Sicurezza dei Sistemi Peer-to-Peer per il Data Store Distribuito

tlei <tlei@s0ftpj.org>
guy montag <gm@nym.borked.net>

BFi14-dev - file 06 - 2007.09.23

Indice

Elenco delle figure

Elenco delle tabelle

BFi14-dev-06 - Sicurezza dei Sistemi Peer-to-Peer per il Data Store Distribuito - 2007.09.23
tlei <tlei@s0ftpj.org> & guy montag <gm@nym.borked.net>

Questa è una review aggiornata dei più interessanti sistemi P2P per il data store distribuito: OceanStore, Mnemosyne, Free Haven e Freenet. Le appendici sono dedicate a Mixminion e Tor, due tecnologie che non offrono funzionalità di storage, ma possono invece essere usate come canali di comunicazione anonima in sistemi P2P per il data store.
Con la descrizione dei protocolli e degli algoritmi impiegati e l'analisi delle problematiche legate alla sicurezza si desidera aiutare il lettore a comprendere questi sistemi affinchè possa in seguito "metterci le mani sopra" in modo consapevole, modificarli e progettarne di nuovi.
Data la lunghezza del documento, i capitoli sono scritti in modo tale da poter esser letti anche singolarmente e fuori ordine senza che questo ne pregiudichi la comprensione.
Grazie a Marco Calamari, Raistlin, smaster, valv0, vecna per commenti, consigli o anche solo per aver letto la bozza di questo testo.

Capitolo 1

Introduzione

La popolarità dei sistemi per il data store distribuito è cresciuta negli ultimi anni grazie ai vantaggi che essi promettono rispetto ai tradizionali data store centralizzati: tolleranza ai guasti, disponibilità, scalabilità e performance. Parallelamente si è registrata una crescente ricerca nella progettazione e messa in opera di reti peer-to-peer: a partire da servizi di comunicazione esistenti esse definiscono un livello di rete sovrapposto (*overlay network*) che realizza un servizio distribuito mediante la condivisione delle risorse (logiche e fisiche) di calcolatori (nodi paritetici). L'architettura P2P rappresenta un'attraente soluzione per implementare data store distribuiti grazie alle sue caratteristiche di decentralizzazione dei dati e del piano di controllo, di ridondanza, di adattabilità, di capacità di organizzarsi autonomamente e di basso costo di gestione [1].

Se da un lato i sistemi P2P offrono indubbi vantaggi rispetto al tradizionale modello client-server, dall'altro pongono nuove sfide nell'ambito della sicurezza. Reti P2P dove non sia limitato il numero di identità che un nodo può assumere contemporaneamente nell'*overlay network* e dove i peer possano unirsi e lasciare la rete in modo arbitrario devono far fronte a due attacchi di carattere generico [2]:

- *Sybil attack.*
  L'attaccante genera identità multiple associate al proprio nodo fisico; il numero delle identità possibili è limitato solo dalle risorse del nodo. In questo modo l'attaccante con la sua entità può arrivare a controllare porzioni significative della rete P2P. L'obiettivo del sybil attack è sovvertire il meccanismo di ridondanza e decentralizzazione della rete P2P.

- *Eclipse attack.*
  In un'estensione dell'attacco precede, l'attaccante giunge a controllare gran parte dei nodi adiacenti al nodo vittima. I nodi ostili dell'attaccante cooperano per ingannare il nodo legittimo e indurlo a inserirli nella sua tabella di routing e considerarli suoi vicini (*neighbour*). Quando il numero di nodi ostili supera un certo limite rispetto alla quantità di nodi vicini legittimi, di fatto i nodi ostili *eclissano* le comunicazioni tra il nodo vittima e gli altri nodi legittimi e a questo punto possono sovvertire il meccanismo di routing del nodo vittima, proporgli una visione della topologia di rete differente da quella reale ed agire per indurlo a cancellare o sostituire file dal data store distribuito.

Le reti P2P per il data store distribuito possono essere vulnerabili ad altri attacchi legati allo specifico protocollo di comunicazione impiegato ed ai servizi forniti; per questo motivo è opportuno condurre un'analisi di sicurezza separatamente per ognuna di esse.

Il paradigma P2P applicato ai sistemi di data store distribuito consente anche di offrire pubblicazione e ritiro anonimo dei dati dal data store. L'anonimato dell'utente (e quindi del suo nodo) è

preservato quando non è possibile identificare quale utente (nodo) della rete ha effettuato l'operazione. L'anonimato non è una condizione raggiungibile in un contesto centralizzato e non distribuito, dal momento che è sempre relativo ad un *anonymity set*, in questo caso l'insieme degli utenti (nodi) che fanno parte della rete P2P; agli occhi dell'attaccante appaiono in modo equiprobabile coinvolti nella transazione di interesse. In generale l'aumento della dimensione dell'anonymity set implica un rafforzamento dell'anonimato.

Il concetto di anonimato di un sistema per il data store include i seguenti aspetti:

- *Forward anonymity*: non è possibile identificare e localizzare il mittente della comunicazione.
- *Reverse/backward anonymity*: non è possibile identificare e localizzare il destinatario della comunicazione.
- *Plausible deniability*: chi ospita i dati è messo in condizione di negare in modo plausibile la propria responsabilità sui contenuti.

La sfida più difficile da vincere per i sistemi anonimi è posta dall'analisi del traffico (*traffic analysis*). L'analisi del traffico consiste nell'estrarre e correlare informazioni dai metadati della rete, inclusi i volumi e le tempistiche dei pacchetti di rete, così come gli indirizzi di rete visibili da cui sono originati o ai quali sono destinati. Nel caso delle comunicazioni anonime, un avversario userebbe questi dati per condurre un'analisi del traffico allo scopo di tracciare chi è l'origine o la destinazione ultima di una connessione, in modo da violare le proprietà di anonimato che il sistema è progettato per offrire.

I concetti finora esposti sono propedeutici alla comprensione delle sezioni successive di questo documento, dove verranno ripresi ed approfonditi. L'analisi dei sistemi P2P per il data store distribuito proposta verterà sulle funzioni di protezione dei file (intergrità, autenticità, segretezza), eventuali garanzie di forward e reverse anonymity e plausible deniability, possibilità di eseguire ricerche sui dati, vulnerabilità note/individuate e relativi attacchi possibili. Gli attacchi alle componenti crittografiche esulano dagli obiettivi di questo documento e pertanto non saranno trattati [3] [4].

Il capitolo 2 presenta le caratteristiche, le componenti e la sicurezza di OceanStore, un sistema P2P per il data store distribuito a livello globale; la sua attenzione si concentra sulla persistenza dei dati in una rete basata su nodi inaffidabili. OceanStore si basa su algoritmi di hash distribuiti (DHT), quali Tapestry e Bamboo DHT.

