezza all'interno delle reti P2P dipendono dalla fondamentale assunzione che ci sia una distribuzione casuale uniforme degli ID di nodo che non possa essere controllata da un eventuale attaccante.
Se gli attaccanti fossero in grado di scegliere i propri ID di nodo potrebbero, senza necessitare un ampio numero di nodi controllati, compromettere l'integrità di un overlay P2P. Per esempio un attaccante potrebbe essere in grado di partizionare un overlay Chord tramite il controllo di due neighbor set completi e disgiunti.

Inoltre, attaccanti che possono scegliere ID di nodo, saranno in grado di controllare l'accesso a particolari oggetti e, nel caso in cui un attaccante scelga un ID di nodo più vicino possibile a tutte le chiavi replica per un particolare oggetto, esso potrà controllare tutti i percorsi delle repliche con il risultato di riuscire a cancellare completamente, corrompere o negare l'accesso all'oggetto in questione.
Anche nel caso in cui gli attaccanti non possano scegliere i propri ID di nodo, però sono in grado di ottenere facilmente un largo numero di ID di nodo legittimi, essi possono essere in grado di effettuare attacchi ugualmente pericolosi.

Questo è il caso del *Sybil Attack* [17, 19, 20]; in questo particolare tipo di attacco, un attaccante si basa sulla facilità di generare nuove identità per creare un largo numero di entità che collaborano allo scopo di controllare il maggior numero possibile di messaggi che circolano nell'overlay. In questo modo, all'aumentare degli ID di nodo che un particolare attaccante può riuscire ad ottenere, aumenterà la pericolosità dell'attacco ed aumenteranno i danni che possono essere provocati sul sistema.
È proprio per questo motivo che i sistemi P2P necessitano di un controllo sulla creazione degli ID di nodo, non solo per quanto riguarda la possibilità di posizionare i

nodi in punti strategici dell'overlay ma anche riguardo alla possibilità di creare molti ID di nodo legittimi: l'assegnamento sicuro è di fondamentale importanza.

Soluzioni

Una soluzione all'assegnamento sicuro di ID di nodo è quella di delegare il problema ad un'autorità centrale verificata. In questo modo si utilizzerà un set di CAs (*Trusted Certification Authorities*) per assegnare ID di nodo e firmare *nodeID certificates* per permettere il collegamento di ogni ID con una chiave pubblica legata al possessore ed al suo indirizzo IP. Le CAs assicurano che gli ID di nodo vengano scelti in modo casuale dallo spazio degli identificatori non permettendo così ai nodi di forgiarsi il proprio identificativo. In più questi certificati garantiscono all'overlay un'infrastruttura a chiave pubblica, adatta a stabilire canali cifrati ed autenticati tra i nodi.
In questa struttura di rete, le CAs non verranno coinvolte nelle regolari operazioni dell'overlay ma potranno essere offline per ridurre il rischio di esposizione delle chiavi di firma del certificato. I nodi con ID valido e certificato potranno in questo modo entrare nell'overlay, instradare messaggi ed uscire senza l'intervento delle CAs.
L'inclusione dell'indirizzo IP all'interno del certificato, inoltre, conferisce più sicurezza al sistema; infatti, senza il riferimento all'indirizzo IP, un attaccante potrebbe riuscire ad ottenere più ID di nodo legittimati ed in questo modo potrebbe aumentare senza limiti la frazione di nodi maligni controllati all'interno dell'overlay con la possibilità, inoltre, di riuscire ad ottenere ID di nodo equamente distribuiti in tutto lo spazio degli identificatori. Considerando invece, nelle operazioni di certificazione, anche l'indirizzo IP, diventerà più difficile per un attaccante ottenere più ID di nodo e soprattutto risulterà essere molto difficoltoso il riuscire a muovere ID lungo la rete.

La soluzione tramite *CAs* è però vulnerabile a più tipologie di attacchi. Inoltre, per reti P2P particolarmente grandi, risulta essere una tecnica ingombrante.
In più, per non perdere il paradigma P2P, si vorrebbero riuscire a creare overlay sicuri senza richiedere l'intervento di autorità centralizzate per il controllo dell'identità. Sfortunatamente un assegnamento degli ID di nodo completamente decentralizzato è impossibile da applicare, in quanto ha limiti molto forti sulla sicurezza e quindi, ancora oggi, non si hanno soluzioni che possano completamente prevenire la possibilità che un attaccante acquisisca una larga collezione di ID di nodo.
Ci sono però alcune tecniche che permettono di moderare il rate con cui un attaccante può acquisire ID di nodo ed una possibile soluzione è quella che richiede ad un potenziale nodo di risolvere un *Crypto Puzzle*.
Il metodo del Crypto Puzzle [21] si basa sul semplice concetto di definire appunto un puzzle come un crittogramma destinato ad essere rotto; per risolvere questo puzzle bisogna crittoanalizzare il crittogramma. Fatto questo si riescono ad apprendere le informazioni risultanti relative al plaintext, ovvero l'informazione che era stata cifrata. Il puzzle, però, è destinato ad essere risolto mentre idealmente un crittogramma non dovrebbe poter essere crittoanalizzato. Risolvendo il Crypto Puzzle, allora, il nodo otterrà il diritto ad avere un ID, ed in questo modo si riusciranno ad evitare la

maggior parte degli attacchi di tipo DOS. Il problema di questo metodo è che, per ottenere puzzle di difficoltà sufficiente per riuscire ad autenticare l'acquisizione degli ID, bisognerebbe affibbiare ad ogni operazione di risoluzione un costo computazionale elevato che renderà molto lento l'algoritmo.

Un meccanismo più efficace per effettuare un assegnamento sicuro degli identificativi di nodo, algoritmo utilizzato nel simulatore *OMNeT++/OverSim*, consiste nel legare l'indirizzo IP del nodo con il suo ID per evitare attacchi che sfruttino la network locality; questa corrispondenza si traduce nel fatto che l'ID del nodo non è altro che il risultato di una funzione di hash, solitamente *SHA-1*, dell'indirizzo IP. In questo modo qualsiasi nodo, per farsi riconoscere dagli altri, dovrà presentare la coppia *(indirizzo IP,node ID)*.
All'interno di questo meccanismo sarà possibile invalidare periodicamente ID di nodo tramite l'utilizzo di entità di fiducia; questa modifica renderà difficoltoso per un attaccante riuscire ad accumulare e riutilizzare ID di nodo, anche se il mantenere nodi legittimati comporterà un maggior costo computazionale.

5.2.2 Mantenimento sicuro delle routing table

Il meccanismo di mantenimento delle routing table e delle neighbor list è utilizzato per la creazione delle tabelle nella fase di Join di un nodo e per il mantenimento durante il tempo di permanenza del nodo stesso nell'overlay.
Idealmente ogni routing table ed ogni neighbor set dovrebbero avere una frazione media di nodi maligni pari ad f, frazione di nodi maligni nell'overlay. In realtà gli attaccanti possono riuscire ad aumentare la frazione di *bad entries*, tramite una modifica delle funzioni di routing updates, in modo tale da riuscire a ridurre la probabilità di un routing di successo.

Una prima tipologia di attacco è quella in cui l'attaccante mente sulla prossimità per riuscire ad aumentare la frazione di nodi maligni nelle routing table; in questo modo, quando un nodo corretto p invia un messaggio di sonda per la stima delle distanze verso alcuni nodi, l'attaccante può intercettarlo ed in questo modo far figurare un nodo maligno come nodo più vicino possibile al nodo p. Nel caso in cui l'attaccante controlli un'ampia frazione di nodi all'interno dell'overlay, potrebbe cercare di porsi vicino alla maggior parte dei nodi corretti e controllare così un'ampia frazione dei messaggi scambiati attraverso l'overlay.

L'attacco Eclipse

Una seconda tipologia di attacco, che verrà analizzata nel dettaglio in quanto rappresenta l'argomento centrale di questo lavoro di tesi, e che verrà poi sviluppata nei capitoli seguenti, non si basa sulla manipolazione delle informazioni di prossimità. Si basa invece sul fatto che è difficile capire quando un routing update è legittimo; i nodi riceveranno informazioni di update al loro ingresso nell'overlay e quando altri nodi entrano ed escono dalla rete. Allora, periodicamente, ogni nodo, tramite informazioni

ricevute da altri nodi presenti in rete, potrà aggiornare la sua routing table eliminando i buchi e riducendo gli hop.

In questi sistemi un attaccante può modificare le funzionalità di update in modo da riuscire ad aumentare la percentuale di nodi maligni in ogni routing table. Questo attacco è noto come *Eclipse Attack* [16–18, 22–24] ed ha un funzionamento molto semplice ma letale per le prestazioni della rete.

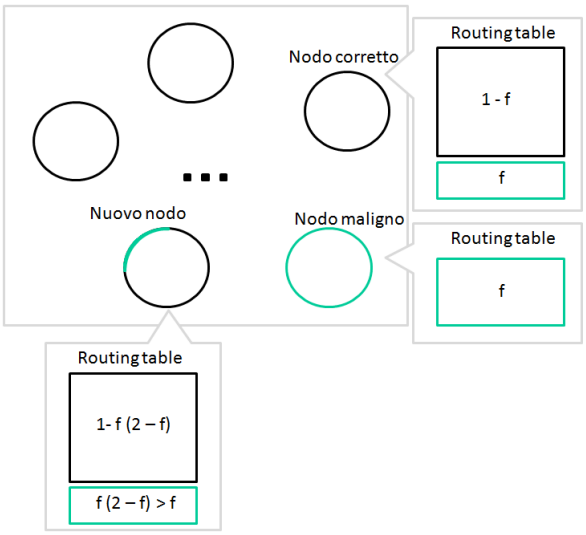


Figura 5.1: Esempio di routing table in una rete soggetta ad *Eclipse Attack*

Come si nota in Figura 5.1, il funzionamento dell'attacco Eclipse è basato sul semplice vincolo per il quale le routing table dei nodi maligni possono contenere solamente nodi maligni e quindi un aggiornamento fornito da un nodo maligno darà con probabilità 1 una entry maligna. Le routing table dei nodi corretti, invece, potranno contenere una percentuale di entry maligne pari ad f e quindi, con probabilità f, forniranno un aggiornamento 'maligno'.
Allora, siccome la percentuale dei nodi maligni nell'overlay è pari ad f si avrà che la probabilità di aggiornamento maligno sarà pari a:

$$(1 - f) \cdot f + f \cdot 1$$

Questa percentuale sarà significativamente maggiore di f e quindi, considerando update continui e frequenti durante tutto il periodo di vita dell'overlay, la percentuale di entry maligne nelle routing table continuerà ad aumentare. In questo modo, con un'esigua percentuale di nodi maligni all'interno dell'overlay, un attaccante potrà ottenere una percentuale di chiavi molto elevata.

Quindi, in un *Eclipse Attack*, un modesto numero di nodi maligni cospira per ingannare nodi corretti a farsi adottare come loro peer, con l'obiettivo di dominare le routing table ed i neighbor set. In caso di funzionamento con successo, l'attacco

Eclipse permetterà agli attaccanti di dominare la maggior parte del traffico dell'overlay e di 'eclissare' i nodi corretti dalla vista degli altri. All'estremo, l'attacco permetterà ai nodi maligni di controllare tutto il traffico della rete permettendo così l'applicazione di tutte le tipologie di attacchi di tipo DOS.
L'*Eclipse Attack* è da considerarsi collegato strettamente all'attacco Sybil [17, 19, 20] in quanto, siccome al termine di un attacco di questo tipo un singolo nodo maligno potrà assumere un alto numero di identificativi, esso può essere pensato come tramite, come punto di partenza, per lanciare in maniera più immediata ed efficiente un attacco Eclipse.

Soluzioni

Per permettere un mantenimento sicuro della routing table è importante imporre forti vincoli sul set di ID di nodo che possono riempire ogni slot della routing table. La soluzione proposta in [16] si basa sull'utilizzo di due tipologie di routing table: una per la gestione di informazioni sulla prossimità di rete per un routing efficiente, utilizzata per l'inoltro di messagggi e per ottenere buone performance, mentre la seconda per imporre vincoli sulle entry ed è il vero e proprio rimedio al problema della modifica delle routing update.

La difesa dall'attacco Eclipse effettuata con il metodo delle *CRT* (*Constrained Routing Table*), però, impone forti vincoli strutturali ai neighbor set.
In questo meccanismo di difesa i nodi avranno identificatori casuali e certificati e la lista dei vicini di un nodo sarà formata da nodi con ID vicini a punti ben definiti nello spazio degli identificatori. La certificazione degli ID riuscirà a prevenire l'attacco Sybil mentre il CRT riuscirà solamente a contrastare, anche se non a prevenire completamente, l'attacco Eclipse. Il problema di questo meccanismo di difesa, come detto, è legato al non lasciare flessibilità nella scelta dei vicini ed impedisce quindi ottimizzazioni come la *PNS* (*Proximity Neighbor Selection*), tecnica importante e largamente usata per l'aumento dell'efficienza dell'overlay.

È stata allora introdotta in [23, 24] una nuova tecnica di difesa contro l'attacco Eclipse nota come tecnica dell'*Enforcing Node Degree Bound*.
Questa tipologia di difesa è basata su di una semplice osservazione: durante un attacco Eclipse il grado d'ingresso di un nodo attaccante nel grafo di overlay sarà sicuramente maggiore rispetto al grado d'ingresso medio dei nodi corretti nell'overlay; di conseguenza un meccanismo per prevenire l'attacco è fare in modo che i nodi corretti scelgano vicini con grado d'ingresso non significativamente sopra la media. Nel caso in cui nell'overlay la relazione tra i vicini non sia riflessiva non è però sufficiente controllare il grado d'ingresso, ma occorre considerare anche il grado di uscita in quanto i nodi maligni potrebbero tentare di 'consumare' tutti i gradi d'ingresso dei nodi corretti in modo da indurli a scegliere il nodo maligno come vicino. Quindi i nodi corretti dovranno scegliere vicini con grado d'ingresso e grado di uscita inferiori ad una certa soglia.

Il problema di questo meccanismo è legato al fatto che non è semplice riuscire a verificare l'osservazione fondamentale prima introdotta; a questo scopo, allora, il punto chiave del meccanismo è il rinforzo del bound, che può essere garantito in varie modalità. La soluzione più banale potrebbe essere quella di usare un sistema centralizzato per tener traccia del grado di ogni membro dell'overlay, ma in questo modo si perderanno tutte le fondamentali peculiarità del paradigma P2P; una soluzione alternativa è allora un meccanismo distribuito nel quale i nodi partecipanti saranno responsabili del monitoraggio dei gradi degli altri nodi.

È stato allora deciso di adottare un meccanismo noto come *distributed auditing* per permettere il rinforzo del degree bound: ogni nodo nel sistema periodicamente fa da revisore dei suoi vicini per riuscire a controllare la corrispondenza con i gradi; per questo scopo ogni nodo x dell'overlay manterrà una lista, denominata *backpointer set*, di tutti i nodi che hanno x nella loro neighbor list. Ad intervalli di tempo casuali x sfiderà in modo anonimo ogni membro della sua neighbor list, chiedendogli la backpointer set e se il numero di entrate in questa tabella è superiore al bound del grado d'ingresso, o x non appare nella backpointer set allora il revisore decreterà il fallimento del test e rimuoverà quel particolare membro dalla sua neighbor list.
Inoltre, per prevenire che un attaccante 'consumi' il grado d'ingresso, viene attuata una procedura di revisione per garantire che i membri della backpointer set di un nodo riescano a mantenere una neighbor list di dimensione appropriata e quindi, nel caso in cui un nodo di revisione trovasse che uno dei suoi vicini non rispetta il degree limit, taglierà immediatamente la connessione associata.
Inoltre, per garantire che le risposte ad una sfida siano fresche ed autentiche, x includerà un nonce casuale nel messaggio inviato; in questo modo, siccome colui che dovrà essere revisionato dovrà includere il nonce nella risposta e firmare il messaggio, il nodo x potrà controllare l'autenticità e la freschezza della risposta.

L'ultima caratteristica base del meccanismo di Enforcing Node Degree Bound è legata al fatto che l'identità del revisionatore deve essere nascosta al nodo revisionato in quanto, se ciò non fosse, un nodo maligno potrebbe produrre risposte false. Nasce quindi un algoritmo detto di *anonymous auditing* ovvero legato alla costruzione di un canale di comunicazione per nascondere l'identità del revisore; questa features viene ottenuta tramite il concetto di *anonymizer node*, ovvero un terzo nodo casuale con il compito di proteggere la comunicazione tra revisore e revisionato. Il meccanismo di selezione deve prevedere la possibilità che l'anonymizer sia maligno ma che non potrà comportarsi durante questo processo in maniera errata poiché verrebbe immediatamente scoperto ed estromesso dall'overlay; esistono tre tecniche di selezione dell'anonymizer node.

- *Random node*; in questa tecnica il nodo viene scelto in maniera casuale tra l'intera popolazione prima di ogni revisione. Questa selezione si ottiene tramite la generazione di un numero casuale nello spazio degli ID e con l'uso del secure routing per scegliere il nodo 'vivo' con ID più vicino al numero generato.

Il problema di questo approccio è che un nodo maligno potrebbe potenzialmente superare la routing request del revisore.