Nel capitolo 3 è descritto un altro data store distribuito, Mnemosyne; anch'esso si basa su Tapestry ed offre interessanti funzionalità steganografiche.

Il capitolo 4 illustra il progetto Free Haven, un ambizioso studio di una rete anonima P2P per lo storage distribuito di contenuti sensibili, che ha prodotto due fra le maggiori tecnologie per la comunicazione anonima, esaminate in appendice.

Il capitolo 5 è dedicato ad un'ampia trattazione ed analisi di sicurezza del progetto Freenet che costituisce lo stato dell'arte per quanto riguarda i sistemi *anonimi* P2P per il data store ed è la rete di questo tipo che gode della più ampia base di utenti.

A partire dallo studio degli specifici software P2P per il data store distribuito, nel capitolo 6 si propongono considerazioni di carattere generale su elementi e proprietà che influiscono sulla sicurezza di questa tipologia di sistemi e dovrebbero essere tenute presenti in fase di progetto. In questa analisi conclusiva i temi trattati includono routing, affidabilità, fiducia, accountability, replicazione dei dati, resilienza ai guasti, usability ed anonimato.

L'appendice approfondisce due tecnologie di comunicazione anonima che possono essere impiegate in sistemi P2P per il data store distribuito per ottenere proprietà di anonimato:

- Mixminion: un protocollo per le comunicazioni anonime ad elevata latenza, concepito per lo scambio anonimo di messaggi di posta elettronica ed utilizzabile come canale anonimo per altri applicativi.
- Tor: applicazione di rete che rappresenta lo stato dell'arte per quanto riguarda la comunicazione anonima a bassa latenza.

Capitolo 2

OceanStore

Ideato nel 2000 presso l'University of California di Berkeley, OceanStore [5] [6] vuole essere un'infrastruttura di rete distribuita a livello globale per offrire accesso controllato a generici dati persistenti. Questo storage di rete, utilizzando ridondanza e tecniche crittografiche, si colloca su server distribuiti inaffidabili; i dati possono essere ospitati ovunque sulla rete ed essere presenti con un numero arbitrario di copie, risolvendo i problemi prestazionali tramite *caching*.

OceanStore è progettato per essere un livello di rete stabile, relativamente performante e sicura, poggiato su una rete instabile; tuttavia il suo sviluppo non è ancora concluso. Dal 2002 è testato presso PlanetLab [7], un insieme globale di nodi per la ricerca nel campo delle reti; nel gennaio 2006 i nodi virtuali utilizzati da OceanStore erano 642. È principalmente portato avanti da John Kubiatowicz che ne ha assegnato lo sviluppo a vari studenti. Su licenza BSD è liberamente disponibile un prototipo di nome Pond [8], scritto in Java, che raccoglie i meccanismi di OceanStore.

Per quanto riguarda l'instradamento e la distribuzione dei blocchi di dati, OceanStore storicamente si basa su Tapestry, un meccanismo di hash distribuito (DHT - *Distributed Hash Table*) molto efficiente, che permette una rapida localizzazione dei blocchi di dati richiesti, anche in caso di malfunzionamenti di rete. Tapestry è stato sviluppato in gran parte dagli sviluppatori di OceanStore, fra cui spicca Ben Zhao. Anch'esso è quindi un software nato a Berkeley, ed è utilizzato come base per diversi progetti di infrastrutture distribuite. Il suo sviluppo si è fermato alla release 2.0, in quanto soppiantato nel 2004 da Chimera [9], una riscrittura sotto forma di libreria C che ne ha raccolto l'eredità. Dal 2006 il meccanismo di instradamento e di localizzazione in OceanStore è affidato a Bamboo, un progetto di Berkeley ancora in fase di sperimentazione, sviluppato da Sean Rhea, uno dei progettisti di OceanStore. È attualmente integrato nel codice di Pond, avendo entrambi la stessa licenza; il pacchetto si può ottenere tramite CVS da SourceForge [10].

2.1 Caratteristiche

Secondo il team di OceanStore, le premesse ad un sistema distribuito di dimensioni globali che gestisca dati persistenti sono le seguenti:

- connessioni veloci sempre disponibili, quindi non di tipo *on-demand*;
- la rete non è fidata, ed i nodi della rete possono essere ostili;
- i nodi della rete possono fallire oppure entrare e uscire a piacere dalla rete.

Inoltre gli obiettivi principali sono:

- persistenza dei dati;
- cifratura delle connessioni;
- autenticazione degli utenti;
- elevate prestazioni.

Tutto questo con l'idea di poter gestire una rete composta da 10 miliardi di persone con 10 mila file immessi da ognuna.

L'indipendenza della rete rispetto al comportamento dei nodi è una premessa necessaria per l'affidabilità di una rete globale, in quanto poggia su server e connessioni preesistenti. Questa infrastruttura pre-esistente non è fidata in quanto possono esistere nodi della rete malevoli. Tuttavia per poter funzionare i client che si collegano alla rete devono conoscere una classe di server sicuramente fidati a cui trasmettere i dati. Ma si deve anche assumere che gran parte dei server lavorino correttamente, a causa della gestione distribuita della consistenza basata sull'algoritmo *Byzantine Agreement Protocol*. Quest'algoritmo permette di essere ragionevolmente certi della corretezza di un messaggio ricevuto che ha attraversato vari nodi; per ogni aggiornamento interno alla rete avviene infatti un controllo di consistenza con gli altri nodi, onde evitare di accettare informazioni generate da un nodo ostile. Dall'algoritmo risulta che il numero massimo tollerabile di nodi disonesti è M su un totale di $N = 3*M+1$ nodi; risulta quindi che per funzionare correttamente il numero di nodi malevoli M è:

$$M < (N-1)/3$$

Il numero più piccolo di nodi per funzionare è quindi 4, con 1 nodo disonesto; se infatti vi siano soltanto 3 nodi, A, B e C, con A disonesto che informa B e C con istruzioni differenti, durante il loro raffronto né B né C possono sapere chi degli altri due è disonesto. Nel caso in cui siano più di M i nodi disonesti, è possibile effettuare un *Eclipse attack*, detto così in quanto più nodi malevoli riescono ad eclissare le informazioni corrette fornite da un nodo onesto.

La struttura generale di OceanStore è una gerarchia di 2 livelli di nodi peer-to-peer. I client accedono ad uno qualsiasi dei nodi di OceanStore, una vasta rete di nodi peer-to-peer che memorizzano i dati ed effettuano le operazioni di instradamento e localizzazione. Alcuni di questi nodi, che formano il *primary tier*, sono i punti di riferimento per quando riguarda gli aggiornamenti ai dati; topologicamente è una rete di nodi peer centrali alla rete. Tutti gli altri nodi formato il *secondary tier*, che possono essere collegati sia ad uno o più nodi del primary tier e a nodi del secondary tier, sia soltanto a nodi peer del secondary tier. Si forma quindi una rete peer-to-peer in cui nelle operazioni di scrittura alcuni peer centrali sono più importanti degli altri nodi. Le modifiche vengono infatti validate nel primary tier e si diffondono nel secondary tier, in entrambi i livelli grazie al Byzantine Agreement Protocol. I nodi del primary tier devono essere chiaramente in numero maggiore di 4.

Per ottenere connessioni veloci in una rete così vasta, è necessario implementare meccanismi di caching: la località dei dati è quindi importante, in quanto si riduce la latenza, si incrementa l'affidabilità poiché i dati fanno meno strada e si riduce la banda complessivamente utilizzata nella rete. I dati possono essere presenti in cache (queste copie sono detti "*replica*") su qualsiasi nodo: la ricerca di un dato presente localmente deve possibilmente interessare soltanto i nodi locali alla richiesta. Si dice quindi che i dati sono *nomadici*: monitoraggi interni al funzionamento della rete scovano relazioni fra gli oggetti memorizzati e li spostano e duplicano a seconda delle necessità, in modo del tutto automatico e progressivo.

Il tipo di memorizzazione offerto mantiene nel tempo ogni versione dei dati immessi, senza perdere alcun contenuto, poiché alcune replica per ogni versione di un contenuto sono statisticamente sempre presenti nella rete: tecnicamente il sistema si definisce quindi come un *version-based archival storage system*. È possibile a posteriori ricostruire ogni versione del dato immesso con le successive modifiche; non è quindi possibile cancellare un contenuto dopo l'inserimento.

Alla base di OceanStore vi è il concetto che l'informazione è separata dalla collocazione fisica; per funzionare al meglio i server dovrebbero essere uniformemente distribuiti sul globo. In tal modo si avrebbe la configurazione ideale per l'accesso in tempi rapidi da parte dei client.

Tapestry, ora sostituito da Bamboo, è integrato con OceanStore ed è utilizzato per quanto concerne la localizzazione e l'instradamento degli oggetti. Il loro utilizzo è fondamentale per gestire localmente tali funzioni, a meno che non sia strettamente necessario fare richieste non locali, sfruttando la località nomadica dei dati illustrata precedentemente.

Per questioni di portabilità, OceanStore, Tapestry e Bamboo si basano su una JVM (*Java Virtual Machine*).

2.2 Tapestry

Tapestry [11] è un livello avanzato di rete che offre un servizio di routing efficiente e scalabile per reti peer-to-peer. Esso implementa un'infrastruttura decentralizzata (è un DOLR, *Decentralized Object Location and Routing* [12]) utilizzando una tabella di hash per la localizzazione e l'instradamento degli oggetti, e quindi è una DHT (*Distributed Hash Table*). È una delle prime quattro DHT concepite verso il 2000, assieme a CAN, Pastry e Chord, alla base delle reti peer-to-peer di seconda generazione. Nel suo caso il posizionamento delle risorse tiene conto del percorso utilizzato per raggiungerlo: questa caratteristica di fondo permette ad OceanStore di ottenere la località degli oggetti cercati.

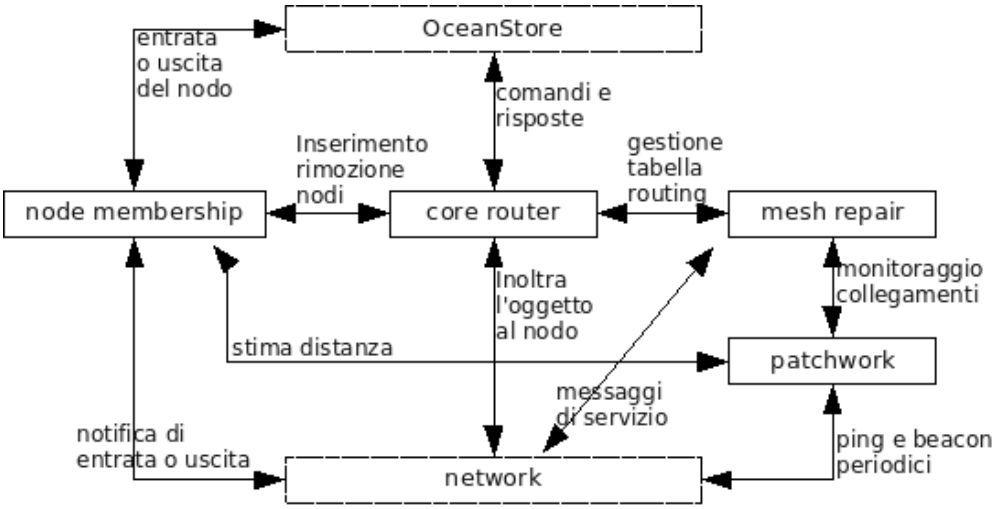

Figura 2.1: Schema delle componenti di Tapestry e loro interazioni con la rete e OceanStore.

È concepito per adattarsi a reti IP in cui sia i nodi che i link siano inaffidabili. Può utilizzare sia TCP che UDP; il vantaggio ben noto di TCP è l'implicito controllo di flusso e anti-congestionamento, mentre come contro ha i lunghi tempi di apertura e chiusura dei flussi. Il livello UDP, essendone privo, viene fornito assieme ad un controllo di congestione simile al TCP. La preferenza di un protocollo

rispetto all'altro si limita di solito a considerazioni sulle prestazioni, in quanto l'utilizzo del TCP utilizza più file descriptor rispetto ad UDP.

Vengono di seguito elencate le principali componenti logiche di Tapestry:

- *Core router*
  Utilizza il routing e una tabella di indici agli oggetti per gestire l'inoltro di messaggi. La ricerca effettuata può essere di due tipi:

  - ricerca di oggetti, che ha complessità $O(1)$ se tutte le referenze sono contenute in memoria;
  - ricerca di nodi, che ha complessità $O(logN)$ e si appoggia su un algoritmo di ricerca veloce chiamato *Bloom Filter*.