- *Node closest to* $H(x)$; in questa tecnica viene selezionato il nodo con ID numericamente più vicino all'hash di x, utilizzando solitamente come funzione di hash *SHA-1*. Una volta che l'anonymizer viene trovato, esso potrà essere utilizzato continuamente e senza problemi fino a che continua ad essere il nodo più vicino ad $H(x)$. In questo modo, i nodi maligni che osservano, non potranno imparare nulla sulle attività dell'overlay, ma nel caso in cui il nodo scelto fosse maligno esso potrà rendere ogni revisione inefficace.

- *Random node among the* l *closest to* $H(x)$; in questa tecnica, sicuramente la più robusta e di conseguenza più utilizzata nelle implementazioni, il revisore seleziona come anonymizer un nodo casuale tra gli l nodi con ID più vicino ad $H(x)$. Siccome ogni revisore utilizzerà lo stesso set di nodi per la selezione dell'anonymizer, le richieste saranno intervallate ed i nodi maligni che osservano il traffico non potranno imparare nulla.

Con la difesa dell'Enforcing Node Degree Bound ogni nodo revisore potrà quindi trovare indipendentemente nodi maligni; il problema resta legato al fatto che un nodo attaccante potrebbe riuscire a rimanere nel sistema anche dopo un attacco ad un sottoset di nodi se riesce a mantenere un comportamento corretto verso gli altri; quindi questa tecnica risulta essere meno efficace nel caso in cui il nodo maligno abbia un comportamento ibrido. L'unico modo per migliorare il funzionamento anche in queste eventualità è permettere uno scambio fitto di informazioni tra i nodi in quanto, se più nodi riuscissero a dimostrare che il nodo in questione si è comportato in modo scorretto, esso potrà essere eliminato dall'overlay; naturalmente questo continuo scambio di informazioni comporterebbe un notevole spreco di energie ed aumenterebbe la complessità computazionale del meccanismo.

5.2.3 Invio sicuro dei messaggi

L'uso di ID di nodo certificati e del mantenimento sicuro delle routing table garantiscono che ogni tabella abbia al massimo una frazione f di entry legate a nodi maligni. Però, nel caso in cui si applichi l'instradamento tramite il metodo delle CRT, non si avrà un livello di sicurezza adeguato in quanto un attaccante potrebbe ridurre la probabilità di successo delle consegne semplicemente non instradando messaggi; queste tipologie di attacchi sono molto dannosi per gli overlay P2P poiché hanno effetti devastanti anche nel caso in cui la frazione f di nodi maligni è piccola.

Tutti gli overlay P2P definiscono delle primitive per permettere l'invio di un messaggio verso una chiave. In assenza di fallimenti il messaggio verrà consegnato al nodo 'radice' di una chiave in un massimo di h hop; però la consegna di questo messaggio potrebbe fallire nel caso in cui anche uno solo degli $h - 1$ nodi lungo il percorso di instradamento sia maligno.

I nodi maligni possono semplicemente eliminare il messaggio, instradarlo verso una destinazione errata o catturarlo impersonando il nodo radice. Quindi, nel caso di overlay con frazione f di nodi maligni, la probabilità di corretto instradamento tra due nodi corretti sarà

$$(1-f)^{h-1}$$

Il nodo radice di una chiave, inoltre, potrebbe essere maligno; le applicazioni P2P, come detto in precedenza, tollerano la presenza di nodi radice maligni tramite la replica delle informazioni associate ad una chiave su più nodi di riferimento. In questo modo la probabilità di un instradamento di successo verso un corretto nodo radice risulterà essere solamente

$$(1-f)^{h}$$

È quindi importante riuscire ad escogitare un meccanismo per garantire, nel limite del possibile, instradamenti sicuri.

Soluzioni

Quello che si vorrebbero definire sono delle *Secure Routing Primitive* che, forniti un messaggio ed una chiave di destinazione, riescano ad assicurare con probabilità molto alta che al minimo una copia del messaggio riesca a raggiungere ogni corretta radice di replica per la chiave. Naturalmente tutto questo dovrebbe essere fatto in modo efficiente.

L'approccio suggerito in [16–18, 22] è quello di effettuare l'instradamento nel modo più performante possibile per poi applicare un failure test in modo da determinare se l'instradamento ha effettivamente lavorato in modo corretto. Nel caso in cui il test fornirà esito negativo si procederà con il più costoso *Redundant Routing*.
Innanzitutto le primitive definite instraderanno il messaggio in modo efficiente verso la radice della chiave assegnata in base alle tabelle di instradamento locali. Quindi verranno generati dei set di radici di replica, riferite al nodo radice assegnato, ed applicato il failure test; nel caso in cui il risultato del test fosse negativo le repliche potranno essere accettate come corrette mentre, in caso di risultato positivo, le copie del messaggio verranno inviate tramite diversi instradamenti.

Il punto chiave in questo meccanismo di difesa è quindi il *Routing Failure Test*; questo test riceve una chiave ed un set di radici di replica per la chiave stessa ed il risultato fornito sarà negativo se il set di radici risulterà essere corretto per la chiave data. Altrimenti il risultato fornito sarà positivo, e questo sarà il risultato anche nel caso in cui, dopo la scadenza di un timer settato all'inizio del test, il meccanismo di instradamento non fornisca nessun risultato.
Il riuscire a scoprire fallimenti nella fase di routing potrebbe essere difficile e di conseguenza l'implementazione del test non è banale. Essa si basa sull'osservazione che la densità media di ID di nodo per unità di volume in uno spazio di identificatori sia

superiore della densità media di ID di nodo maligni e quindi lavorerà comparando la densità di ID di nodo nella neighbor list del mittente con la densità degli ID di nodo vicini alle radici di replica della chiave di destinazione. In questo modo overlay che devono distribuire repliche per una chiave uniforme sullo spazio degli ID potranno utilizzare questo controllo comparando la densità al mittente con la distanza media tra ogni chiave di replica e l'ID del suo nodo radice.

Nel caso in cui il failure test dia risultato positivo, il meccanismo di routing standard può essere sostituito con due tecniche alternative:

- *Redundant routing* [16, 22]; l'idea è semplicemente quella di instradare diverse copie del messaggio su cammini multipli verso ogni nodo radice della chiave di destinazione. In questo modo, nel caso venga inviato un sufficiente numero di copie ad ogni replica, con alta probabilità almeno una versione corretta del messaggio riuscirà a raggiungere il nodo radice. Quindi, tramite la creazione di strade alternative e l'invio di più messaggi identici lungo diversi instradamenti, si aumenterà di molto la probabilità di successo nella consegna della chiave.

- *Iterative routing* [18]; questo meccanismo inizia con il mittente che, guardando il next hop della sua routing table, imposta una variabile n come puntatore verso questo nodo. Quindi il mittente chiederà al nodo identificato da n di fornirgli il suo next hop ed aggiornerà il valore di n associandolo al nodo appena ricevuto. Il procedimento procederà così in modo iterativo aggiornando continuamente il valore del puntatore n fino a che non si raggiungerà la destinazione e solo a questo punto, stabilendo un collegamento diretto con la destinazione, si invierà il valore della chiave.
  Questo meccanismo riesce ad aumentare la probabilità di instradamento corretto anche se duplicherà il costo rispetto al Redundant routing.

5.3 Dati autocertificati

Le primitive di secure routing, però, aggiungono overhead significativo rispetto alle funzioni di routing tradizionali anche se il suo uso può essere ottimizzato ed il suo costo ridotto tramite l'uso di dati autocertificati.

L'incidenza del secure routing sul sistema, quindi, potrebbe essere ridotta immagazzinando dati autocertificati nell'overlay, solitamente per valori con integrità che può essere verificata dal client; questo permette al client stesso di utilizzare in modo efficiente il routing per richiedere la copia di un oggetto.
Se un client riceverà una copia di un oggetto, potrà verificare la sua integrità e ricorrere quindi alle primitive di routing sicuro solo nel caso in cui l'integrity check fallisca o non si ottengano risposte entro un dato periodo di timeout.

I dati autocertificati non aiuteranno nell'inserimento di nuovi oggetti nell'overlay o nella verifica del fatto che un particolare oggetto non sia immagazzinato all'interno dell'overlay; in questi casi è obbligatorio utilizzare le primitive di secure routing per

garantire che tutte le corrette radici di replica siano state raggiunte. Anche nelle operazioni di Join o di Leave di un nodo, per riuscire a ristabilire le corrette funzionalità, si necessita l'applicazione del secure routing anche se, nei casi più semplici e più comuni, l'uso di dati autocertificati potrebbe semplificare le operazioni e rendere così le funzionalità dell'overlay più veloci e più semplici.

Il problema dell'autocertificazione dei dati, che comunque può essere applicata a tutti i sistemi, riguarda il legame specifico con le particolari caratteristiche del protocollo e dell'applicazione e quindi non esiste un'implementazione univoca ma le modalità di realizzazione dipendono dal caso specifico considerato.

Capitolo 6

Implementazione dell'attacco Eclipse in Chord

La prima parte di lavoro effettuata è stata incentrata sull'applicazione della teoria relativa all'*Eclipse Attack*, già introdotto precedentemente, all'interno dell'overlay simulativo Chord realizzato sulla piattaforma OMNeT++/OverSim. Lo scopo del lavoro voleva essere quello di riuscire ad implementare l'attacco Eclipse in modo da poterne studiare dettagliatamente l'effetto sulle strutture di rete. Per poter raggiungere questo obiettivo si è deciso di implementare e di studiare innanzitutto il *Sybil Attack*, sicuramente più semplice anche se meno performante, per poi introdurre modifiche al comportamento del sistema allo scopo di realizzare l'attacco Eclipse.

6.1 L'attacco Sybil in Chord

L'approccio scelto per l'implementazione dell'attacco Eclipse è stato, come detto, il metodo classico, già discusso nel capitolo precedente, che consiste nell'implementazione dell'attacco Sybil e nel suo utilizzo come tramite per il lancio dell'*Eclipse Attack*.

Per l'implementazione del *Sybil Attack* [17, 19, 20] ci si è basati sulla possibile differenziazione tra nodi corretti e nodi maligni permessa all'interno del simulatore; in un overlay qualsiasi realizzato con OverSim, infatti, è sempre possibile creare una percentuale f di nodi maligni su di un totale di N nodi tramite l'impostazione del parametro

```
*.globalObserver.globalNodeList.maliciousNodeProbability
```

all'interno del file di configurazione *omnetpp.ini*.

Nella versione originale del simulatore un nodo maligno è però caratterizzato da un comportamento radicalmente differente rispetto a quello dei nodi corretti, in quanto esso ignora tutte le possibili richieste che gli vengono fatte, rendendo così inefficiente l'overlay.

Questa modalità di azione, però, non è coerente ne con lo sviluppo dell'attacco Sybil ne tantomeno con quello dell'attacco Eclipse poiché un nodo maligno di questo tipo, ignorando ogni possibile richiesta, non potrà essere in grado di entrare nell'anello Chord e quindi rimarrà isolato o verrà immediatamente scoperto a causa del suo comportamento radicalmente differente; in questo modo l'attaccante non avrà la possibilità di intercettare richieste e di catturare chiavi.

Per implementare un attacco Sybil, invece, un nodo maligno dovrebbe comportarsi esattamente come un nodo corretto in quanto, per la buona riuscita dell'attacco, la sua identità dovrebbe essere mantenuta segreta, in modo da rendere difficoltoso il rintracciamento.
Inoltre, in un *Sybil Attack*, un attaccante dovrebbe poter controllare un numero di nodi maligni sicuramente superiore ad uno. Allora, nell'implementazione, è stata fatta un'ipotesi: si è considerata la presenza di un solo attaccante, unico controllore della totalità dei nodi maligni dell'overlay; in questo modo tutti i *malicious node* saranno collusi e di conseguenza il conteggio dell'efficienza dell'attacco sul sistema risulterà molto più semplice.

Per riuscire a mantenere l'anonimato del generico nodo maligno, requisito base per il successo dell'attacco, si sono allora apportate delle modifiche al simulatore; in questo modo un qualsiasi nodo dell'overlay, in caso di richieste relative alla creazione o alla modifica dell'anello o di richieste legate all'instradamento di un messaggio applicativo, attuerà un comportamento identico a quello di un nodo corretto, accettando, quando possibile, le richieste ed instradando correttamente i messaggi ricevuti non destinati a lui. Così anche il nodo maligno avrà la capacità di costruirsi una finger table e di entrare all'interno di finger table di altri nodi, comportandosi come un nodo corretto ed essendo, di conseguenza, meno distinguibile dal punto di vista strutturale.

Come già descritto durante l'analisi dell'implementazione di Chord, la fase di ricerca del `successor(k)`, effettuata con l'applicazione `KBRTestApp`, consiste nell'instradamento di messaggi relativi ad una particolare chiave verso il nodo responsabile, ricercato attraverso i collegamenti tra le entry delle finger table.
Nel caso di attacco Sybil con una percentuale pari ad f nodi maligni sul totale, allora, l'operazione di ricerca resterà inalterata, con il risultato che l'attaccante controllore dei nodi maligni assumerà il controllo di una frazione circa pari ad f delle chiavi ricercate.

L'attacco Sybil in Chord, quindi, rispecchia pienamente le nozioni teoriche precedentemente introdotte e, come mostrato nel grafico in Figura 6.1, relativo ad un overlay di $N = 100$ nodi, al variare della percentuale di nodi maligni nel sistema (naturalmente sempre associati ad un singolo attaccante) il volume di chiavi controllato avrà una crescita lineare.
Questo particolare attacco, che ha il pregio della semplicità, mostra l'inevitabile difetto dell'alto costo implementativo in quanto se un attaccante volesse controllare una cospicua porzione di chiavi nel sistema, dovrebbe possedere e controllare una frazione altrettanto elevata di nodi nell'overlay; questo non sempre è possibile e, anche nel caso in cui lo fosse, ha un costo elevato.

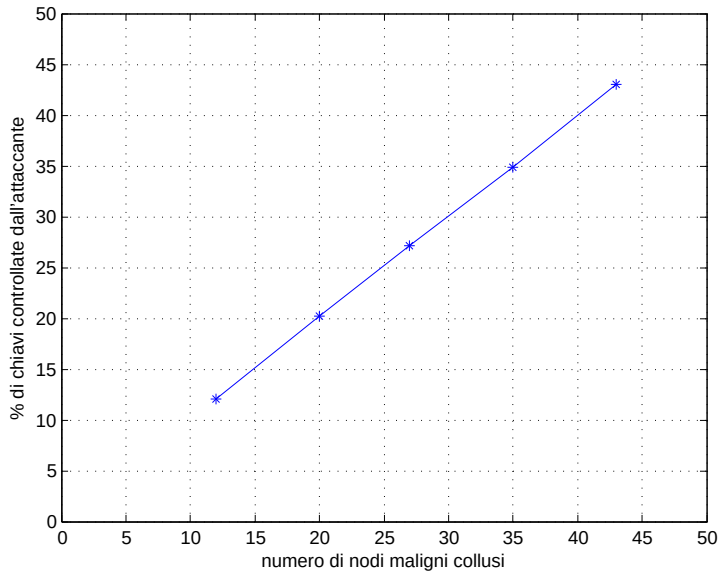


Figura 6.1: Percentuale di chiavi controllate dall'attaccante al variare del numero di nodi maligni in un overlay composto da $N = 100$ nodi soggetto a *Sybil Attack*, in caso di rete statica

È per questo motivo che l'attacco Sybil non è quasi mai utilizzato in autonomia ma è spesso impiegato come punto di partenza per riuscire a lanciare un attacco più efficace, come l'attacco Eclipse.

Risulta essere quindi interessante riuscire a comprendere a fondo il meccanismo alla base dell'implementazione di un *Eclipse Attack* partendo da un attacco Sybil, e studiare di conseguenza gli effetti dell'attacco stesso sulle prestazioni del sistema.

6.2 L'attacco Eclipse in Chord

L'implementazione dell'*Eclipse Attack* nell'overlay Chord ha richiesto particolare attenzione in quanto in [16–18, 22–24] le regole base dell'attacco non erano state definite rigorosamente e non era stata considerata un'implementazione in Chord.
Si è allora cercato di interpretare le istruzioni riferite ad un generico overlay, ovvero quelle descritte nel capitolo precedente, applicate però al simulatore OMNeT++/OverSim tramite la modifica dei moduli del protocollo Chord. Successivamente, tramite lo studio della distribuzione delle chiavi tra i nodi ed un postprocessing basato sulla struttura del grafo, sono stati valutati gli effetti dell'attacco. In questo modo si è riuscito a comprendere meglio l'effetto dell'*Eclipse Attack* sulle strutture di rete.

6.2.1 Implementazione dell'attacco

L'idea base dell'*Eclipse Attack* è quella di riuscire a catturare il maggior quantitativo possibile di chiavi nel sistema controllando una porzione relativamente piccola di nodi per avere, in modo poco oneroso, un ampio controllo sulla rete.
Come visto in precedenza, l'attacco Sybil è solamente un buon punto di partenza, ma non fornisce un'elevata efficienza in quanto la frazione di chiavi controllata è strettamente legata e direttamente proporzionale alla frazione di nodi collusi governati dall'attaccante.