    Bloom Filter [13] è un algoritmo probabilistico per verificare rapidamente l'appartenenza di un elemento ad un vasto insieme di valori (*set*), utilizzando h differenti funzioni di hash su un un vettore di bit identificativo dell'elemento.
    Nel caso di Tapestry si utilizza un versione semplificata e applicabile all'ambito distribuito, l'*Attenuated Bloom Filter* (ABF). Esso utilizza un solo algoritmo di hash per la ricerca di un elemento. ABF è una rappresentazione approssimativa di un gruppo di elementi, che oltre a determinare l'assenza di un elemento molto velocemente, può lavorare per passi successivi su diversi set di elementi.
    In Tapestry l'algoritmo gestisce elementi identificati da vettori v lunghi 160 bit non ordinati, ovvero gli identificativi di un oggetto (GUID) generati con un hash SHA-1.
    È approssimativa in quanto può generare falsi positivi, ma mai falsi negativi. È quindi possibile che un elemento venga individuato su un nodo benchè non presente, ma non capita mai che un elemento non venga trovato anche se presente; il numero statistico di questi falsi positivi è inversamente proporzionale con il numero v di bit utilizzati per rappresentare l'elemento.
    È veloce in quanto localmente il tempo per calcolare l'appartenenza di un elemento al set non dipende dal numero di elementi del set, ma è costante e dipende dalla lunghezza dell'hash; per cui ben si applica ad un sistema peer-to-peer con un elevato numero di oggetti.
    Il suo funzionamento è distribuito poiché permette, oltre a verificare l'appartenenza dell'elemento al set locale, di passare la richiesta al nodo vicino più indicato, proprio grazie all'hash dell'elemento cercato. Ogni volta che si vuole localizzare un elemento si parte da un punto a caso della rete e si confronta l'hash attuale con gli elementi vicini, spostandosi sull'elemento che ha più bit in comune con il nostro hash.
    In figura 2.2 è raffigurato il funzionamento in Tapestry dell'Attenuated Bloom Filter con $v = 5$. Esso prevede che ogni nodo mantenga un numero di oggetti il cui hash corrisponde, solo con i bit posti ad uno, con il valore identificativo del nodo stesso. La ricerca (così come l'inserimento) dell'oggetto il cui hash ha posto ad uno i bit 0, 1 e 3 (ovvero $11*1*$) parte casualmente dal nodo a. Dopo aver verificato che i bit del valore del nodo a (11100) non soddisfano il Bloom Filter locale, si confronta con la tabella dei nodi adiacenti, contenente per ogni valore dei nodi adiacenti, l'OR logico dei nodi vicini di secondo grado; nello schema l'unico nodo adiacente è b, con valore 11100, a cui corrisponde il pattern 11011. Il nodo b non può soddisfare i bit richesti; tuttavia il pattern del suo vicinato sì. La richiesta giunge quindi a b, che scopre nella sua tabella

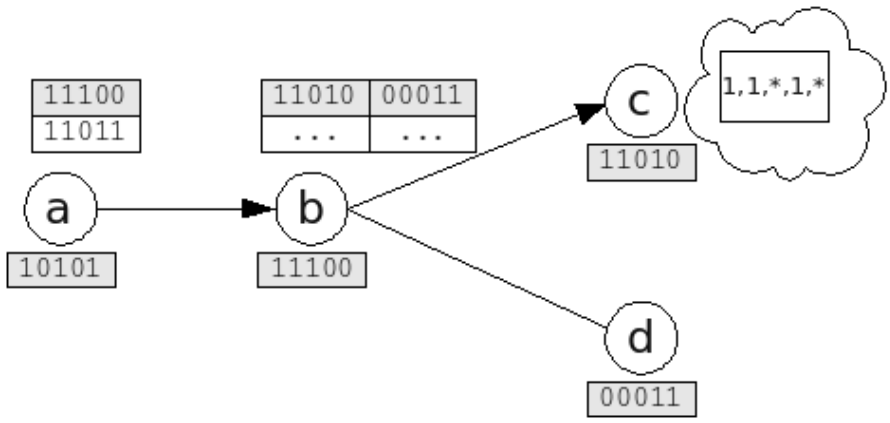

Figura 2.2: Funzionamento dell'Attenuated Bloom Filter in Tapestry.

che il valore del nodo c (11010) soddisfa il Bloom Filter; per cui la richiesta arriva a c, che verifica se effettivamente contiene nei suoi oggetti locali quello il cui hash ha i bit 0, 1 e 3 posti ad uno.
Il suo utilizzo in Tapestry è decisivo poiché la verifica avviene in parte su differenti nodi; a livello globale tutti i riferimenti non stanno certo in memoria principale. Inoltre siccome la maggior parte dei percorsi di routing riceve risultati negativi, è molto probabile che bastino pochi passi per identificare la sequenza di bit corretta.
Si noti infine che l'identificativo di un oggetto corrisponde anche all'informazione sulla sua locazione; inoltre nel caso reale di OceanStore può convenire iniziare la ricerca dai nodi del primary tier, poiché sono più centrali.

- *Node membership component*
  È responsabile dell'integrazione dei nuovi nodi e dell'uscita volontaria dei nodi.
- *Mesh repair*
  È responsabile dell'adattamento delle connessioni Tapestry a seconda dei cambiamenti della rete, nel caso di collegamenti interrotti oppure nel caso di nodi malfunzionanti.
- *Patchwork*
  Monitora le informazioni sulla latenza e sulle perdite dei canali di comunicazioni.

Ad ogni nodo Tapestry viene assegnato un `NodeID` pseudo-casuale composto da 160 bit generati tramite SHA-1, espresso sotto forma di stringa esadecimale da 40 caratteri. Il meccanismo di instradamento dei messaggi è detto *Mesh Routing*, poiché ogni nodo utilizza direttamente i `NodeID` dei nodi vicini per raggiungere la meta. Infatti dato un nodo da raggiungere, il messaggio è trasmesso da ogni nodo al vicino con il `NodeID` più simile, utilizzando un algoritmo che ad ogni nodo incrementa il suffisso di una stringa inizialmente di 1 carattere. La ricerca su una rete con N `NodeID` esadecimali casualmente distribuiti è dell'ordine di $log_{16}N$; se alla fine della ricerca il nodo non viene trovato, il messaggio rimane sul nodo con il `NodeID` più simile, che può quindi appropriarsene.

2.2.1 Analisi vulnerabilità note

Tipicamente questi framework peer-to-peer per il routing sono vulnerabili ad un noto tipo di attacco, il *Sybil attack* [14], in cui si costruiscono ad-hoc molte identità per un nodo. Infatti questi sistemi

per resistere alle informazioni errate di un nodo malizioso utilizzano ridondanza; tuttavia, se un nodo raccoglie un gran numero di identità può tentare collisioni, poiché controlla una parte sostanziosa del sistema, insidiando il meccanismo di ridondanza.

Tapestry limita questi tipi di attacchi in quanto si basa su una infrastruttura fidata a chiave pubblica (*PKI*) per assegnare alle identità degli ID univoci, che sono quindi certificati. Per una discussione più approfondita sul Sybil attack ed una valutazione dell'efficacia della soluzione basata su PKI, si faccia riferimento alla sezione 6.3 del capitolo 6.

Al fine di garantire il corretto funzionamento del Mesh Repair, per limitare i possibili danni causati da qualche nodo compromesso o malfunzionante, i nodi di Tapestry possono lavorare appaiati scambiandosi messaggi attraverso i nodi adiacenti per verificare i percorsi.

Il software di Tapestry è implementato a livelli, permettendo così di analizzare solo l'header dei messaggi ricevuti da trasmettere ad altri nodi. Questo evita di copiare il messaggio completo in memoria (*byte copying*) e quindi evita il passaggio dei dati trasportati (appartenenti ad OceanStore in questo caso) alla memoria riservata a Java, che può essere sotto osservazione da parte di un processo malevolo.

2.3 Bamboo / OpenDHT

OpenDHT [15] [16] è una rete sperimentale che espone un servizio DHT, utilizzando il software Bamboo [17] [18] [19]; in effetti OpenDHT serve anche al testing e allo sviluppo di Bamboo. Bamboo è implementato in Java ispirandosi in parte ai protocolli di Pastry; il suo meccanismo di interazione fra i nodi è del tutto nuovo. Rispetto ai DHT di generazione precedente offre prestazioni e scalabilità migliori, soprattutto in applicazioni in cui vi sono notevoli e continui ingressi ed uscite di nodi dalla rete; tutto questo cercando di limitare al massimo la banda occupata dalle comunicazioni del sistema.

Il servizio OpenDHT espone un'interfaccia standard ad un DHT pubblica e senza limiti di accesso, grazie alle tecnologie SunRPC e XML-RPC. È possibile immettere ed estrarre chiavi senza dover installare un DHT; questo è utile allo sviluppo di reti distribuite che vogliano appoggiarsi a tale servizio senza reimplementare un DHT. Il progetto è incubato presso il PlanetLab, di cui utilizza circa 200 nodi.

Bamboo è implementato in Java strutturato utilizzando una macchina a stati fornita dal framework SEDA (*Staged Event-Driven Architecture*); il codice è quindi organizzato con code di eventi di una macchina real-time.

Le funzionalità fornite ad OceanStore sono le stesse di Tapestry, ovvero routing, ricerca, inserimento ed estrazione di chiavi; anche qui il routing utilizza un meccanismo ricorsivo durante la ricerca di chiavi. Esso si basa come Tapestry su chiavi date da hash SHA-1 a 160 bit.

Essendo in fase di testing, alcuni dettagli sono ancora poco chiari, soprattutto gli aspetti riguardanti la sicurezza, che è proprio l'ambito di sviluppo attuale. Alcuni problemi di sicurezza presenti al momento sono per esempio la possibilità di effettuare una serie di insermenti su un nodo Bamboo, provocando l'esaurimento del suo spazio di memorizzazione, causando così un Denial of Service (*DoS*). Un altro tipo di DoS è un attacco costruito in modo tale che un nodo di Bamboo rediriga molte richieste verso un terzo nodo obiettivo.

2.4 Modello dei dati

L'unità fondamentale per la memorizzazione di informazioni in OceanStore è l'oggetto: sono oggetti copie aggiornabili di dati (dette *replica*) e copie archiviate di blocchi di dati. Gli oggetti si trovano

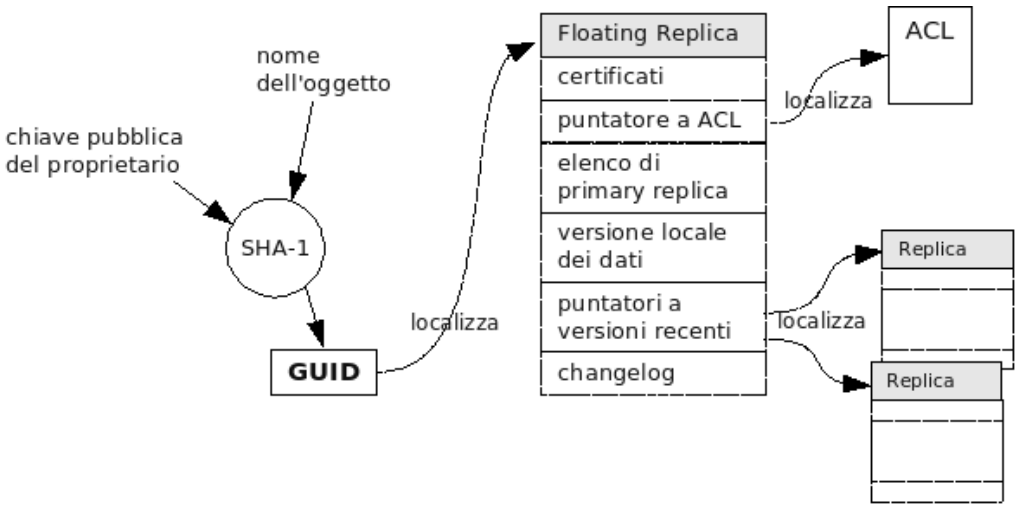

Figura 2.3: Un oggetto attivo definito da un GUID e con puntatori ad oggetti aggiornati.

quindi in due forme dette *attiva* e *archiviata*.

Nella forma attiva vi è l'ultima versione dell'oggetto, con i necessari puntatori per l'aggiornamento; l'oggetto contiene infatti una copia dell'ultima versione archiviata, i cambiamenti non ancora resi noti, i permessi di accesso e i metadati necessari. La forma attiva si trova presso i nodi dove viene utilizzata più spesso: queste copie prendono il nome di *floating replica*; per riunire le modifiche ai vari blocchi e ridistribuire le forme attive e archiviate vengono impiegati i *primary replica*, alcune copie attive di riferimento.

Un oggetto di tipo archiviato è una rappresentazione in sola lettura che viene sparsa su centinaia o migliaia di nodi; in caso di gravi errori o perdita dei replica, dalle varie versioni delle forme archiviate si dovrebbe sempre poter ricostruire i dati.

Il sistema con i due livelli di replica e le forme archiviate viene chiamato dagli sviluppatori *deep archival storage*, poiché fornisce una ridondanza di ogni versione degli oggetti, raggiungendo un buon compromesso con lo spazio utilizzato. Sono state fatte delle analisi statistiche a riguardo: ipotizzando di avere m frammenti archiviati e n copie, l'aumento di spazio necessario è n/m. Nel caso si utilizzi 10^6 nodi in cui il 10% perde i dati, e si replichi i dati ($n/m = 2$), otteniamo per 4, 8 e 16 frammenti di un documento:

- $m = 4$, $n = 8$: probabilità di trovare il documento è 0.99
- $m = 8$, $n = 16$: probabilità di trovare il documento è 0.99999
- $m = 16$, $n = 32$: probabilità di trovare il documento è 0.99999... (20000 nove)

2.4.1 Identificazione e localizzazione

Un oggetto di OceanStore è identificato da un GUID (*Globally Unique IDentifier*); è una stringa pseudo-casuale con lunghezza in bit fissa, che viene generata differentemente a seconda che l'oggetto sia in sola lettura oppure sia attivo. La figura 2.3 illustra la formazione ed il contenuto di un oggetto attivo.