L'*Eclipse Attack*, allora, partendo da questa situazione, deve implementare un meccanismo per il quale i nodi maligni riescano, senza particolari sforzi e senza costi elevati, ad allargare la loro area di controllo in modo da incrementare l'efficienza dell'attacco stesso.
In accordo con le idee presentate in [16–18, 22] ed ampliate in [23, 24], anche se in relazione ad un particolare meccanismo di difesa, si è deciso di implementare l'attacco Eclipse all'interno dell'overlay Chord come una modifica del comportamento dei nodi maligni realizzata dopo l'applicazione di un attacco Sybil; quindi il generico nodo maligno non si comporterà più esattamente come un nodo corretto ma introdurrà delle variazioni comportamentali in modo da incrementare l'efficienza dell'attacco, naturalmente preservando le caratteristiche base del protocollo Chord, in modo tale da rendere comunque difficile la sua identificazione.
Queste modifiche possono essere suddivise in due fondamentali variazioni legate alle due funzionalità base dell'attacco e quindi, dopo una fase di studio, sono state attuate sul protocollo Chord implementato sulla piattaforma di riferimento.

Modifica alla funzione di routing

La prima fondamentale modifica è stata quella legata all'instradamento delle richieste e alla creazione della finger table. L'obiettivo finale è quello di incrementare, senza costi aggiuntivi, l'area d'azione dei nodi maligni e per farlo ci si è basati sulla semplice relazione matematica, già introdotta nel capitolo precedente, legata alla frazione dei nodi maligni nell'overlay.
In un attacco Sybil, infatti, sia in caso di nodo corretto che in caso di nodo maligno, si ha una frazione pari ad f (percentuale di nodi maligni nel sistema) entry maligne nella finger table; in questo modo la probabilità che il generico nodo, rispondendo ad una `find node`, fornisca ad un nuovo nodo un riferimento maligno sarà pari ad f.
Se però un nodo maligno, conscio delle richieste che gli potrebbero essere fatte in fase di creazione dell'anello, decidesse di fornire riferimenti legati esclusivamente ad altri *malicious node*, la frazione di entry maligne all'interno di una nuova finger table potrebbe essere continuamente incrementata.

È proprio su questa idea che si basa il *routing modificato* dei nodi maligni nell'attacco Eclipse: in uno scenario in cui tutti i *malicious node* sono sotto il controllo dello stesso attaccante, che è proprio lo scenario di riferimento che è stato scelto per

le varie implementazioni, un singolo nodo colluso conoscerà tutti gli altri nodi maligni del sistema ed allora potrà imporre una modifica alla creazione della sua finger table ammettendo come nodi responsabili delle sue entry esclusivamente altri nodi che sono sotto il controllo dell'attaccante; in questo modo il contributo fornito dal nodo in questione ad un'operazione di ricerca potrà essere esclusivamente maligno, a differenza dei nodi corretti i quali potranno fornire un contributo che sarà solamente con probabilità f maligno.
Con questa modifica si avrà che una generica richiesta di entry da parte di un nuovo nodo potrà essere soddisfatta in modo differente a seconda se la risposta verrà fornita da un nodo maligno o da un nodo corretto; infatti, un nodo maligno, che avrà probabilità di essere scelto pari ad f, fornirà con probabilità 1 una entry maligna mentre un nodo corretto, che verrà selezionato con probabilità pari ad $1 - f$, fornirà un riferimento maligno con probabilità f. Di conseguenza la probabilità per una nuova entry di essere maligna sarà

$$f \cdot 1 + (1 - f) \cdot f$$

che risulta essere sicuramente maggiore di f. In questo modo, iterando, la percentuale percepita di riferimenti maligni nelle finger table aumenterà continuamente, incrementando così l'efficienza dell'attacco.
Per attuare questa modifica è stato necessario intervenire sia sul modulo generico `Chord` sia sui moduli specifici per le liste `ChordSuccessorList` e `ChordFingerTable` andando ad alterare le procedure di creazione di una nuova entry e di update di entry esistenti; durante queste modifiche si è dovuto prestare attenzione a mantenere la classica struttura ad anello in modo da preservare i principi base del protocollo originale. Inoltre, per rendere onniscienti i nodi maligni ed implementare quindi il grado di conoscenza prima descritto, secondo il quale i nodi maligni sono tutti collusi cioè tutti legati ad uno stesso attaccante ed ogni *malicious node* conosce tutti gli altri, si è dovuto intervenire su alcuni moduli generici di OMNeT++/OverSim, relativi alla caratterizzazione dei nodi ed all'elenco dei nodi presenti in ogni istante in rete; a questo scopo si è allora realizzata una lista, accessibile esclusivamente da parte dei nodi collusi, contenente l'elenco di tutti i nodi controllati dall'attaccante.

L'attacco Eclipse così realizzato è applicabile sia a reti statiche che a reti dinamiche ma, come si vedrà poi nella sezione del capitolo relativa alla valutazione dell'attacco, in caso di reti dinamiche l'efficienza salirà molto più rapidamente grazie alla continua necessità di update e quindi al continuo incremento della percentuale f percepita all'interno delle finger table dei nodi entranti.

Modifica alle operazioni di ricerca

In caso di *Sybil Attack*, come già accennato in precedenza, sia i nodi corretti che i nodi maligni all'interno della rete instradavano i messaggi applicativi secondo lo schema stabilito da `KBRTestApp`, secondo il meccanismo classico di Chord. In caso di attacco Eclipse, invece, anche la fase di ricerca del nodo responsabile deve essere modificata in

quanto un nodo maligno dovrà cercare di catturare il massimo quantitativo possibile di chiavi che circolano nel sistema.
Per raggiungere questo scopo si è pensato di introdurre delle modifiche comportamentali secondo le quali, nel momento in cui una particolare richiesta per l'individuazione del responsabile di una chiave raggiunge un qualsiasi nodo maligno, esso risponde impersonificando il ruolo del responsabile e catturando di conseguenza la chiave; in questo modo l'efficienza dell'attacco risulterà notevolmente amplificata.
Per implementare questa soluzione è stata necessaria una modifica sia al modulo applicativo `KBRTestApp` che al modulo `Chord`, oltre ad altri moduli generici per la gestione del trattamento e dell'instradamento dei messaggi applicativi.
Questo è stato necessario poiché, nel momento in cui un messaggio applicativo contenente una particolare chiave arriva ad un nodo corretto,

- viene trattenuto nel caso in cui il nodo sia il reale responsabile di quella chiave
- viene instradato secondo il meccanismo di Chord analizzato precedentemente nel caso in cui il nodo in questione non sia il corretto *successor(k)*

Se però il nodo in questione è maligno, grazie alle modifiche attuate esso impersonificherà in ogni caso il destinatario della chiave e catturerà il messaggio; in questo modo, quindi, ogni nodo maligno si comporterà da *successor(k)* per ogni chiave che vede transitare.

Naturalmente, perché la fase di ricerca abbia successo, deve essere implementata correttamente la fase di routing modificato descritta in precedenza in quanto, se un nodo maligno nell'overlay fosse presente solamente in poche entry delle finger table, esso potrà essere contattato molto raramente dalle operazioni di ricerca e quindi riuscirà a catturare esclusivamente un piccolo quantitativo di chiavi. Risulta quindi chiaro che le due operazioni di modifica sono tra di loro collegate e sono entrambe di fondamentale importanza per la buona riuscita e per l'alta efficienza dell'attacco.

Terminata la fase di modifica e di implementazione dell'*Eclipse Attack* all'interno dell'overlay Chord sono state effettuate varie simulazioni e si è cercato di interpretare i risultati per capire se realmente l'attacco così effettuato è coerente con le nozioni teoriche introdotte nei capitoli precedenti.

6.2.2 Analisi della distribuzione delle chiavi

La prima fase di interpretazione dei risultati è stata quella relativa alla valutazione della gestione e del possesso delle chiavi tra i vari nodi.
Inizialmente si sono analizzate, come fatto anche per l'attacco Sybil, le percentuali delle chiavi legate a nodi maligni sia in caso di overlay statico che di overlay dinamico; successivamente sono state suddivise le chiavi tra i vari nodi responsabili in modo tale da riuscire a valutare l'incidenza dell'attacco sul *consistent hashing*.

Variazione del volume di chiavi assegnato a nodi maligni

Andare ad analizzare il volume di chiavi controllato dai nodi maligni è il primo e fondamentale metodo per stabilire l'efficacia dell'attacco.
In caso di attacco Sybil, come visto in precedenza, la percentuale di chiavi controllate dai nodi maligni rispecchia la percentuale di nodi maligni controllati dall'attaccante nel sistema, in quanto non vengono apportate modifiche ne al meccanismo di funzionamento ne tantomeno al comportamento dei nodi.
Nel caso di *Eclipse Attack*, invece, grazie alle modifiche implementative, l'attaccante riuscirà, a pari costo computazionale, a controllare una porzione molto più ampia di chiavi nel sistema e di conseguenza l'attacco sarà molto più performante.

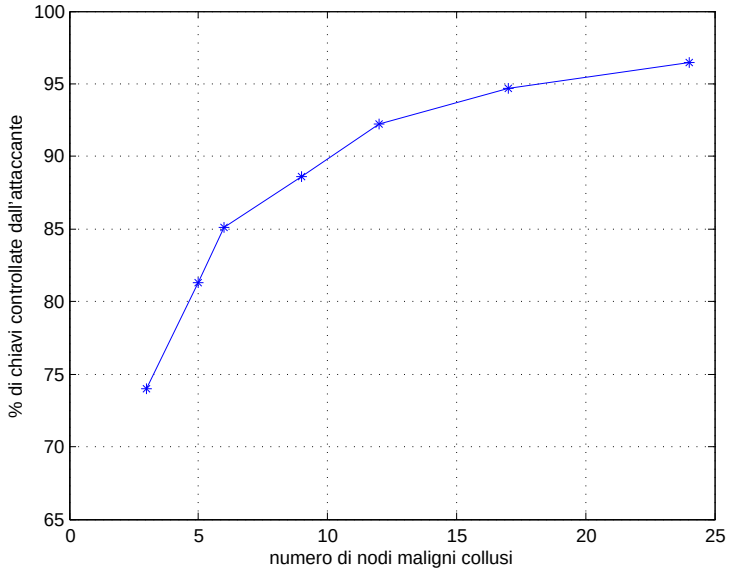


Figura 6.2: Percentuale di chiavi controllate dall'attaccante al variare del numero di nodi maligni in un overlay formato da $N = 100$ nodi soggetto ad *Eclipse Attack*, in caso di rete statica

Il primo risultato utile alla descrizione delle prestazioni è legato alla crescita del volume di chiavi all'aumentare della frazione di nodi maligni controllati dall'attaccante. Come si nota dal grafico in Figura 6.2, realizzato considerando un overlay con $N = 100$ nodi, all'aumentare della percentuale di nodi maligni nel sistema il volume di chiavi controllate dall'attaccante cresce molto rapidamente. Dall'analisi dei grafici di Figura 6.1 e di Figura 6.2, si nota facilmente come, con una spesa minore rispetto all'attacco Sybil, il volume di chiavi controllato dall'*Eclipse Attack* risulta essere molto maggiore; addirittura già in caso di $f = 12$, cioè controllando circa il 12% dei nodi del sistema, l'attaccante che ha lanciato l'attacco Eclipse riuscirà a catturare un

quantitativo molto elevato di chiavi memorizzate (più del 90%).

Un altro parametro fondamentale per la valutazione delle prestazioni dell'attacco Eclipse è la grandezza dell'overlay; le prestazioni dell'attacco, infatti, mutano al variare della dimensione dell'overlay, in quanto più è ampio più le finger table conterranno entry e quindi potranno essere maggiormente inquinate dai nodi maligni.
Nel grafico in Figura 6.3 è mostrato l'andamento del volume di chiavi controllate dall'attaccante al variare della frazione di nodi maligni nel sistema in caso di overlay di dimensioni differenti; si nota chiaramente come, nel caso di rete formata da $N = 1000$ nodi, l'attacco risulta essere molto più performante in quanto, con una frazione meno elevata di nodi maligni rispetto al totale, l'attaccante riuscirà ad ottenere il controllo della quasi totalità delle chiavi del sistema. Questo dimostra quindi come nelle reti reali, che solitamente sono molto estese, l'applicazione dell'*Eclipse Attack* riesca a fornire elevate prestazioni anche con una spesa in proporzione molto bassa.

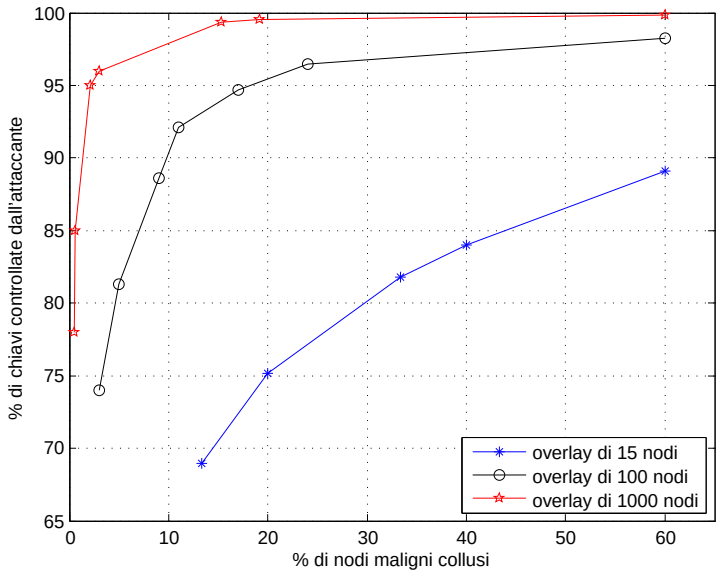


Figura 6.3: Percentuale di chiavi controllate dall'attaccante al variare del numero di nodi maligni per overlay di dimensione pari ad $N = 15$ nodi, $N = 100$ nodi ed $N = 1000$ nodi, in caso di rete statica

Come già accennato precedentemente, però, le prestazioni dell'attacco Eclipse dovrebbero migliorare in caso di rete dinamica, cioè nel caso più reale in cui i nodi possono entrare ed uscire dal sistema rispettando a perfezione il paradigma P2P.
Per riuscire ad analizzare l'effetto della dinamicità sulla distribuzione delle chiavi, si sono effettuate differenti simulazioni considerando un modello `*.underlayConfigurator` di tipo `lifetimeChurn` che, come già introdotto nell'analisi di OMNeT++/Over-

Sim, permette l'impostazione di un tempo medio di vita ai nodi e quindi la simulazione della loro nascita e della loro morte.

Analizzando differenti simulazioni si è osservato come l'efficienza dell'*Eclipse Attack* cambi sia al variare del tempo di simulazione stesso sia al variare del tempo medio di vita dei nodi; questo perché:

- tenendo costante il tempo medio di vita dei nodi ed aumentando il tempo di simulazione aumenterà il numero di Join e di Leave e di conseguenza l'inquinamento apportato dai nodi maligni alle finger table; questo provocherà un aumento di efficienza dell'attacco
- tenendo costante il tempo di simulazione e riducendo il tempo medio di vita dei nodi aumenterà il numero di Join e di Leave e, come nel caso precedente, l'efficienza dell'attacco

Il parametro da monitorare in caso di reti dinamiche, il parametro che determina realmente l'efficienza dell'*Eclipse Attack*, è quindi il numero di Join e di Leave che avvengono all'interno del sistema. Questo parametro sarà indipendente dal tempo medio di vita di un nodo e dal tempo di simulazione in quanto dipenderà solamente dal loro rapporto, dal loro legame.

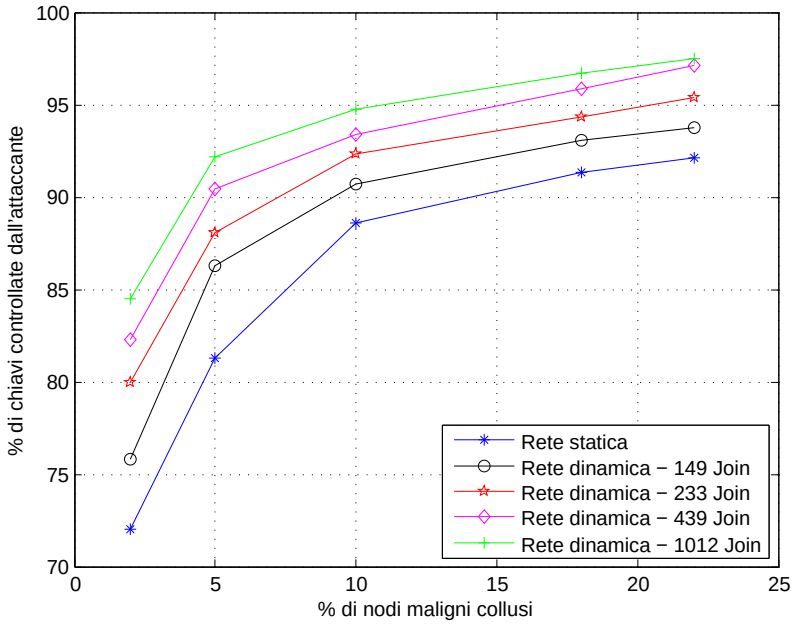


Figura 6.4: Percentuale di chiavi controllate dall'attaccante al variare del numero di nodi maligni in un overlay composto da $N = 100$ nodi, sia nel caso di rete statica che di rete dinamica

In Figura 6.4 è mostrato come, in un overlay formato da $N = 100$ nodi, all'aumentare del numero di Join e di Leave aumenta l'efficienza dell'attacco; si nota infatti molto chiaramente come l'efficienza dell'*Eclipse Attack* aumenti passando da una rete statica, senza Join e Leave, ad una rete dinamica e come migliori all'aumentare del numero di Join e Leave. Questo è facilmente deducibile dalla relazione, già introdotta precedentemente, per cui la frazione di entry maligne all'interno di un nuovo nodo risulta essere

$$f \cdot 1 + (1 - f) \cdot f > f$$

e quindi all'aumentare del numero di Join e di Leave la frazione f percepita di entry maligne continua ad aumentare incrementando la probabilità con cui una chiave, durante il tragitto tra i riferimenti delle tabelle di Chord, passi da un nodo maligno e venga catturata, influenzando di conseguenza l'efficienza dell'attacco.

La variazione del numero di Join e di Leave, strettamente legata sia al tempo di simulazione che al tempo medio di vita di un nodo, quindi, sarà sempre il parametro di riferimento per la valutazione della dinamicità della rete in quanto l'impostazione di un particolare tempo di simulazione o di uno specifico tempo medio di vita di un nodo sono situazioni molto aleatorie e di difficile definizione mentre il considerare l'andamento rispetto al numero di Join e Leave rende la valutazione più generica in quanto, così facendo, non si vanno a considerare particolari istanti di simulazione, magari scelti casualmente, ma ci si basa esclusivamente sul loro rapporto, sicuramente più generico e rappresentativo.

Ripartizione delle chiavi ai singoli nodi

Considerando un generico sistema P2P è di vitale importanza riuscire a capire quando esso sarà soggetto ad un *Eclipse Attack* e quando invece il funzionamento sarà da considerarsi corretto. Un qualsiasi nodo, per riuscire a riconoscere la presenza di attacco Eclipse, però, non dovrebbe per forza conoscere i dati relativi a tutti gli altri nodi dell'overlay in quanto questo grado di conoscenza sarà raramente disponibile ed in ogni caso molto costoso da procurare. Quindi l'obiettivo è quello di permettere ad un singolo terminale di riconoscere la presenza dell'attacco esclusivamente tramite l'analisi dei dati in suo possesso.
A questo scopo è stata necessaria un'analisi centrata sul singolo nodo per riuscire ad estrarre un parametro, o un metro di giudizio, tramite il quale un generico partecipante possa riuscire a capire, con un grado di certezza abbastanza elevato, se la rete è soggetta o meno all'attacco Eclipse.
Verranno ora analizzati dei parametri di studio che sono stati scelti per poter comprendere se è in atto un attacco mentre nel capitolo seguente questi concetti verranno ripresi e formalizzati all'interno di un modello matematico e di un algoritmo applicabile a dei casi reali.

Per ottenere questo obiettivo ci si è basati ancora sul *consistent hashing*, concetto di fondamentale importanza per il funzionamento del protocollo Chord. Secondo questo principio le chiavi assegnate ai nodi di un overlay dovrebbero ripartirsi in modo uniforme tra i vari nodi in modo da affidare una quota più o meno costante a tutti e non avere quindi nodi scarichi ed altri troppo utilizzati, e di conseguenza lenti nel fornire informazioni in caso di richieste.
L'analisi considerata è fondamentale in quanto, come detto, non comporta la necessità di conoscere lo stato di tutti i nodi dell'overlay ma un singolo nodo, analizzando esclusivamente i dati in suo possesso, può essere in grado di capire, con un buon livello di sicurezza, se il sistema è soggetto ad un attacco Eclipse o se il suo funzionamento è corretto.

L'idea implementativa è stata quella di andare a memorizzare, per ogni singolo nodo, tre parametri specifici, tre contatori, che subiscono poi modifiche durante la fase di misurazione.

- *Numero di ricerche iniziate dal nodo x*; questo contatore verrà incrementato ogni volta che il nodo in questione inizierà la ricerca del *successor(k)*, ovvero genererà un messaggio applicativo legato alla chiave considerata. Questo è possibile in quanto, utilizzando `KBRTestApp` come modulo applicativo, queste richieste avverranno casualmente nel tempo, e si avrà quindi una distribuzione più o meno costante tra i vari nodi dell'overlay.

- *Numero di ricerche in transito nel nodo x*; questo contatore verrà incrementato ogni volta che x instraderà verso un altro nodo, secondo il meccanismo di ricerca di Chord, il messaggio applicativo. Naturalmente un qualsiasi nodo maligno, impersonificando il responsabile di ogni chiave che gli verrà instradata, avrà questo parametro sempre pari a 0.

- *Numero di chiavi memorizzate nel nodo x*; questo contatore verrà incrementato ogni volta che il nodo diventerà *successor(k)*, ovvero ogni volta che il messaggio applicativo troverà in x il nodo destinazione.

Indice del nodo	R_e	K_t	K_m
24	7	16	5
25	9	21	9
26	7	15	9
27	8	20	12
28	7	15	11
29	8	19	8

Tabella 6.1: Esempio di valori dei parametri di riferimento relativo ad un overlay Chord non soggetto ad attacco

Indice del nodo	R_e	K_t	K_m
24	155	305	10
25	158	347	9
26	157	0	467
27	154	305	9
28	157	303	9
29	156	301	11

Tabella 6.2: Esempio di valori dei parametri di riferimento relativo ad un overlay Chord in cui è in atto un *Eclipse Attack*

In Tabella 6.1 ed in Tabella 6.2 sono rappresentati due esempi raffiguranti i parametri prima descritti; in entrambe le tabelle la prima colonna contiene un numero incrementale per identificare i differenti nodi, mentre le altre tre colonne rappresentano, rispettivamente, le ricerche effettuate (R_e), le chiavi in transito (K_t) e le chiavi memorizzate (K_m) nei nodi presi in considerazione. La simulazione rappresentata in Tabella 6.1 è relativa ad un overlay non soggetto ad attacco ed infatti, confrontando ricerche effettuate e chiavi memorizzate, si può notare come venga rispettato il *consistent hashing*. La simulazione riportata in Tabella 6.2, invece, rappresenta uno scenario soggetto ad attacco e questo lo si nota dalla netta discordanza tra i parametri di riferimento, in quanto il numero delle chiavi memorizzate è molto minore del numero delle ricerche effettuate in caso di nodo corretto mentre è decisamente più alto in caso di nodo maligno. Inoltre, sempre in caso di *malicious node*, il parametro rappresentante il numero di chiavi in transito è pari a 0 in quanto, come già spiegato in precedenza, l'attaccante cercherà di bloccare tutte le chiavi che transiteranno attraverso i nodi collusi rendendo nullo proprio il numero di chiavi transitanti.

Per valutare le variazioni che l'attacco Eclipse impone sul *consistent hashing*, prima di analizzare i risultati relativi ad un overlay soggetto ad attacco, si è dovuta considerare la situazione di overlay Chord standard, senza nodi maligni e nessun attacco in atto; l'istogramma in Figura 6.5, relativo ad un overlay di $N = 100$ nodi, mostra come vi sia una equa distribuzione nella memorizzazione delle chiavi tra tutti i nodi in quanto per tutti il parametro K_m/R_e, dove K_m, come già accennato in precedenza, rappresenta le `chiavi memorizzate` ed R_e le `ricerche effettuate`, sarà sempre circa pari ad 1; questo significa che, in un overlay Chord non soggetto ad attacco, tutti i nodi gestiscono circa lo stesso quantitativo di chiavi. In questo modo il generico nodo x, controllando esclusivamente i suoi parametri e notando che il numero di ricerche iniziate è circa pari al numero di chiavi per cui lui è *successor(k)*, capirà che nella rete vi è equa distribuzione delle chiavi e quindi, con buona probabilità, non è in atto un *Eclipse Attack*.

Gli istogrammi in Figura 6.6 mostrano invece la distribuzione delle chiavi ad ogni nodo in caso di attacco Eclipse in atto. L'overlay considerato è sempre quello formato da $N = 100$ nodi e le condizioni al contorno per la simulazione sono sempre le stesse,

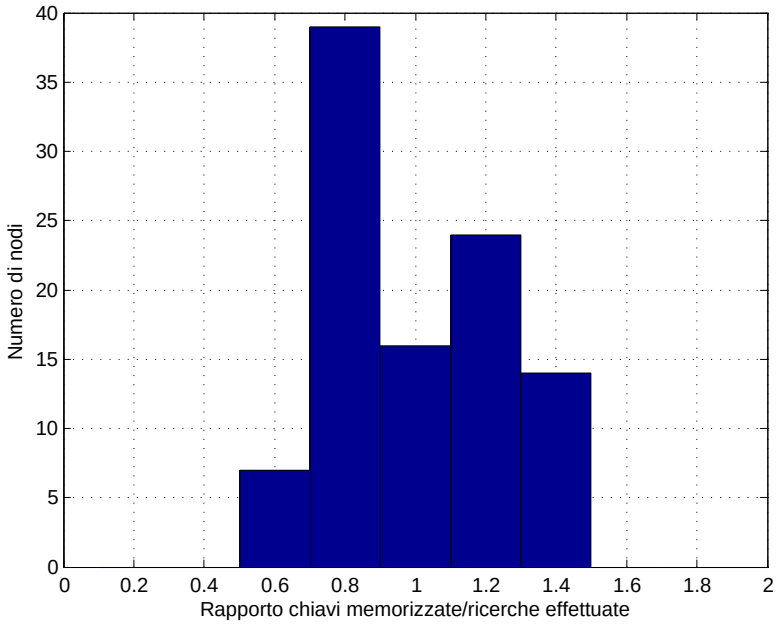
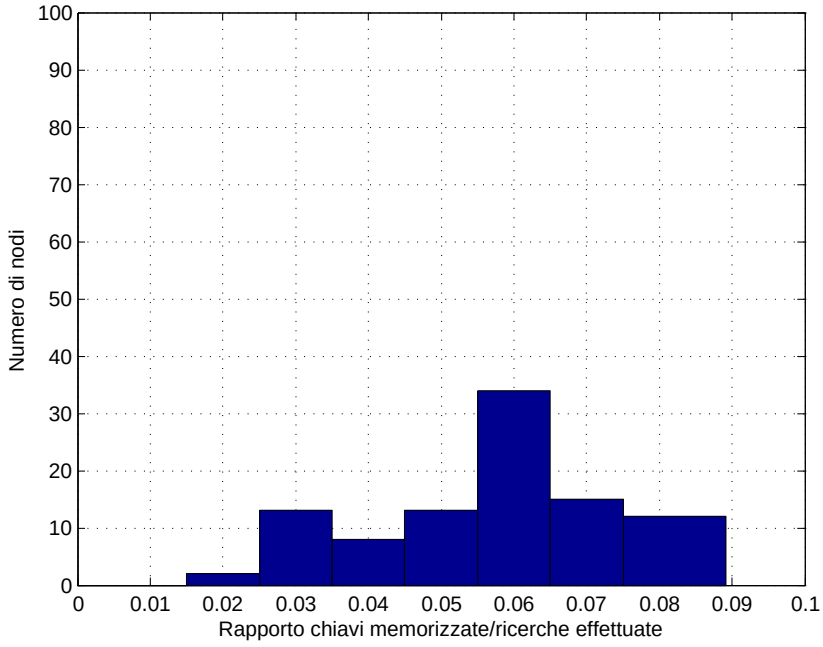


Figura 6.5: Distribuzione empirica del rapporto `chiavi memorizzate/ricerche effettuate` (K_m/R_e) in un overlay Chord con $N = 100$ nodi non soggetto ad attacco

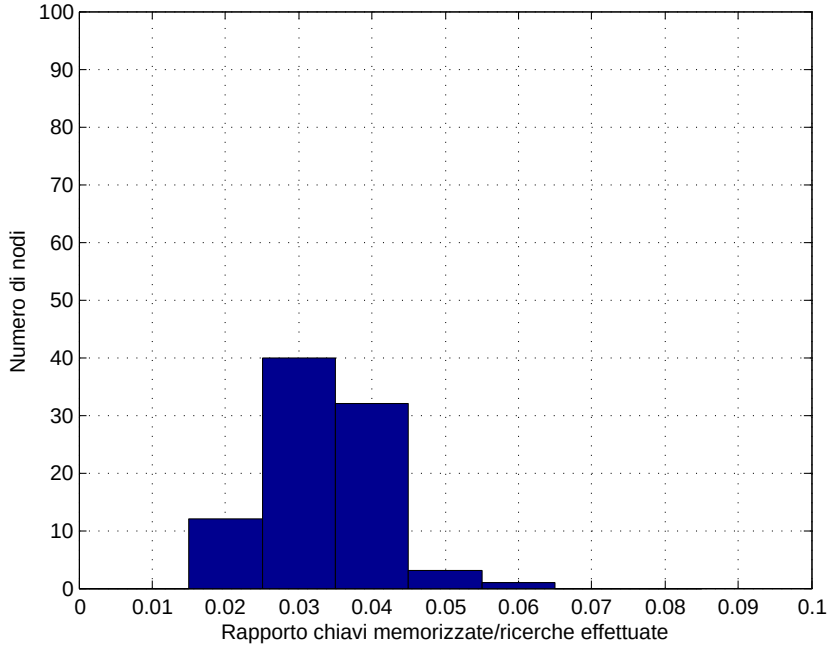
ma nel primo caso si ha una percentuale $f = 3\%$ di nodi maligni nel sistema mentre nel secondo caso la percentuale di nodi collusi è pari al 12%. Nella rappresentazione sono stati trascurati i nodi maligni che, come detto, hanno un rapporto K_m/R_e molto maggiore di 1; in questo modo si nota chiaramente come il parametro, monitorato nei nodi corretti, risulti essere molto prossimo a 0 in quanto questi nodi saranno responsabili di un basso numero di chiavi a causa proprio dei nodi maligni che, come già mostrato in Tabella 6.2, gestiranno una frazione di chiavi molto maggiore rispetto alle ricerche effettuate.

In questo modo, già da una sommaria analisi degli istogrammi, si riesce facilmente a capire se l'overlay in esame è soggetto ad attacco o meno in quanto, in caso di presenza di nodi maligni, la rappresentazione non sarà più centrata intorno al valore 1, ma intorno ad un valore molto prossimo a 0.

In più, da un'analisi degli istogrammi di Figura 6.6 si può notare come, senza dover fare uno studio molto approfondito, è semplice dare una stima del livello di penetrazione dell'attacco in quanto all'aumentare della percentuale di nodi maligni nel sistema il rapporto K_m/R_e si sposterà sempre più verso lo 0. Questa importante valutazione può essere tratta anche dal grafico riportato in Figura 6.7, rappresentante l'andamento del rapporto K_m/R_e al variare della percentuale di nodi maligni all'interno dell'overlay, composto sempre da $N = 100$ nodi. Si nota facilmente, infatti,



(a)



(b)

Figura 6.6: Distribuzione empirica del rapporto `chiavi memorizzate/ricerche effettuate` (K_m/R_e) in un overlay Chord con $N = 100$ nodi soggetto ad *Eclipse Attack* con percentuale di nodi maligni nel sistema pari ad $f = 3\%$ (a) ed $f = 12\%$ (b)

come il rapporto, che in caso di assenza di nodi maligni è circa pari ad 1, diminuisca sempre più all'aumentare della percentuale di nodi maligni all'interno del sistema, dimostrando quindi l'abilità dell'attaccante nel catturare un volume di chiavi sempre più ampio.

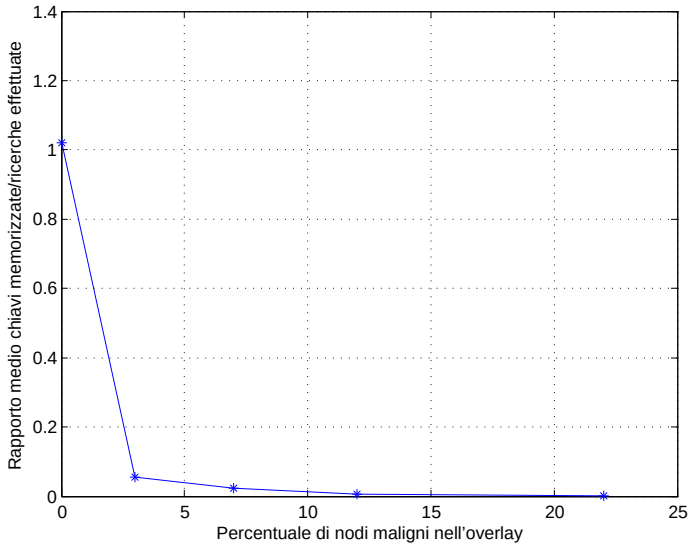


Figura 6.7: Valore medio del rapporto K_m/R_e al variare del numero di nodi maligni all'interno di un overlay composto da $N = 100$ nodi

Infine, grazie ai parametri monitorati, si può essere in grado, come accennato in precedenza, di effettuare una valutazione specifica in quanto il generico nodo x potrà riuscire a capire in autonomia se è in atto un attacco Eclipse; infatti, se da un semplice confronto tra il numero di ricerche effettuate ed il numero di chiavi memorizzate, esso troverà una netta discordanza, capirà che ci saranno sicuramente nodi in possesso di un alto numero di chiavi, ovvero non sarà più rispettato il fondamentale paradigma del *consistent hashing*.
Il problema di questo meccanismo di valutazione è legato al fatto che, sia il parametro K_m che il parametro R_e sono molto variabili e forniranno quindi un'indagine che può essere considerata esclusivamente approssimativa. Nel capitolo seguente, allora, ci si baserà sul parametro relativo alle chiavi in transito che permetterà un'analisi più robusta e funzionale.