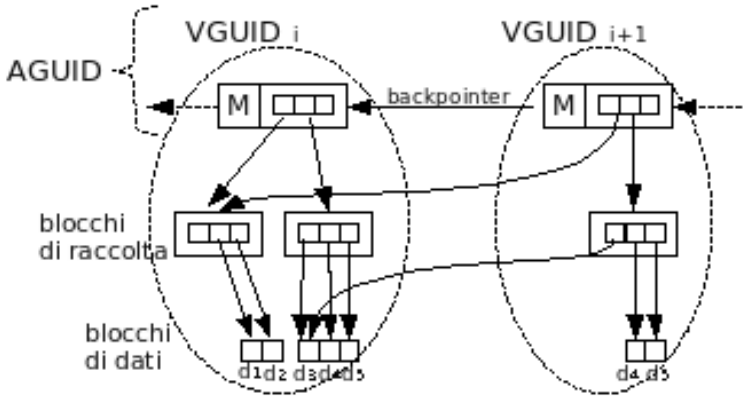

Figura 2.4: Relazione tra AGUID e VGUID di un primary replica in Pond. Due versioni di un oggetto e loro composizione dai blocchi di dati archiviati; il secondo VGUID aggiorna i blocchi $d4$ e $d5$.

Il GUID di un oggetto in sola lettura è un hash (si utilizza SHA-1) sui blocchi di dati che contiene; in questo modo ogni versione di ogni oggetto è identificato univocamente. Inoltre i client che chiedono un oggetto archiviato in sola lettura attraverso il suo GUID possono facilmente verificare la sua correttezza, ricalcolando l'hash.

Per i replica la questione è più delicata, in quanto è necessario poterli trovare dato un nome facile da memorizzare per l'uomo; serve quindi un meccanismo decentralizzato per assegnare un GUID resistente ad attacchi di *hijacking* dei dati da parte di utenti ostili, che vorrebbero distribuire un dato non autentico con un nome od un GUID autentico. Per evitare il problema, il GUID viene costruito generando l'hash con SHA-1 della stringa ottenuta concatenando il nome dell'oggetto (il titolo del blocco di dati) e la chiave pubblica del proprietario (l'ultimo ad aver modificato questa versione). I nodi possono quindi verificare l'autore dei dati rapidamente, facilitando il controllo degli accessi ai dati.

In OceanStore si possono generare dei percorsi a file utilizzando oggetti che contengono riferimenti ad altri oggetti; il percorso è quindi dato da una successione di GUID. Si noti che in sé questo meccanismo simile ad un filesystem non ha una radice unica, ma è l'utente che può costruirsene una: prendendo un oggetto di tipo directory, che contiene riferimenti ad altri oggetti, è possibile per un utente utilizzarlo come radice generandone una propria versione con la propria chiave; da lì in poi ogni nuova directory è implicitamente una firma digitale della successiva (*self-certifying path* [20]).

Il meccanismo per generare i GUID non assicura la totale infallibilità della corrispondenza univoca con gli oggetti, poiché sposta il problema sull'algoritmo di hash utilizzato, che per ora, sia in Tapestry che in Bamboo, è SHA-1 a 160 bit. OceanStore è comunque implementato in modo tale da permettere aggiornamenti futuri sulle funzioni di hashing utilizzate (per esempio SHA-512).

I GUID formati con l'hash del nome e della chiave pubblica del proprietario sono chiamati brevemente AGUID (Active GUID), quelli di un blocco di dati archiviato BGUID (Block GUID) e quelli del blocco d'origine di una versione corrente VGUID (Version GUID, è in verità un BGUID ma è il riferimento presente nel replica).

2.4.2 Accesso ai dati

Le azioni consentite sugli oggetti sono:

- la memorizzazione di un nuovo dato;

- l'aggiornamento, che crea una nuova versione;
- lettura dell'ultima versione;
- lettura di una versione precedente.

La memorizzazione e l'aggiornamento non cancellano mai dati preesistenti, ma generano nuovi oggetti o nuove versioni di oggetti. È quindi assicurato il cosiddetto *time travel*, che permette di analizzare il resoconto di ogni modifica, in quanto nella versione corrente di un oggetto vi sono i VGUID agli oggetti archiviati.

La lettura di un oggetto presuppone l'individuazione dello stesso, che può essere effettuata in due modi: conoscendo già l'identificativo e la versione voluta di un blocco di dati tramite il BGUID o VGUID, oppure facendo una ricerca sul nome dell'oggetto, ottenendo quindi l'AGUID e l'ultima versione dell'oggetto, con i collegamenti ai VGUID.

2.4.2.1 Permessi di lettura e scrittura

OceanStore fornisce due primitive per il controllo degli accessi: permesso di lettura e permesso di scrittura. Altri tipi di regole più complicate, quali gruppi di lavoro, si riconducono a queste.

Per limitare la lettura non autorizzata di documenti, tutti i dati di OceanStore non completamente pubblici sono crittografati, e la chiave di decifratura è distribuita agli utenti con permessi di lettura. Per revocare il permesso di lettura ad un oggetto senza modificarlo, il proprietario dell'oggetto deve richiedere che i replica debbano essere cancellati e cifrare di nuovo il contenuto.

Può sembrare strano il fatto che un utente a cui è stato revocato il permesso di lettura può accedere agli oggetti in cache (nei floating replica) non ancora ri-cifrati; è anche possibile che un nodo malevolo non aggiorni i cambiamenti di permessi. Tuttavia questo problema è implicito a tutti i filesystem, in quanto non è possibile impedire che un utente ricordi ciò che ha letto.

L'accesso in scrittura è verificato in modo molto diverso da quello in lettura, in quanto le letture sono limitate dagli utenti grazie allo scambio di chiavi, mentre le scritture sono controllate dai server esaminando certificati digitali. Infatti una modifica all'oggetto è accettata oppure no dal nodo a seconda della ACL (*Access Control List*) presente nell'oggetto stesso imposta dal proprietario. Questo campo nell'oggetto può anche essere un puntatore ad un altro oggetto contenente una ACL. I permessi concessi da queste access list sono molto semplici: si descrive per ogni chiave pubblica di un utente i suoi permessi sull'oggetto, non si descrive l'identità degli utenti né la presenza di utenti maggiormente privilegiati.

Per ovvi motivi di gestione, questo campo contenente i permessi e gli oggetti che contengono solo permessi non sono mai cifrati, cosicché i nodi possano sempre verificare le regole.