6.2.3 Analisi del grafo

La seconda fase di interpretazione dei risultati si è incentrata sull'analisi delle proprietà strutturali del grafo. Questa fase è stata caratterizzata da operazioni di post-processing sui risultati estratti dal simulatore e relativi alle entry delle finger table

dei nodi della rete. È un'operazione fondamentale in quanto il riuscire ad analizzare l'incidenza dell'attacco sulla struttura del grafo permette in modo molto più semplice l'individuazione di scenari soggetti ad *Eclipse Attack*.
Per ottenere questo obiettivo sono stati considerati due differenti metodi di analisi.

- In primo luogo ci si è basati sul meccanismo della *Google Matrix* [25], procedimento, legato alla strategia attuata dal motore di ricerca Google, molto utilizzato per la caratterizzazione di grafi direzionati.

- Successivamente si è analizzato il grafo basandosi sulla teoria legata alla *Matrice Laplaciana* [26], meccanismo realizzato ad hoc per grafi non direzionati e quindi non esattamente mirato al caso in esame, ma comunque utile per fornire uno studio qualitativo anche in caso di archi orientati.

La Google Matrix

La Google Matrix [25] è una particolare struttura a grafo utilizzata per riordinare i risultati nelle ricerche web realizzate con il motore di ricerca Google, ed è sicuramente la tecnica di riferimento per l'analisi di grafi direzionati.
Infatti, per un motore di ricerca qualsiasi, è di fondamentale importanza riuscire a fornire, nel minor tempo possibile, riferimenti ai vari siti web collegati ad una certa parola chiave che l'utente potrebbe digitare. Il principale problema è legato al fatto che bisognerebbe effettuare in modo ottimale un *ranking*, ovvero riuscire ad ordinare i risultati in modo da poter identificare rapidamente quelli più coerenti.
All'interno del panorama internet, però, nascono due fondamentali problemi: innanzitutto esistono *web site* senza link diretti verso altri siti ed inoltre è necessario identificare un meccanismo per riconoscere i siti web che rispecchiano maggiormente la parola chiave ricercata.

Il motore di ricerca Google risolve queste problematiche in modo efficiente realizzando una struttura di rete nota appunto come Google Matrix che permette di soddisfare un alto volume di clientela e garantisce una ricerca rapida; per quanto riguarda la raggiungibilità, siccome è necessario che le informazioni raccolte attraverso la rete internet ritornino al punto di partenza in modo da poter consegnare il loro contributo, nel momento in cui viene raggiunto un sito web senza link diretti verso nessun altro sito, la ricerca verrà inoltrata casualmente ad un web site qualsiasi del panorama internet, in modo da realizzare un grafo virtualmente connesso, che ha per nodi i vari web site, eliminando i possibili 'vicoli cechi'. Per quanto riguarda l'output fornito dalla ricerca, ovvero la visualizzazione dell'elenco dei siti web in ordine di coerenza con la parola digitata dall'utente, ci si basa sulla probabilità che ha ogni pagina all'interno della Google Matrix di essere visitata, probabilità che viene ricavata monitorando le visite tramite un opportuno camminatore casuale.

Perché si possa realizzare questa idea implementativa, bisogna trarre delle considerazioni sulla *matrice di adiacenza* del grafo di partenza: questa matrice dovrà avere delle caratteristiche particolari per poter essere considerata una Google Matrix.

La matrice di adiacenza è una matrice binaria quadrata $N \cdot N$, con N numero di nodi nel grafo, dove l'elemento in posizione (i,j) vale 1 se esiste nel grafo un arco che va dal vertice i al vertice j e 0 altrimenti. Perché essa possa essere definita Google Matrix, appunto, devono essere verificate tre fondamentali proprietà:

- *Stocasticità*; la matrice di adiacenza deve essere stocastica, ovvero la somma degli elementi di ogni riga deve essere pari ad 1.
- *Aperiodicità*; la matrice di adiacenza deve essere aperiodica, ovvero alcuni stati devono poter essere raggiunti solamente ad istanti di tempo multipli di un certo valore.
- *Irriducibilità*; la matrice di adiacenza deve essere irriducibile, ovvero deve esserci una probabilità non nulla di raggiungere un qualsiasi stato. Per ottenere questa proprietà i creatori di Google hanno pensato, come già accennato in precedenza, di collegare a web site qualsiasi tutti i vicoli cechi garantendo così l'irriducibilità e permettendo il funzionamento del meccanismo di ricerca tramite il camminatore casuale.

Si è quindi cercato di apportare delle modifiche alla matrice di adiacenza del grafo rappresentante l'overlay Chord in esame in modo da rispettare le proprietà richieste dalla Google Matrix; ottenuto questo risultato si è valutato l'andamento degli autovalori sia in caso di overlay non soggetto ad attacco che in caso di presenza di *Eclipse Attack*, con lo scopo di derivare osservazioni sulle modifiche strutturali che si vengono a formare. L'obiettivo da raggiungere è quello di capire se, in presenza di attacco, i nodi maligni sono identificabili grazie al monitoraggio della probabilità che hanno di essere visitati, fornita dall'analisi della Google Matrix.
Come detto in precedenza, il punto di partenza per il postprocessing del grafo è un file, output del simulatore, contenente per ogni nodo dell'overlay la sua finger table; questo risultato, ottenuto molto semplicemente andando a stampare, all'istante di Leave di un certo nodo, la sua finger table, permette l'analisi sia in caso di rete statica (nella quale le Leave dei nodi avvengono contemporaneamente alla fine dell'intera simulazione) che in caso di rete dinamica, ed è di base nella costruzione della matrice di adiacenza.
Infatti, tramite queste entry, è stato costruito un file contenente gli archi del grafo formati da un nodo sorgente dell'arco stesso, ovvero il responsabile della finger table, ed un nodo destinazione, ovvero il nodo di riferimento della entry considerata e tramite questi archi è stato possibile realizzare la matrice di adiacenza, fondamentale per l'analisi del grafo.

In accordo con le proprietà elencate precedentemente, si sono dovute apportare modifiche alla matrice di adiacenza in modo da ottenere una struttura a Google Matrix. Si sono allora considerate le tre ipotesi strutturali separatamente e si è cercato un meccanismo di modifica per farle rispettare.

Per quanto riguarda la *Stocasticità*, la si è ottenuta semplicemente dividendo ciascun elemento per il grado della riga di appartenenza, scalando così tutti gli elementi

della matrice di adiacenza in modo da garantire la somma per riga pari ad 1. In questo modo la matrice ottenuta non sarà più binaria ma avrà la garanzia di essere stocastica.

Per quanto riguarda l'*Aperiodicità*, la si è verificata tramite il controllo degli autovalori. Innanzitutto, osservando la matrice di adiacenza riscalata per garantire la stocasticità, si è osservato che il processo che si viene a formare è non reversibile e quindi gli autovalori che si ottengono sono sicuramente complessi. Dalla definizione di periodicità si ha che, se si riesce a verificare che la catena ha un unico autovalore di modulo unitario, essa sarà aperiodica [25]. Questo autovalore è fondamentale ed è anche noto in letteratura con il nome di *autovalore di Perron-Frobenius* e quindi, per garantire l'aperiodicità, deve essere l'unico con modulo unitario. Tramite il postprocessing delle matrici di adiacenza di grafi risultanti da alcune simulazioni effettuate, si è potuto verificare l'aperiodicità in quanto in ogni caso esiste un unico autovalore con modulo pari ad 1.

Per quanto riguarda l'*Irriducibilità*, si è notato che, in caso di *Eclipse Attack*, il grafo è caratterizzato dalla presenza di un sottoinsieme di stati assorbenti, ovvero i nodi maligni, i quali, a causa della modifica delle loro funzionalità di routing, hanno esclusivamente entry maligne all'interno delle loro finger table. Allora la matrice di adiacenza non è irriducibile e bisogna quindi applicare la modifica accennata precedentemente per renderla tale.
In realtà il meccanismo adottato per ottenere l'irriducibilità potrebbe essere un accorgimento utilizzato dagli stessi nodi maligni per non farsi scoprire. Infatti, realizzando ad hoc un semplice camminatore casuale, partendo da un nodo e muovendosi casualmente verso un vicino iterativamente fino a tornare al nodo di partenza, si è verificato che, nel momento in cui esso transiterà attraverso un nodo maligno, non tornerà più al nodo di partenza ma verrà bloccato; in questo modo verrà scoperta molto semplicemente la presenza dell'attacco in quanto, come detto, i *malicious node* hanno entry esclusivamente maligne e quindi archi solamente verso altri nodi maligni e di conseguenza, transitando attraverso uno di questi nodi, si entrerà in uno stato assorbente dal quale il camminatore casuale non uscirà più.
Il meccanismo utilizzato per imporre l'irriducibilità, allora, è in realtà il meccanismo utilizzato dai nodi maligni per non far scoprire, tramite un semplice camminatore casuale, la presenza dell'attacco: nel momento in cui un nodo maligno riceverà il camminatore casuale esso lo invierà ad un qualsiasi nodo nell'overlay; così facendo si otterrà una matrice che, anche a seguito dell'attacco, sarà difficilmente distinguibile dalla matrice standard e quindi si renderà impossibile l'individuazione dell'attacco tramite il metodo del camminatore casuale; per scoprire la presenza di un attacco in atto, allora, il metodo migliore e meno dispendioso è quello, riportato precedentemente, dell'analisi della ripartizione delle chiavi ai singoli nodi.

Avendo apportato le modifiche richieste ed avendo verificato che le tre proprietà strutturali della Google Matrix vengono in questo modo rispettate, si è finalmente

potuto effettuare il calcolo degli autovalori alla matrice modificata per effettuare uno studio dell'effetto dell'*Eclipse Attack* sulla struttura del grafo.
Si è allora considerato un overlay composto da $N = 100$ nodi e si sono effettuate differenti simulazioni:

- overlay Chord standard, senza presenza di nodi maligni
- overlay statico affetto da attacco Eclipse
- overlay dinamico affetto da attacco Eclipse, osservato al variare del numero di Join e Leave e cioè, come dimostrato in precedenza, al variare del grado di efficacia dell'attacco.

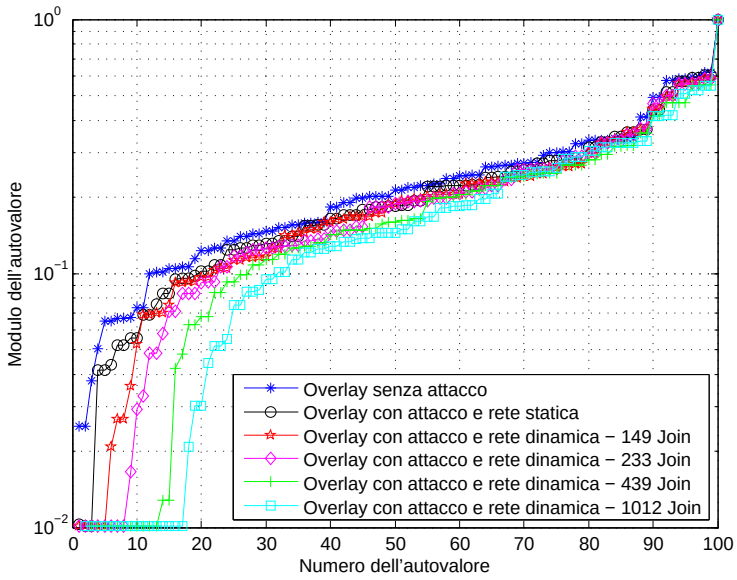


Figura 6.8: Modulo degli autovalori della *Google Matrix* per diversi scenari di attacco ed in caso di rete statica e dinamica; overlay formato da $N = 100$ nodi

Il grafico in Figura 6.8 mostra l'andamento del modulo degli autovalori della Google Matrix ottenuta al variare della tipologia di simulazioni prima descritte; da un'analisi sommaria dei grafici si nota innanzitutto come ci sia un unico autovalore con modulo pari ad 1, ovvero l'autovalore di Perron-Frobenius, e questo dimostra l'aperiodicità delle matrici in tutte le simulazioni considerate.
Incentrando l'analisi sull'individuazione di una modifica strutturale in base all'andamento del modulo degli autovalori, però, non si riescono ad estrarre differenze significative tra i vari scenari analizzati; l'andamento, infatti, risulta essere poco discorde e solamente in caso di autovalori piccoli in modulo, i meno importanti per le analisi strutturali, si notano variazioni mentre per autovalori in modulo più prossimi ad 1,

legati alla caratterizzazione del grafo, non si notano particolari differenze.
Questo significa che le modifiche introdotte per rispettare le proprietà delle Google Matrix al grafo in esame, hanno provocato una modifica alle proprietà strutturali del grafo stesso ed hanno reso i risultati relativi alle differenti simulazioni molto simili tra di loro. Per questo motivo si può concludere che lo studio delle caratteristiche del grafo soggetto ad *Eclipse Attack* effettuato con il metodo della Google Matrix non risulta essere molto indicativo e non fornisce risultati interessanti dal punto di vista del riconoscimento dei nodi maligni in quanto non è molto utile basarsi sulla stima delle probabilità di visitare un nodo.

La Matrice Laplaciana

A causa dell'inefficienza del meccanismo della Google Matrix è stata quindi necessaria l'analisi alternativa tramite lo studio della Matrice Laplaciana. Questo tipo di valutazione si basa sull'applicazione degli studi sugli autovalori della matrice laplaciana [26] ai grafi rappresentanti gli overlay simulati.
In realtà queste considerazioni sono state realizzate per lo studio di grafi non orientati, mentre i grafi su cui ci si deve basare per la valutazione dell'attacco sono sicuramente orientati in quanto gli archi derivati dalle entry delle finger table sono direzionali. Quindi le conclusioni che si possono trarre da questo studio sono esclusivamente qualitative, poiché basate su di una teoria approssimativa, non esatta ma solamente simile; si è fatta questa scelta in quanto la teoria sui grafi orientati è ancora oggi in una fase preliminare poiché grafi di questo tipo portano alla nascita di matrici non simmetriche e di conseguenza di autovalori complessi, che presentano una difficile analisi. Per questo motivo, perciò, l'analisi seguente fornirà risultati non del tutto corretti anche se permetterà di stabilire una linea guida coerente.

Innanzitutto si deve definire il concetto di *laplaciano*; la matrice laplaciana L è stata calcolata direttamente dalla matrice di adiacenza introdotta in precedenza tramite la relazione

$$L = D - A$$

dove A rappresenta proprio la matrice di adiacenza mentre D è una matrice diagonale in cui l'elemento di una particolare riga corrisponde al grado della riga corrispondente nella matrice di adiacenza. In questo modo si ottiene una matrice che ha sulla diagonale principale i gradi mentre gli altri elementi possono assumere valore 0 o -1 a seconda della matrice di adiacenza.
Determinato il laplaciano lo studio si è concentrato sul calcolo dei suoi autovalori, e risultano essere di fondamentale importanza ai fini dell'analisi quelli con modulo più piccolo, ovvero:

- *l'autovalore nullo*, detto anche *autovalore banale*. Ogni matrice L avrà sicuramente almeno un autovalore nullo e la sua molteplicità rappresenterà il numero di sottografi disgiunti che compongono il grafo principale.

- *il secondo autovalore più piccolo*; è l'autovalore fondamentale per la valutazione delle prestazioni dell'attacco, in quanto il suo modulo fornisce una misura sulla connettività della rete; infatti più questo valore è elevato più il grafo presenterà un forte grado di connessione mentre più il modulo si avvicinerà a 0 più la connessione della rete sarà debole. Addirittura, nel caso in cui il modulo di questo particolare autovalore assume il valore 0, ovvero coincide con l'autovalore nullo, si potrà concludere che il grafo è partizionato in quanto saranno presenti almeno due sottografi distinti all'interno del grafo principale.

La parte simulativa relativa allo studio del grafo, quindi, si è incentrata sulla valutazione del parametro λ, ovvero del secondo autovalore più piccolo, in overlay di dimensione fissa pari ad $N = 100$ nodi. Naturalmente, per poter valutare la variazione di λ sono stati considerati scenari differenti ovvero:

- nessun attacco Eclipse, ovvero assenza dell'attaccante e dei nodi maligni collusi.
- attacco Eclipse con una percentuale $f = 20\%$ di nodi maligni all'interno di una rete statica, senza `Churn`.
- attacco Eclipse con una percentuale $f = 20\%$ di nodi maligni considerando una rete dinamica, ovvero un `LifetimeChurn` con numero di Join e di Leave nel sistema pari a 149, 233, 439 e 1012. In queste simulazioni, come anche in quelle precedenti relative alla valutazione delle prestazioni dell'attacco, si è considerato come parametro caratteristico il numero di Join e Leave verificatosi poiché in questo modo i risultati possono essere considerati indipendenti dal tempo di simulazione e dal tempo medio di vita dei nodi, in quanto la variabile determinante ai fini dell'efficacia dell'attacco è proprio l'incremento della frazione di Join e Leave all'interno dello scenario simulativo.

Scenario	λ
No attacco Eclipse	3,5346
`NoChurn`	1,6647
`LifetimeChurn` con 149 Join	1,2194
`LifetimeChurn` con 233 Join	1,0000
`LifetimeChurn` con 439 Join	0
`LifetimeChurn` con 1012 Join	0

Tabella 6.3: Secondo autovalore (λ) in differenti scenari realizzati su di un overlay di $N = 100$ nodi

La Tabella 6.3 mostra la variazione del valore di λ al variare degli scenari prima descritti. Si nota chiaramente come, in caso di overlay Chord standard, senza attacco, il valore di λ è decisamente elevato e questo significa che la rete ha un alto grado di connettività. In presenza di attacco Eclipse, invece, il valore di λ scenderà più

sensibilmente in caso di `LifetimeChurn` ed all'aumentare del numero di Join e di Leave; questo risultato conferma quindi ciò che è stato mostrato precedentemente, in particolare nel grafico in Figura 6.4, ovvero che l'*Eclipse Attack* risulta essere più efficiente in caso di overlay dinamico ed il suo effetto crescerà all'aumentare del numero di Join e di Leave in quanto le finger table verranno inquinate maggiormente; al limite, infatti, questo inquinamento potrà portare l'attaccante a partizionare la rete, come in caso di `LifetimeChurn` con rate di Join e Leave pari a 439 e 1012, eclissando così una parte di nodi corretti dalla vista degli altri, obiettivo finale dell'*Eclipse Attack*.