Inoltre la memorizzazione di un nuovo blocco di dati non richiede consensi, se non limitazioni da parte del nodo a cui l'utente è connesso; il proprietario deve solamente generare l'oggetto con la sua chiave pubblica, e imporre se ne ha bisogno i permessi necessari. L'inserimento in un percorso a directory, come detto precedentemente, è del tutto personale e non necessita quindi di essere reso pubblico.

2.4.2.2 Scrittura ed operazioni sui blocchi cifrati

In OceanStore i cambiamenti ai dati di un oggetti vengono effettuati dagli utenti che generano nuove versioni. Ogni tentativo di aggiornamento viene registrato, che sia andato a buon fine oppure no. Nel caso di aggiornamenti concorrenti eseguiti con la stessa chiave pubblica, viene utilizzata una politica

di merging dei blocchi di dati; possono però accadere conflitti non risolvibili, che vengono gestiti generando una nuova versione dell'oggetto.

Il modello di aggiornamento dei dati di OceanStore funziona anche con i blocchi di dati cifrati; infatti i nodi implicati nella procedura di aggiornamento non devono poter conoscere il contenuto di oggetti riservati. Questo è essenziale per poter attuare un sistema distribuito peer-to-peer affidabile. È da notare che la maggior parte dei dati sono contenuti in oggetti cifrati, in quanto la presenza di dati con permessi assegnati sono la norma in un qualsiasi filesystem.

Questa complicazione è risolta utilizzando un algoritmo di cifratura position-dependent a blocchi. Le operazioni possibili sono quindi:

- confronto di versione;
- confronto di dimensione;
- confronto di blocchi;
- ricerca;
- sostituzione di blocchi;
- aggiunta in coda di blocchi;
- inserimento di blocchi;
- rimozione di blocchi.

Le prime due operazioni sono banali in quanto lavorano sui metadati. Il confronto di blocchi è possibile utilizzando un algoritmo di cifratura come indicato: l'utente calcola l'hash del blocco cifrato che vuole confrontare e lo invia ai nodi OceanStore che effettueranno il controllo corrispondente.

La ricerca può essere effettuata direttamente sul testo cifrato, senza rivelare in chiaro la stringa di ricerca ma solo l'esito della ricerca; né è possibile per i nodi della rete effettuare ricerche di loro iniziativa.

> Il funzionamento di questa ricerca all'interno di un blocco di dati cifrato è stato dimostrato rigorosamente da Dawn Xiaodong Song, David Wagner e Adrian Perrig [21]. Il trucco sta nel produrre il testo cifrato da memorizzare effettuando uno XOR bit a bit tra le singole parole cifrate con un algoritmo deterministico (di tipo ECB) e degli *stream cipher* generati partendo da sequenze pseudo casuali con una chiave nota.
>
> La ricerca su questo testo cifrato prodotto dallo XOR presente sul server inaffidabile avviene inviandogli la parola da cercare cifrata e la chiave utilizzata per generare lo stream impiegato nell'operazione di XOR. Il server deve quindi effettuare lo XOR fra il testo cifrato e la parola cifrata inviatagli, e verificarne la corrispondenza con il ciphertext generato dalla chiave nota.
>
> Schematizzando in figura 2.5, al fine di generare il testo protetto da memorizzare sul nodo inaffidabile si procede come segue:
>
> - il testo in chiaro viene cifrato dall'utente con una chiave privata Ks utilizzando un algoritmo di codifica a blocchi (ECB) sulle parole prese ad una ad una;
> - l'utente produce una sequenza di bit pseudocasuali (stream cipher) generato con una chiave nota Kp;

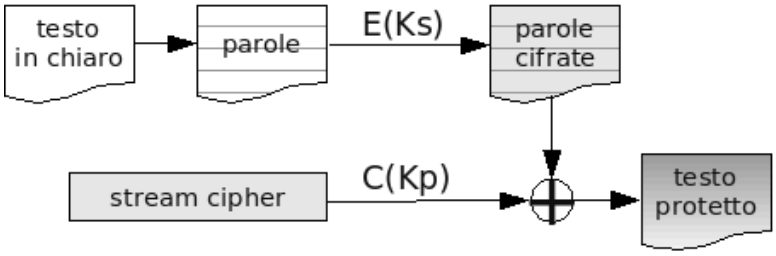

Figura 2.5: Creazione ciphertext da memorizzare sul nodo inaffidabile.

- essendo il testo cifrato facilmente riconoscibile dal momento che una parola produce sempre lo stesso ciphertext, l'utente effettua lo XOR binario tra il testo cifrato e la sequenza di bit pseudocasuale;
- il risultato produce il testo in una nuova forma cifrata in cui ad ogni medesima parola non corrisponde lo stesso ciphertext;
- questo testo protetto viene inviato al nodo che lo memorizza.

Ora che tale testo cifrato è presente sul nodo, la ricerca di una parola da parte di un utente si effettua come segue:

- la parola da ricercare viene cifrata dall'utente con la stessa chiave privata Ks e l'algoritmo ECB di prima;
- l'utente invia al nodo tale parola in forma cifrata e la chiave nota Kp utilizzata per generare la sequenza pseudocasuale di bit utilizzata prima;
- il nodo genera la sequenza pseudocasuale tramite la chiave nota inviatagli;
- il nodo effettua lo XOR blocco per blocco tra il testo cifrato che memorizza e la parola cifrata ricercata dall'utente;
- per ogni blocco verifica se il risultato compare nella sequenza pseudocasuale;
- in caso affermativo la parola cercata dall'utente è presente nel testo cifrato che risiede sul nodo.

Il server non può venire a sapere nulla del contenuto del testo, né della stringa di ricerca; tuttavia analizzando la lunghezza della stringa di ricerca, il server potrebbe effettuare attacchi statistici. Il server non può effettuare ricerche di sua iniziativa, non disponendo del testo cifrato con la chiave Kp.

Infine si noti che quest'algoritmo è computazionalmente pesante per il server avendo un ordine $O(n)$ lineare con la lunghezza n del testo presente; le cose si complicano nel caso di ricerche con caratteri jolly, poiché è necessario ricercare tutte le combinazioni possibili.