Capitolo 7

Individuazione dell'attacco Eclipse

In questo capitolo sono riportati i risultati degli studi effettuati sulla struttura del grafo con lo scopo di riuscire a realizzare un meccanismo per l'individuazione dell'*Eclipse Attack*. Questo obiettivo è indispensabile in quanto le considerazioni a cui si è giunti nel capitolo precedente necessitano la conoscenza di tutti i parametri della rete o comunque, nel caso di ripartizione delle chiavi ai singoli nodi, non è stato presentato un meccanismo 'standard' per identificare la presenza dell'attacco.
È stato quindi necessario realizzare uno studio matematico per riuscire a modellare lo scenario in esame, in modo da definire delle procedure per calcolare i parametri fondamentali per la caratterizzazione della rete; inoltre sono stati formalizzati un modello per la descrizione dell'overlay ed un algoritmo per l'individuazione dell'attacco, in modo da poter avere un riferimento relativo alle modifiche strutturali imposte dall'*Eclipse Attack* al sistema.
Le definizioni dei modelli, naturalmente, devono essere supportate da verifiche sull'effettiva correttezza e sono state quindi effettuate delle fasi di verifica tramite il simulatore per dimostrare l'effettivo legame tra i modelli realizzati e la realtà; in questa fase ci si è riferiti ai parametri, già introdotti nel capitolo precedente, rappresentanti il volume di chiavi gestito da ogni singolo nodo in quanto, grazie al loro andamento, si sono potute verificare relazioni matematiche di base per la rappresentazione del modello.

Gli studi riportati in questo capitolo sono incentrati sul fatto che, l'identificazione della presenza di un *Eclipse Attack* all'interno di una rete, può essere basata su di un'osservazione già largamente discussa nel capitolo precedente: in reti oneste e con traffico omogeneo, in cui vale il paradigma del consistent hashing, il numero di chiavi di cui un nodo è responsabile è circa pari al numero di ricerche effettuate dal nodo stesso; accanto a questa relazione se ne è poi considerata un'altra che è stata fondamentale per l'individuazione dell'attacco: sempre considerando lo stesso scenario di riferimento, il numero di richieste in transito sarà legato al numero di richieste uscenti ed entranti secondo una particolare legge.
È quindi l'andamento delle chiavi in transito il parametro determinante per rivelare

la presenza dell'*Eclipse Attack*, e si è quindi cercato di derivare una legge di valenza generale per la rappresentazione delle chiavi in transito, basando la descrizione sul numero di hop, ovvero sul tragitto delle ricerche.

Successivamente si è incentrata l'attenzione su overlay soggetti ad attacco e sono state osservate le modifiche e le alterazioni del modello rispetto al comportamento dello scenario senza nodi maligni. Analizzando le differenze si è infine proposto un test, un algoritmo, per identificare l'attacco Eclipse e ricavare le probabilità di classificazione errata partendo, come già accennato in precedenza, da informazioni locali ottenute misurando il volume di chiavi transitanti.

7.1 Definizione del modello matematico

Come detto, il modello matematico che è stato formalizzato ha lo scopo di permettere la rappresentazione dell'overlay sia nel caso classico che in caso di presenza di *Eclipse Attack*, in modo da poter osservare con più semplicità ed immediatezza le modifiche strutturali che l'attacco impone all'overlay.

Riuscire a realizzare un modello perfettamente dettagliato, però, risulta essere molto complesso se non quasi impossibile a causa del molteplice numero di variabili da considerare.
Ci si è allora basati su di uno scenario semplificato, in modo da rendere più semplice la formalizzazione; all'interno di questo scenario sono stati considerati:

- N nodi tutti equispaziati all'interno dell'anello e rappresentati da identificativi equidistanti
- finger table semplificate composte da entry equispaziate e con nodi responsabili dislocati in posizioni fisse pari a $N/2$, $N/4$, $N/8$, ... fino ad arrivare, in corrispondenza della prima entry, al nodo a distanza 1 dal possessore della finger table, ovvero il suo diretto successore.

A partire da questo scenario molto semplificato si è cercato di estrarre una relazione generale per descrivere l'andamento del numero di hop con i quali può essere soddisfatta una richiesta, parametro, come già detto in precedenza, fondamentale per la gestione dell'andamento delle ricerche all'interno dell'overlay.

7.1.1 Overlay Chord senza attacco

Naturalmente, prima di considerare l'overlay soggetto ad attacco, si è deciso di considerare il caso classico, ovvero un overlay Chord standard [9, 10] senza nodi maligni; in questo modo è stato poi più semplice rappresentare l'effetto dell'*Eclipse Attack* all'overlay tramite modifiche mirate al modello classico.
Per semplificare l'analisi si è deciso di considerare un overlay di esempio con un numero di nodi finito, per poi estendere il tutto al caso generico di N nodi. In Figura 7.1,

infatti, è rappresentato l'overlay formato da $N = 16$ nodi nel quale è stata considerata l'operazione di ricerca a partire dal nodo 0 verso un qualsiasi altro nodo all'interno dell'anello. In questo modo, analizzando l'andamento del numero di hop al variare del nodo responsabile, si è potuta ricercare una legge generica per la sua descrizione, ovvero una legge per rappresentare la variazione del numero di hop necessari alla ricerca al variare del nodo destinazione della ricerca stessa, ovvero del responsabile della chiave ricercata.
Nell'esempio di Figura 7.1 è stato rappresentato lo scenario più probabile, ovvero quello in cui la chiave è memorizzata nel nodo più lontano possibile dal nodo di partenza, in questo caso il nodo 15; in questo specifico scenario la ricerca necessita il massimo numero di hop ed è da considerarsi il caso più probabile in quanto ad ogni 'salto' il responsabile contattato è sempre quello che controlla l'arco più esteso dell'anello, ovvero controlla una frazione di overlay più ampia rispetto alle altre entry della finger table del nodo sorgente, ed è quindi statisticamente il più probabile destinatario di ogni hop di ricerca.

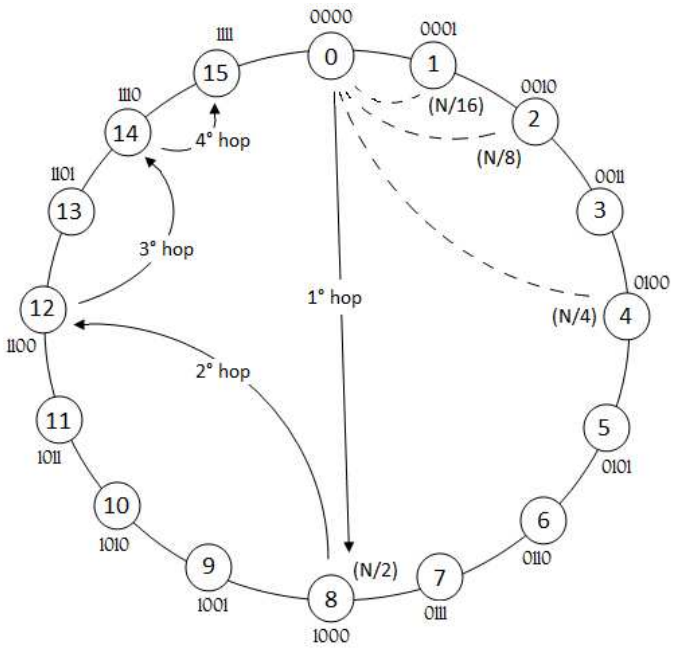


Figura 7.1: Esempio di ricerca in un overlay Chord non soggetto ad *Eclipse Attack*

Per questo motivo, al primo hop a partire dal nodo 0, la ricerca avrà probabilità $1/2$ di raggiungere il nodo 8, che infatti è il responsabile di una frazione pari ad $N/2$ dell'anello, e probabilità sempre decrescenti di raggiungere gli altri elementi della finger table rappresentati dai nodi 4, 2 e 1, ovvero il successore.
Al termine del primo salto l'operazione viene poi ripetuta, come mostrato sempre in Figura 7.1, a partire dal nodo 8, anche se in questo caso il valore di N di riferimento non sarà più 16 ma sarà $16/2 = 8$, in quanto, rispetto alla ricerca iniziata dal nodo 0,

il nodo 8 gestisce esclusivamente metà anello; in questo modo, iterando il processo di ricerca secondo il meccanismo già analizzato nel capitolo relativo a Chord, si ridurrà sempre più l'intervallo di pertinenza fino a raggiungere il nodo responsabile.
Nel caso in esame saranno necessari 4 hop per raggiungere il nodo 15; questo è il massimo numero di hop possibile all'interno di un overlay composto da $N = 16$ nodi e non è un valore casuale ma rispecchia la teoria sul protocollo Chord già descritta in precedenza, per la quale il massimo numero di hop necessari per raggiungere una destinazione in un overlay di N nodi sarà pari ad $O(\log_2 N)$.

Ripetendo l'operazione di ricerca al variare del nodo destinazione, del responsabile della chiave, si è potuta stilare la Tabella 7.1 che rappresenta la distanza delle chiavi dal nodo di riferimento che, nel caso in esame, è quello con identificativo pari a 0.

Numero di chiavi	Hop necessari
1	0
4	1
6	2
4	3
1	4

Tabella 7.1: Distanza delle chiavi dal nodo di riferimento, ovvero il nodo con ID pari a 0000

Osservando il modello di Figura 7.1 ed analizzando i dati di Tabella 7.1 riferendosi ad un nodo considerando il suo identificativo binario, che nel caso di overlay di $N = 16$ nodi è di $m = 4$ bit in quanto $2^4 = 16$, si è potuto osservare che, per esempio, l'unica ricerca che necessita 4 hop per essere portata a termine è quella verso il nodo con identificativo 1111, ovvero quello posto ad una *distanza di Hamming* pari a 4 rispetto al nodo di partenza 0000.

Da questa semplice osservazione, riscontrata da altre prove effettuate sull'overlay al variare del nodo di destinazione e confutata in overlay di dimensioni differenti, si è potuta formalizzare una legge:
il numero di hop necessari per la ricerca di una chiave all'interno di un overlay Chord non soggetto ad attacco Eclipse, è pari alla distanza di Hamming tra il nodo responsabile della chiave stessa ed il nodo di partenza per la ricerca, ovvero

$$\mathrm{P}(x \text{ hop}) = \text{numero di parole a distanza } x \text{ dal nodo autore della ricerca}$$

Come già introdotto precedentemente, lo studio della ripartizione delle chiavi tra i vari nodi dell'overlay è fondamentale per riuscire a descriverne forma e caratteristiche. Nel capitolo precedente l'attenzione è stata incentrata sul rapporto K_m/R_e senza considerare le chiavi in transito; questo terzo parametro, o meglio il rapporto K_t/R_e per rendere indipendente l'analisi dalla durata della simulazione, in cui K_t rappresenta proprio le `chiavi in transito`, è fondamentale in quanto permette uno studio, come già accennato in precedenza, più robusto ed efficace.

Si è voluto quindi studiare l'andamento di K_t/R_e all'interno di un overlay Chord standard, senza *Eclipse Attack* in atto; considerando l'esempio rappresentato in Figura 7.1, in cui si può mostrare con semplicità come il numero medio di hop sia pari a 2, è necessario riuscire ad analizzare il quantitativo di chiavi che ogni nodo vede transitare, ovvero si vuole studiare l'andamento del parametro K_t al variare del numero medio di hop; Naturalmente questo valore sarà tanto maggiore tanto più alto sarà il numero medio di hop dell'overlay considerato e quindi, per generalizzare questo studio, bisognerà poi riferirsi ad un caso generico caratterizzato da un numero medio di hop pari ad h.
Considerando $h = 2$, ovvero l'esempio riferito alla Figura 7.1, si può derivare con facilità che, in media, il numero di chiavi in transito relative ad una singola ricerca sarà pari sia alle chiavi uscenti che a quelle entranti, ovvero pari ad 1, in quanto in 2 hop una chiave, che è uscente per un solo nodo e memorizzata solo in un altro, transiterà esclusivamente una volta attraverso un terzo.
Generalizzando lo studio si ha quindi che, considerando un overlay con un numero medio di hop pari ad h, per ogni ricerca si avrà un numero di chiavi in transito circa pari ad $h - 1$ ovvero, all'interno di un overlay Chord standard, supponendo un tasso di ricerca pari a λ, si avranno in transito circa $\lambda \cdot (h - 1)$ chiavi e quindi

$$K_t/R_e \simeq h - 1$$

Il parametro di studio, quindi, è fortemente legato al numero medio di hop dello scenario considerato, il cui valore è calcolabile, in modo molto semplice, dalla relazione $h = np$, in cui n è direttamente legato al numero di nodi, ovvero in uno scenario in cui ho N nodi si avrà che $n = \log_2(N)$, e p sarà invece sempre pari ad $1/2$, in quanto va considerato l'assegnamento di un identificativo di n bit ad ogni nodo, per poter poi analizzare la distanza di Hamming, ed i bit che compongono l'ID saranno sempre tutti equiprobabili. Per esempio, considerando ancora lo scenario rappresentato in Figura 7.1 in cui $N = 16$, si avrà che n sarà pari a 4 e quindi, come già riportato in precedenza, il numero medio di hop sarà $h = 4 \cdot 1/2 = 2$.

7.1.2 Overlay Chord soggetto ad attacco Eclipse

In caso di *Eclipse Attack* [16–18, 22–24], invece, il modello caratterizzante l'overlay cambierà in quanto si ha che ogni nodo corretto, al momento dell'instradamento della richiesta, potrà contattare con probabilità $1 - f$ un altro nodo corretto e con probabilità f un nodo maligno; f, come già descritto nei precedenti capitoli, rappresenta la frazione di entry della finger table che hanno come nodo di riferimento un nodo maligno e questo valore non sarà costante ma cambierà al variare del numero di Join e di Leave, in quanto all'aumentare di ingressi ed uscite di nodi dal sistema crescerà l'inquinamento delle finger table.
Di conseguenza, rispetto al modello di Figura 7.1, nel caso di overlay corrotto, nel momento in cui la chiave viene instradata ad un nodo maligno si avrà la terminazione

della ricerca in quanto il nodo in questione si fingerà responsabile della chiave e la catturerà, bloccando così il processo di instradamento verso il reale responsabile.

Quindi, in caso di overlay soggetto ad attacco Eclipse, la ricerca procederà secondo il meccanismo analizzato precedentemente esclusivamente se ad ogni hop il responsabile appartiene alla frazione $1 - f$ di entry della finger table, ovvero quella delle entry corrette. Nel momento in cui interverrà un nodo maligno, invece, esso catturerà la chiave e di conseguenza bloccherà l'instradamento fingendosi il corretto responsabile. Allora l'andamento del numero di hop non sarà più pari alla distanza di Hamming tra il vero responsabile della chiave e l'autore della ricerca ma sarà sicuramente minore, escludendo naturalmente il caso in cui la richiesta evita entry maligne, nel quale si otterrà una situazione identica al caso di overlay senza attacco.

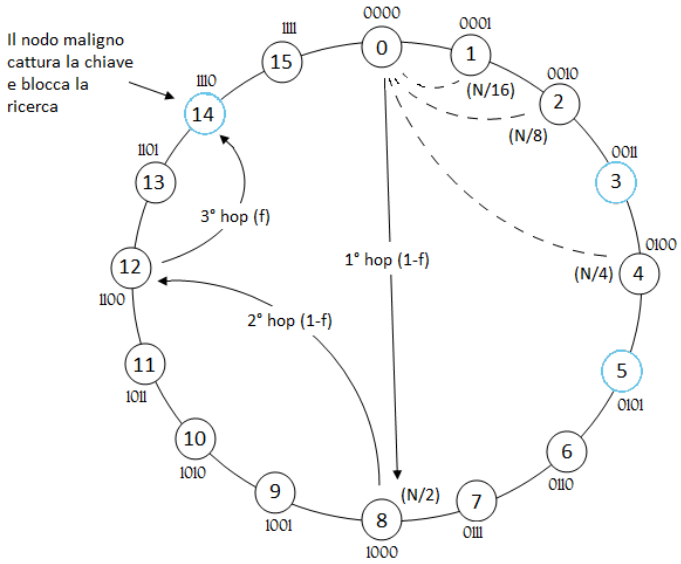


Figura 7.2: Esempio di ricerca in un overlay Chord in cui è in atto un *Eclipse Attack* con percentuale f di entry maligne nelle finger table

In Figura 7.2 è rappresentato un esempio di ricerca simile a quello mostrato nel caso di overlay standard dove però, al penultimo hop del percorso reale, che dovrebbe raggiungere il nodo 15, avviene l'instradamento ad un nodo maligno il quale blocca la ricerca e cattura la chiave. Quindi, in questo particolare caso, la ricerca si fermerà un hop prima rispetto a quanto dovrebbe e quindi il numero di hop sarà pari alla distanza di Hamming ridotta di 1; in generale però, siccome le entry maligne possono essere selezionate ad un qualsiasi hop della ricerca, si ha che l'andamento del numero di hop sarà legato sia alla distanza di Hamming (non potrà mai superare questo valore) che anche alla frazione di entry maligne nell'overlay in quanto all'aumentare di f si avrà una probabilità maggiore di selezionare prima una entry maligna, e quindi di interrompere la ricerca ad un numero di hop inferiore.
Per concludere la caratterizzazione del modello si è anche stimata la probabilità di in-

stradamento corretto in scenari soggetti ad attacco Eclipse: se una ricerca, all'interno di un overlay standard, necessita di k hop per arrivare alla corretta destinazione, nel caso di rete soggetta ad attacco la stessa ricerca raggiungerà il corretto responsabile con probabilità

$$\overbrace{(1-f)\cdot(1-f)\cdot\ldots\cdot(1-f)}^{k \text{ volte}}$$

in quanto, se per k volte verrà selezionata una entry corretta della finger table, si raggiungerà sicuramente il reale responsabile. In caso di instradamento ad un nodo maligno ovvero di moltiplicazione per f, invece, la ricerca terminerà immediatamente riducendo così il numero di hop necessari.

Cosa succede, in caso di presenza *Eclipse Attack*, alle chiavi transitanti?
Come già spiegato in precedenza, in presenza di attacco, se una chiave dovesse essere inoltrata ad un nodo maligno esso si fingerebbe il reale responsabile e la catturerebbe, bloccando, di conseguenza, il suo corretto percorso verso il reale responsabile. L'effetto dell'*Eclipse Attack* sull'instradamento, quindi, sarà una netta diminuzione del numero di hop rispetto ad un overlay non soggetto ad attacco, e quindi una riduzione del parametro rappresentante le chiavi transitanti.
Di conseguenza il parametro K_t/R_e all'interno di overlay soggetti ad attacco Eclipse, diminuirà all'aumentare dell'efficacia dell'attacco stesso in quanto, più l'*Eclipse Attack* sarà ben radicato all'interno della rete, maggiore sarà la probabilità che una ricerca venga inoltrata ad un nodo maligno e, di conseguenza, bloccata.

7.2 Verifica sperimentale del modello

Dopo aver formalizzato il modello, l'obiettivo è diventato quello di verificare sperimentalmente la coerenza tra i risultati ottenuti dalle simulazioni effettuate e quello che è stato dedotto dall'applicazione del modello stesso ai casi in esame.
A questo scopo, come già accennato in precedenza, è necessario analizzare l'andamento delle chiavi in transito, ovvero il quantitativo di chiavi che raggiungono un particolare nodo ma vengono inoltrate in quanto non sono arrivate a destinazione, al variare della struttura dell'overlay e dell'efficacia dell'attacco. Si è deciso di utilizzare questo parametro, invece delle chiavi memorizzate o delle ricerche effettuate, in quanto permette un'analisi molto più precisa e robusta e perché il riferirsi al transito consente uno studio meno soggetto a variazioni, garantendo così risultati più attendibili.

La fase di verifica è stata allora caratterizzata dall'esecuzione di differenti simulazioni sull'overlay Chord, sia soggetto che non soggetto ad attacco Eclipse, implementato su OMNeT++/OverSim, con lo scopo di confermare l'effettiva valenza dei modelli prima introdotti per quanto riguarda l'andamento delle chiavi transitanti; lo scopo di questa fase è fondamentale in quanto è molto importante riuscire a dimostrare che le varie stime analizzate hanno effettivamente un riscontro reale.

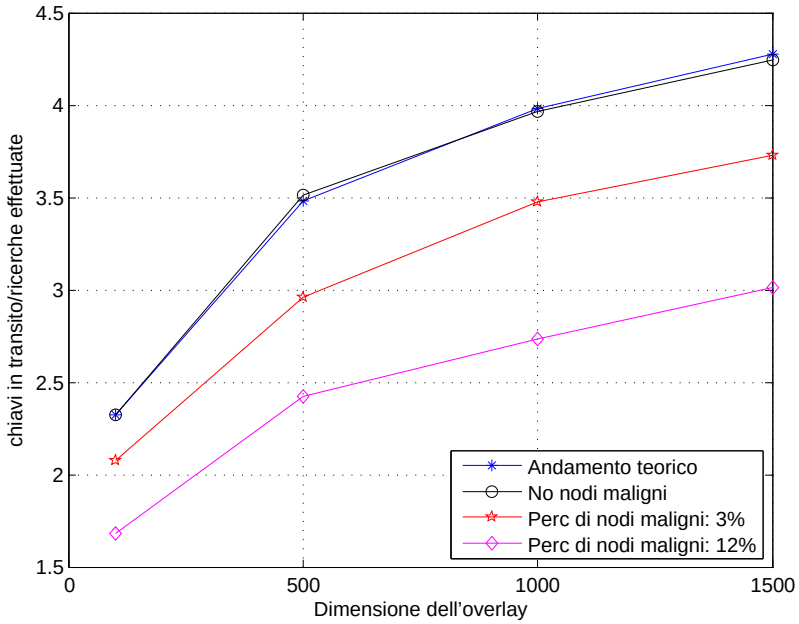


Figura 7.3: Andamento del valore medio del parametro K_t/R_e al variare della dimensione dell'overlay e del numero di nodi maligni

Il grafico in Figura 7.3 mostra l'andamento del valore medio di K_t/R_e al variare della dimensione dell'overlay, assunta pari ad $N = 100$ nodi, $N = 500$ nodi, $N = 1000$ nodi ed $N = 1500$ nodi, considerando scenari differenti. Infatti, come è mostrato in legenda, sono state eseguite differenti simulazioni al variare della percentuale di nodi maligni nell'overlay; in questo modo si è potuto facilmente osservare come, all'aumentare del numero di nodi maligni, il numero di chiavi in transito diminuisca proprio perché si ha probabilità maggiore di instradare una richiesta ad un nodo maligno, il quale catturerà la chiave.
Per quanto riguarda la simulazione effettuata in assenza di attacco si nota come il rapporto medio K_t/R_e è circa pari ad $h-1$, ovvero l'andamento teorico del parametro stimato in precedenza, e che nella rappresentazione si sovrappone quasi perfettamente alla curva relativa all'assenza di nodi maligni; questo significa che l'analisi pratica conferma ciò che era stato ipotizzato, ovvero dimostra che la relazione estratta in precedenza per la rappresentazione dell'andamento delle chiavi in transito al variare delle caratteristiche della rete descrive il comportamento che è realmente osservabile nella realtà. Per quanto riguarda le simulazioni effettuate in presenza di nodi maligni, invece, sempre dal grafico in Figura 7.3 si nota con chiarezza come, all'aumentare della percentuale di *malicious node* nel sistema, decresce il valore del parametro in esame in quanto aumenta la probabilità che una ricerca venga inoltrata ad un nodo maligno e quindi bloccata.

In questo modo, allora, un singolo nodo potrà capire se vi è in atto un *Eclipse Attack* o meno grazie alla semplice osservazione del parametro K_t/R_e ed al confronto di quest'ultimo con la relazione che lo descrive introdotta in precedenza; nel caso in cui

$$K_t/R_e \ll h - 1$$

il nodo considerato potrà stabilire di essere all'interno di un overlay in cui è in atto un attacco Eclipse, in quanto il valore osservato è sensibilmente inferiore rispetto al suo andamento caratteristico.
Il problema di questa osservazione è quello di riuscire a quantificare il concetto di $\ll$ ovvero, dopo aver stabilito la correttezza di questa relazione è necessario definire un algoritmo standard per permettere l'identificazione dell'*Eclipse Attack* all'interno di un overlay, ovvero un test per formalizzare la fase di rivelazione dell'attacco e le probabilità di errata classificazione.

7.3 Algoritmo di identificazione dell'attacco

La definizione dell'andamento del rapporto K_t/R_e all'interno di overlay non soggetti ad *Eclipse Attack* verificato in precedenza, non fornisce un algoritmo per l'identificazione dell'attacco stesso in quanto è stato si mostrato che in caso di presenza di attacco K_t/R_e decresce, ma è sicuramente necessario uno studio più approfondito per formalizzare l'identificazione. È stato allora realizzato, come già accennato precedentemente, un test per l'identificazione dell'attacco Eclipse all'interno di un overlay Chord; lo scopo di questo algoritmo è quello di fornire il grado di veridicità con il quale un qualsiasi nodo, esclusivamente tramite informazioni locali in suo possesso, può riuscire a rivelare la presenza di attacco all'interno dell'overlay che lo ospita.

Per questo tipo di analisi ci si è incentrati su di un overlay Chord formato da $N = 100$ nodi e sono state considerate le simulazioni effettuate in assenza di nodi maligni ed in presenza di 3 nodi collusi, ovvero gli esempi già esaminati in precedenza e per i quali l'andamento del rapporto K_t/R_e è stato rappresentato nel grafico in Figura 7.3. Sono stati considerati questi due scenari e non l'esempio caratterizzato da 12 nodi collusi in quanto il riuscire ad identificare l'attacco lanciato da 3 nodi garantisce sicuramente l'identificazione di attacchi caratterizzati da una percentuale di nodi maligni maggiore. In entrambe le simulazioni sono state misurate esclusivamente informazioni locali in possesso di ogni singolo nodo ovvero i parametri, già introdotti in precedenza, R_e, K_t e K_m. Questi parametri sono fondamentali poiché possono essere raccolti dal singolo nodo in un qualsiasi istante e senza un elevato costo computazionale; naturalmente, ai fini dell'algoritmo di identificazione, sono stati considerati esclusivamente R_e e K_t.

Per utilizzare il più appropriato test statistico ci si è basati su [27] che, concentrandosi sul testing delle ipotesi, fornisce dei metodi generali da applicare ai comuni problemi statistici per riuscire a classificare dei campioni ed a stabilire poi le varie

probabilità d'errore.
Nello specifico, il caso in esame riguarda un semplice classificatore binario ed è stato quindi necessario un test di ipotesi per due classi che, naturalmente, sono l'assenza e la presenza di *Eclipse Attack*.
Il test che è stato applicato è concettualmente molto semplice.
Le informazioni di partenza, ricavate dalle simulazioni effettuate nei due casi presi in esame, sono media e varianza di K_t/R_e, in quanto grazie ad esse è stato possibile rappresentare l'andamento dei rapporti con delle distribuzioni gaussiane. Come riportato anche nel grafico in Figura 7.3 la media vale 2,3215 in caso di overlay non soggetto ad attacco e 2,0781 per overlay soggetto ad *Eclipse Attack*; per quanto riguarda la varianza, invece, essa vale 0,018 in caso di overlay non soggetto ad attacco e 0,0072 in caso di presenza di *Eclipse Attack*. Quindi, siccome il rapporto medio K_t/R_e in caso di assenza di attacco è superiore rispetto allo stesso rapporto osservato all'interno di overlay soggetti ad *Eclipse Attack*, è stata fissata una soglia *Th* sul rapporto stesso in modo tale che:

- se $K_t/R_e > Th$ si sarà in presenza di un overlay Chord non soggetto ad attacco Eclipse
- se $K_t/R_e < Th$ si sarà in presenza di un overlay Chord soggetto ad *Eclipse Attack*

Naturalmente il punto chiave del test è la scelta della soglia *Th* in quanto al suo variare cambierà l'accuratezza della decisione.
Infatti, considerando la rappresentazione gaussiana delle due simulazioni ed un valore di soglia, nascono quattro differenti aree, due delle quali assumono fondamentale importanza nell'analisi della correttezza della classificazione e sono rappresentate in Figura 7.4.

- *FPF* (*False Positive Fraction*); è rappresentata dall'area sottesa alla gaussiana, relativa alla simulazione in presenza di *Eclipse Attack*, per valori maggiori della soglia *Th*. È un falso positivo in quanto, siccome si stanno considerando valori maggiori di *Th*, il classificatore li considera non soggetti ad attacco anche se in realtà l'attacco è presente.
- *FNF* (*False Negative Fraction*); è rappresentata dall'area sottesa alla gaussiana, relativa alla simulazione non soggetta ad *Eclipse Attack*, per valori inferiori alla soglia *Th*. È un falso negativo in quanto, siccome si stanno considerando valori inferiori a *Th*, il classificatore li considera soggetti ad attacco anche se in realtà l'attacco non è presente.

Al variare della soglia *Th* cambieranno i valori delle frazioni di falso positivo e falso negativo, ovvero cambierà l'accuratezza del classificatore nel commettere errori.

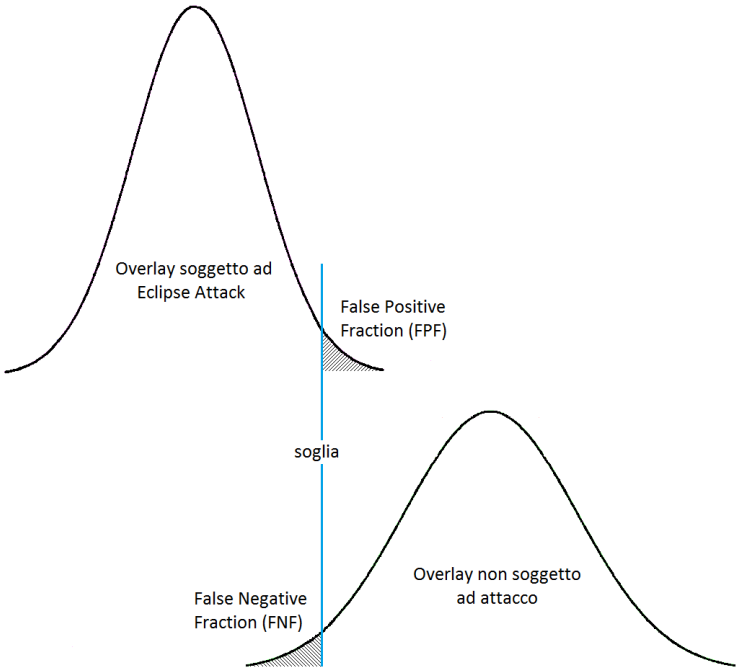


Figura 7.4: Visualizzazione grafica di *False Positive Fraction* e *False Negative Fraction* per distribuzioni gaussiane

Per valutare queste due aree negli scenari considerati è stato necessario introdurre un vincolo, ovvero si è dovuto fissare un range di valori che una delle due frazioni deve rispettare e di conseguenza andare a determinare la soglia corrispondente e l'estensione dell'altra frazione.

Soglia	*FPF*	*FNF*
2,115	$1{,}488 \cdot 10^{-7}$	$1{,}034 \cdot 10^{-30}$
2,12	$2{,}952 \cdot 10^{-9}$	$2{,}461 \cdot 10^{-29}$
2,125	$3{,}661 \cdot 10^{-11}$	$5{,}421 \cdot 10^{-28}$
2,13	$2{,}832 \cdot 10^{-13}$	$1{,}107 \cdot 10^{-26}$
2,135	$1{,}332 \cdot 10^{-15}$	$2{,}092 \cdot 10^{-25}$

Tabella 7.2: Andamento di *FPF* e *FNF* al variare della soglia t per overlay Chord di $N = 100$ nodi in caso di assenza e di presenza di *Eclipse Attack*

Si è deciso, per la valutazione dell'accuratezza della classificazione, di accettare un valore di falso positivo circa pari a 10^{-9}; è stata fissata questa frazione e non il falso negativo in quanto, siccome è più grave non rivelare un attacco presente, si è deciso di imporre molto bassa la probabilità che questo accada, lasciando libera la probabilità corrispondente. Osservando la Tabella 7.2, dove sono rappresentati alcuni

valori delle frazioni di falso positivo e falso negativo al variare della soglia Th, si nota che, al valore ricercato, corrisponde una soglia $Th = 2{,}12$ che permette di ottenere una frazione di falso negativo circa pari a 10^{-29}.
Da questa analisi si può quindi concludere che la classificazione effettuata, per la quale

- se $K_t/R_e > 2{,}12$ l'overlay non è soggetto ad attacco
- se $K_t/R_e < 2{,}12$ l'overlay è soggetto ad *Eclipse Attack*

fornisce un livello di correttezza estremamente elevato con una probabilità di errore nella rivelazione dell'attacco molto bassa in quanto si ha:

- probabilità $2{,}952 \cdot 10^{-9}$ di non rivelare un attacco anche se è presente;
- probabilità $2{,}461 \cdot 10^{-29}$ di rivelare la presenza di attacco che invece non è presente;

Si può quindi concludere che il meccanismo di testing elaborato e la soglia scelta permettono una classificazione molto efficiente e consentono ad un nodo di identificare con alto grado di veridicità un *Eclipse Attack* basandosi esclusivamente sulla conoscenza delle ricerche da lui effettuate e delle chiavi in transito.

Capitolo 8

Conclusioni

In questo lavoro di tesi è stato analizzato l'attacco Eclipse al protocollo Chord, con l'obiettivo di riuscire a fornire delle linee guida per l'implementazione ed un algoritmo di test per l'individuazione dell'attacco utilizzando esclusivamente informazioni disponibili localmente.

Lo scopo dell'*Eclipse Attack* è quello di riuscire, tramite il controllo di un numero limitato di peer, a controllare una frazione elevata di chiavi del sistema.
È un'evoluzione del *Sybil Attack* nel quale l'attaccante, controllando una frazione f di nodi del sistema, riesce ad entrare in possesso di una percentuale circa pari ad f di chiavi; considerando, quindi, questo attacco realizzato su reti *Chord-based* e modificando il comportamento dei nodi maligni in termini di instradamento e di ricerca, si è ottenuta l'implementazione dell'*Eclipse Attack* al protocollo Chord e sono state osservate differenti proprietà.

- Al crescere del numero di nodi maligni nel sistema l'andamento della frazione di chiavi controllate dall'attaccante non è più direttamente proporzionale alla frazione di nodi collusi, come nel caso dell'attacco Sybil, ma cresce molto rapidamente. Si è osservato che, per esempio, in un overlay composto da $N = 100$ nodi, controllando solamente 5 di essi l'attaccante riesce ad entrare in possesso di più dell'80% delle chiavi mentre, nel caso di 12 nodi collusi, l'attaccante riesce a controllare la totalità dell'overlay.

- Al crescere della dimensione dell'overlay l'efficacia dell'attacco aumenta sensibilmente. Infatti è stato osservato che, mentre in un overlay composto da $N = 15$ nodi se un attaccante controlla il 20% dei nodi del sistema riesce a catturare circa il 75% delle chiavi, nel caso in cui l'overlay è formato da $N = 1000$ nodi, la presenza del 20% di essi maligni e collusi garantisce all'attaccante il controllo completo della rete.

- In caso di rete dinamica l'efficienza dell'attacco aumenta, in quanto al crescere del rate di Join e di Leave si avrà un inquinamento maggiore delle finger table. Ad esempio è stato osservato che, in caso di overlay composto da $N = 100$

nodi, di cui 10 maligni, in caso di rete statica la percentuale di chiavi catturate dall'attaccante è minore del 90%, mentre in caso di rete dinamica la frazione di chiavi catturate è sensibilmente maggiore; inoltre l'efficienza di reti dinamiche cresce all'aumentare del rate di Join e Leave ed infatti, sempre considerando l'esempio di riferimento, nel caso di 233 Join la frazione di chiavi controllate è circa pari al 92%, mentre con 1024 Join supera il 95%.

Per quanto riguarda l'effetto dell'attacco sulla distribuzione delle chiavi ai singoli nodi, è stato osservato come, considerando un overlay soggetto ad *Eclipse Attack*, non viene più rispettato il principio del *consistent hashing*. Infatti, mentre in caso di overlay Chord standard, senza attacco, il rapporto medio `chiavi memorizzate/ricerche effettuate` si mantiene circa pari ad 1, in caso di presenza di un attaccante questo rapporto scende molto rapidamente e decresce all'aumentare della percentuale di nodi maligni collusi. Ad esempio, in un overlay composto da $N = 100$ nodi, di cui 3 maligni, il rapporto medio è circa pari a $5 \cdot 10^{-2}$, mentre in caso di 12 nodi maligni collusi vale circa $3 \cdot 10^{-3}$.

Infine, sempre per quanto riguarda il postprocessing della fase implementativa, è stata osservata la modifica topologica imposta dall'*Eclipse Attack* al grafo di rete. Il meccanismo 'standard' per questa analisi è quello della Google Matrix ma, nel caso in esame, non forniva riscontri significativi. Ci si è avvalsi allora di un meccanismo alternativo basato sull'analisi degli autovalori della matrice laplaciana, metodo base per analisi su grafi non orientati ma comunque utile per fornire delle linee guida anche in caso di grafi orientati come quello in esame. Analizzando gli autovalori si è notato come, in caso di attacco e di crescita del rate di Join e Leave, il grafo di rete assume una struttura sempre meno connessa. Nell'esempio riportato nel capitolo 6, relativo ad un overlay composto da $N = 100$ nodi, è mostrato l'andamento del secondo autovalore, λ, il cui valore è direttamente legato al grado di connettività della rete; è stato osservato come, già in caso di attacco con rate di Join e Leave pari a 439, il secondo autovalore assume valore 0, ovvero il grafo è suddiviso in sottografi. Questo mostra come l'*Eclipse Attack* porti ad una partizione della rete, permettendo all'attaccante di eclissare una parte dei nodi alla vista degli altri.

Per quanto riguarda l'individuazione dell'attacco sono stati proposti innanzitutto dei modelli matematici per la descrizione dell'overlay sia nel caso standard, senza attacco, che in caso di presenza di *Eclipse Attack*; tramite delle simulazioni sono stati verificati due importanti risultati.

- È stato studiato l'andamento del numero di hop necessari per portare a termine una ricerca. Si è verificato che questo valore, in caso di assenza di attacco, è pari alla distanza di Hamming tra il nodo sorgente ed il nodo destinazione della ricerca, mentre decresce in caso di presenza di *Eclipse Attack*.

- È stato studiato l'andamento delle chiavi in transito, parametro che fornisce un'analisi più efficace e precisa rispetto alle chiavi memorizzate e alle ricerche

effettuate. Si è dimostrata matematicamente e poi verificata sperimentalmente una relazione che lega il rapporto `chiavi in transito/ricerche effettuate` al numero medio di hop in caso di assenza di attacco Eclipse: $K_t/R_e \simeq h-1$; si è poi verificato come, in caso di presenza di attacco, questo parametro assume un valore inferiore, il quale inoltre decresce all'aumentare del numero di nodi maligni in quanto, al crescere della frazione f di nodi collusi, è più probabile che lungo il tragitto della ricerca la chiave venga catturata dall'attaccante. Ad esempio, nel caso di overlay composto da $N = 100$ nodi, si è osservato come il rapporto medio K_t/R_e, in caso di assenza di attacco, è circa pari a 2,3 che corrisponde, con un piccolo margine di errore, ad $h-1$; in caso di presenza di una frazione f nodi maligni, invece, questo valore decresce ed è circa pari a 2,1 per $f = 3\%$ ed 1,7 nel caso in cui $f = 12\%$.

Grazie a queste informazioni è stato costruito un semplice algoritmo di test tramite il quale un qualsiasi nodo, esclusivamente osservando le informazioni locali in suo possesso, può riuscire a capire se è in atto un *Eclipse Attack* all'interno della rete. Questo algoritmo, che è l'implementazione di un semplice classificatore, è stato testato su di un overlay di $N = 100$ nodi considerando i due scenari di assenza di attacco Eclipse e di presenza di 3 nodi maligni collusi; in riferimento a queste simulazioni è stata osservata un'ottima precisione nel classificare la tipologia dell'overlay considerato.

Per concludere è necessario analizzare i possibili sviluppi futuri legati a questo lavoro di tesi.
Il lavoro effettuato è stato incentrato innanzitutto su di un'indagine quantitativa per la valutazione dell'impatto dell'*Eclipse Attack* in reti di diverse dimensioni e successivamente sulla realizzazione di un test per l'identificazione dell'attacco, partendo esclusivamente da misure locali.

Potrebbe quindi essere proposto un meccanismo di difesa contro l'*Eclipse Attack* in modo da rendere più sicure le reti basate su paradigmi distribuiti.
Ad oggi in letteratura esistono differenti meccanismi di difesa contro l'attacco Eclipse, ma esclusivamente al prezzo di abbandonare il paradigma P2P, tramite l'inserimento di controllori centralizzati.

La sfida potrebbe quindi essere quella di riuscire ad implementare un meccanismo di difesa puramente distribuito per questo particolare tipo di attacco, magari basandosi, in una fase preliminare, sull'algoritmo di identificazione proposto in questo lavoro.

Elenco delle figure

Elenco delle tabelle

Acronimi principali

SIP	Session Initiation Protocol
P2P	Peer-to-peer
IETF	Internet Engineering Task Force
ITU-T	International Telecommunication Union - Telecommunication
RTP	Real Time Transport Protocol
UA	User Agent
HTTP	Hypertext Transfer Protocol
SDP	Session Description Protocol
URI	Uniform Resource Indicator
URL	Uniform Resource Locator
RELOAD	REsource LOcation And Discovery
P2PSIP	Peer-to-peer Session Initiation Protocol
NAT	Network Address Translation
ICE	Interactive Connectivity Establishment
API	Application Programming Interface
AOR	Address of Record
GRUU	Globally Routable UA URI
PKI	Public Key Infrastructure
TLS-PSK	Transport Layer Security-Pre Shared Key Ciphersuites
DOS	Denial of Service
DHT	Distributed Hash table
TLS	Transport Layer Security
SHA	Secure Hash Algorithm
RPC	Remote Procedure Call
CA	Trusted Certification Authority
CRT	Constrained Routing Table
PNS	Proximity Neighbor Selection
FPF	False Positive Fraction
FNF	False Negative Fraction

Bibliografia

[1] D. Collins. *Carrier Grade Voice Over IP*. McGraw-Hill Companies, September 2000.

[2] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, and et al. M. Handley, E. Schooler. SIP: Session Initiation Protocol. RFC 3261, Internet Engineering Task Force, 2002.

[3] ITU-T. Packet-based Multimedia Communications Systems. Recc H.323, International Telecommunication Union, 1998.

[4] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne. REsource LOcation And Discovery (RELOAD). Internet Draft `draft-bryan-p2psip-reload-04`, work in progress, July 2008.

[5] The P2PSIP documentation. http://www.p2psip.org/.

[6] D. Chopra, H. Schulzrinne, E. Marocco, and E. Ivov. Peer-to-peer Overlays for Real-Time Communication: Security Issues and Solutions. *Communications Surveys Tutorials, IEEE*, 11, January 2009.

[7] D. Bryan, P. Matthews, E. Shim, and D. Willis. Concepts and Terminology for Peer-to-Peer SIP. Internet Draft `draft-ietf-p2psip-concepts-01`, November 2007.

[8] Z. Tian, X. Wen, W. Zheng, Y. Sun, and Y. Cheng. Evaluation and Simulation on the Performance of DHTs Required by P2PSIP. *Information Assurance and Security, Fifth International Conference on*, August 2009.

[9] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[10] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. *Networking, IEEE/ACM Transactions on*, 11(1):17–32, February 2003.

[11] X. Zheng and V. Oleshchuk. Improving Chord Lookup Protocol for P2PSIP-Based Communication Systems. *New Trends in Information and Service Science, International Conference on*, June 2009.

[12] J. Maenpaa and G. Camarillo. Study on Maintenance Operations in a Chord-based Peer-to-peer Session Initiation Protocol Overlay Network. *Parallel Distributed Processing, IEEE International Symposium on*, May 2009.

[13] A. Varga and R. Hornig. An Overview of the OMNeT++ Simulation Environment. In *Simutools '08: Proceedings of the 1st International Conference on Simulation tools and techniques for Communications, Networks and Systems Workshops*, Brussels, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[14] I. Baumgart, B. Heep, and S. Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of 10th IEEE Global Internet Symposium in conjunction with IEEE INFOCOM 2007*, Anchorage, AK, USA, May 2007.

[15] OverSim: The Overlay Simulation Framework. http://www.oversim.org/.

[16] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-peer Overlay Networks. In *Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI 2002)*, Boston, MA, USA, December 2002.

[17] G. Urdaneta, G. Pierre, and M. Van Steen. A Survey of DHT Security Techniques. *ACM Computing Surveys*, 5, 2009.

[18] E. Sit and R. Morris. Security Considerations for Peer-to-peer Distributed Hash Tables. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, USA, March 2002.

[19] J. Douceur. The Sybil Attack. In *Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

[20] O. Jetter, J. Dinger, and H. Hartenstein. Quantitative Analysis of the Sybil Attack and Effective Sybil Resistance in Peer-to-peer Systems. *Communications (ICC), IEEE International Conference on*, May 2010.

[21] R. C. Merkle. Secure Communications over Insecure Channels. *Commun. ACM*, 21(4):294–299, 1978.

[22] M. Castro, M. Costa, and A. Rowstron. Performance and Dependability of Structured Peer-to-peer Overlays. Technical report, Microsoft Research, December 2003.

[23] A. Singh, T.W. Ngan, P. Druschel, and D.S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *Proc IEEE INFOCOM*, Barcelona, Spain, April 2006.

[24] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against Eclipse Attacks on Overlay Networks. In *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*, Leuven, Belgium, September 2004.

[25] T. H. Haveliwala and S. D. Kamvar. The Second Eigenvalue of the Google Matrix. Technical report, Stanford University, 2003.

[26] L. Lovász. Random Walks on Graphs: A Survey. In *Combinatorics, Paul Erdös is Eighty*, volume 2, pages 1–46. Janos Bolyai Mathematical Society, Keszthely, Hungary, 1993.

[27] A. M. Mood, F. A. Graybill, and D. C. Boes. *Introduction to the Theory of Statistics*. McGraw-Hill Companies, April 1974.

Ringraziamenti

Ed eccoci al termine.
Al termine del mio percorso scolastico; al termine di cinque bellissimi ma anche faticosissimi anni di università; al termine di quella che può essere definita 'parte lineare' della mia vita e che mi introdurrà (finalmente?) nel mondo del lavoro.

Soffermandomi solamente sull'anno trascorso a realizzare questa tesi di laurea, a cui ho dedicato veramente molto tempo e che mi ha entusiasmato, poi depresso, poi emozionato ed anche rallegrato, devo effettivamente spendere due parole e ringraziare alcune persone che sono state fondamentali per me sia durante questo periodo che durante tutto il mio corso di studi.

Un ringraziamento va sicuramente al professor *Giacomo Verticale*, il relatore che mi ha seguito instancabilmente ed ha sopportato tutte le mie richieste di chiarimento e le mie domande, magari a volte anche poco coerenti.
Devo anche riconoscenza al professor *Cesare Alippi*, relatore durante la laurea triennale ma soprattutto mentore durante tutti i cinque anni di università; per qualsiasi problema, chiarimento, consiglio mi sono sempre rivolto a lui e mi è sempre stato fornito sostegno ed aiuto.

Non posso poi dimenticare tutti i ragazzi del *Bonsai Lab* del *DEI*, con i quali ho trascorso intere giornate lavorando si individualmente ai singoli progetti ma, per sdrammatizzare, scambiandosi anche quattro chiacchere e qualche battuta! Direi tutto fondamentale perché in questo modo poi ci si concentra meglio... Devo quindi ringraziare *Francesca*, *Cristina*, *Vittorio*, *Marco*, *Daniele*, *Matteo* e soprattutto *Paolo* (grandissimo), per questi momenti trascorsi insieme.
In più un ringraziamento va anche indistintamente a tutti i dottorandi che, anche se non direttamente collegati al mio progetto, mi hanno sempre dato il loro aiuto, soprattutto riguardo alla scrittura in Latex.

Per quanto riguarda tutto il mio percorso scolastico, cinque anni che stanno per terminare e che sembrano essere iniziati un secolo fa, non posso citare singolarmente tutte le persone con cui ho avuto contatti poiché diventerebbe un elenco interminabile. Mi soffermo allora sul gruppo di compagni con cui ho avuto ed ho ancora un

rapporto speciale, perché è cresciuto nel tempo, si è rafforzato; so che su di loro potrò sempre contare, come anche loro potranno sempre contare su di me. Grazie *Dani*, *Marians*, *Paolo* e *Matta*; ricordo veramente con piacere le varie partite a briscola chiamata giocate durante pausa pranzo, le ore di studio trascorse insieme anche senza rendere al massimo e tutte le grigliate organizzate e sempre perfettamente riuscite. Mi sembra poi giusto nominare anche *Feo*, *Sciutosky*, i due *Matteo*, *Riccardo*, *Ema* e i due *Luca*.
Non ho citato volutamente una persona che merita di essere ringraziata singolarmente: grazie *Gian*, non potrò mai dimenticare tutti i momenti, sia seri che divertenti, che abbiamo trascorso insieme; a partire dalla tesi per la laurea triennale, le nottate di lavoro per riuscire a rispettare le scadenze, poi tutte le varie seratone passate in giro per Milano e gli infiniti aperitivi sempre in posti diversi ed infine l'indimenticabile estate in Irlanda, cercando un lavoro e conoscendo miliardi di persone in giro ed all'interno del nostro squallidissimo ostello... Veramente grazie mille!!

Per quanto riguarda gli 'amici di vecchia data', quelli di Lecco, che non sono stati fondamentali dal punto di vista universitario ma sono stati decisamente importanti moralmente, posso citare *Gabry*, *Bolt*, ... Ma un ringraziamento veramente grande va a *Ciccio*: nonostante per un breve periodo ci siamo persi di vista, lui mi ha fatto capire cosa vuol dire amicizia ed aiutarsi nel momento del bisogno. E non posso certo dimenticare *Sara*; grazie di tutto, dagli infiniti viaggi in treno fino a tutte le varie chiaccherate...

Naturalmente non può mancare un grazie di cuore alla mia famiglia, i miei genitori e i miei due fratelli, *Davide* e *Michele*; sono stati e saranno sempre fondamentali per me. Hanno sopportato senza mostrarlo i miei 'scleri' durante i periodi di studio, mi hanno consigliato sempre bene quando ero indeciso sul da farsi, sia in ambito scolastico che extrascolastico; non potrò mai dimenticare il loro supporto.
Un altro ringraziamento molto sentito va a tutti i miei parenti e soprattutto al clima che riescono sempre ad instaurare ogni volta che ci si vede; siamo una grande famiglia molto unita e questo è un valore per me veramente fondamentale.

Infine un ultimo ringraziamento va a *ME*; sembrerò egocentrico ma bravo *Stefano*, proprio bravo, sei riuscito ad arrivare a questo traguardo ed ora ti aspetta l'inizio di una nuova avventura in un 'altro mondo' quindi... IN BOCCA AL LUPO!!!