La sostituzione di blocchi e l'accodamento di blocchi cifrati sono semplici, sempre utilizzando un algoritmo di cifratura indicato per i confronti. Per l'inserimento e la cancellazione di blocchi cifrati l'operazione è più delicata e richiede che i dati siano organizzati in due gruppi: blocchi di indice e blocchi di dati. I blocchi di indice non sono cifrati e contengono puntatori ad altri blocchi all'interno dell'oggetto. Per effettuare un inserimento, il blocco presente al punto di inserimento è sostituito con un blocco di indice; questi conterrà prima l'indice al nuovo blocco da inserire, e poi l'indice del

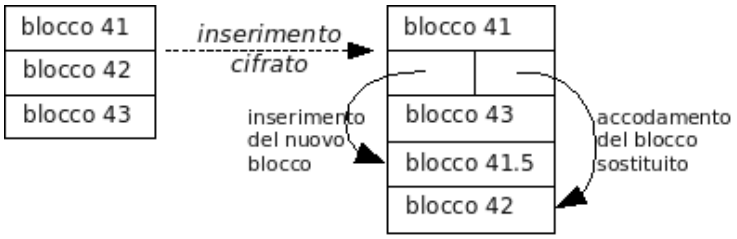

Figura 2.6: Inserimento di un blocco cifrato. L'inserimento di un blocco tra il 41 ed il 42 provoca la sostituzione del 42 con un blocco di indice e l'accodamento del nuovo blocco 41.5 e del blocco 42.

blocco sostituito, che verranno entrambi accodati all'oggetto. In questo modo i nodi che effettuano l'operazione non vengono a conoscenza del contenuto dei blocchi.

Per la cancellazione, anche qui si utilizzano i blocchi di indice: i blocchi da rimuovere sono sostituiti da blocchi di indice vuoti.

Queste operazioni di inserimento e cancellazione possono però fornire al nodo informazioni sul traffico effettuato, ed essere suscettibili ad attacchi di analisi passiva; la soluzione temporanea a questo problema inizialmente venne rimandata ad applicazioni intelligenti a livello superiore, che dovevano accodare nuovi blocchi inserendovi le operazioni da effettuare ed ogni tanto ri-cifrare gli oggetti effettuando tali operazioni.

In Pond si è poi deciso di utilizzare uno schema a due livelli di oggetti, nel più basso dei quali si trovano oggetti contenenti solo blocchi cifrati; a livello superiore si trovano oggetti contenenti solamente blocchi di indice. In questo modo le operazioni di inserimento e cancellazione si riducono a spostamenti di puntatori sul primo livello. Questo dovrebbe limitare gli attacchi di analisi di traffico, ma aggiunge gradi di complessità al modello.

2.4.2.3 Aggiornamento distribuito su rete inaffidabile

In caso di conflitti nel processo di aggiornamento di un oggetto, è necessario scegliere un ordine con il quale eseguire i vari passi, ed eseguirli nella corretta sequenza; il modello di riferimento è lo schema ACID di risoluzione dei conflitti, quello tipico delle basi di dati. Il modo più semplice per effettuare questo lavoro sarebbe far elaborare gli aggiornamenti ad un singolo nodo; tuttavia questo schema andrebbe contro l'idea di fondo di OceanStore, poiché ci si fiderebbe di un unico nodo. Il lavoro di questo nodo viene quindi ripartito ad una classe ridotta di nodi, completamente connessa, chiamata *primary tier* (o *inner ring*). Questi nodi cooperano fra di loro tramite il Byzantine Agreement Protocol nella scelta dell'ordine definitivo di esecuzione delle operazioni. Come descritto in precedenza, questo protocollo prevede che i nodi malevoli non superino M nodi su un totale di $N = 3 * M + 1$; i quattro nodi della figura 2.7 prevedono quindi che al massimo 1 nodo possa essere malevolo. Si è già accennato alla sua vulnerabilità nota col nome di Eclipse attack; questo limite degli M nodi ostili su $N = 3*M+1$ nodi totali è una limitazione implicita. Immaginando però una rete globale basata su migliaia di nodi diversi per piattaforma e proprietario diventa difficile riuscire nell'attacco; tuttavia il risvolto negativo è l'elevato numero di comunicazioni per le verifiche, dell'ordine di $O(N^2)$.

Gli altri nodi appartengono al secondary tier e comunicano fra di loro e con il primary tier tramite un algoritmo epidemico, che inoltra le informazioni a tutti i nodi. Il protocollo di diffusione dei dati è tratto dal sistema Bayou sviluppato presso il PARC [22]. La scelta di utilizzare due classi di nodi per la propagazione degli aggiornamenti è un giusto compromesso tra numero non eccessivo di connessioni

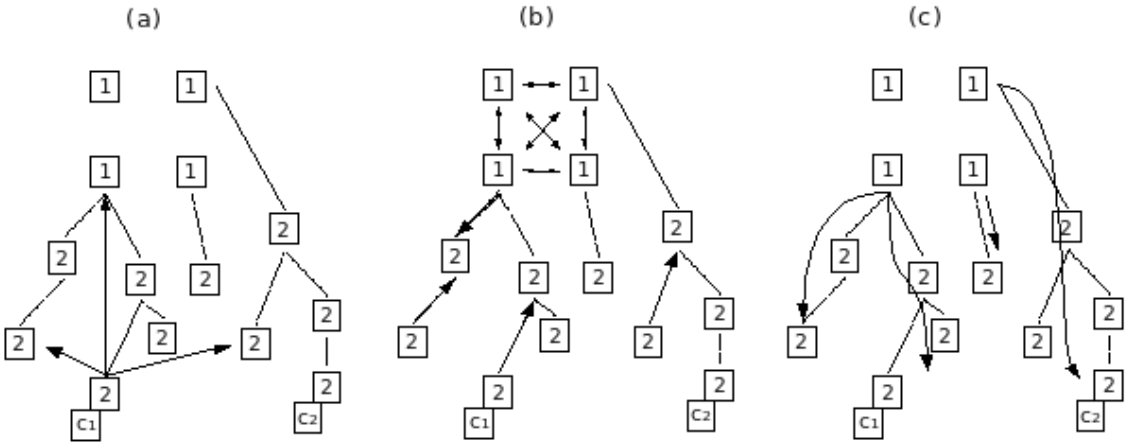

Figura 2.7: Aggiornamento con le due classi di nodi. (a) Un client invia la notifica di una modifica a vari nodi tra cui un nodo primario; (b) mentre il primary tier elabora il risultato della modifica, gli altri nodi propagano la notifica; (c) quando il primary tier ha concluso, diffonde tra i nodi il risultato.

per le verifiche e distribuzione dei compiti. Il primary tier, essendo in teoria connesso da linee a banda elevata, può effettuare le verifiche necessarie in tempi veloci.

Prima di ricevere l'ordine esatto di esecuzione degli aggiornamenti, i nodi del secondary tier continuano a diffondere l'aggiornamento non ancora verificato, non sapendo se la richiesta è giunta al primary tier; è quindi possibile per una applicazione che necessita di risultati non definitivi ma in tempi rapidi di utilizzare le richieste di aggiornamento. Infatti con la richiesta è presente un timestamp e il client può ottimisticamente valutare i timestamp di due aggiornamenti concorrenti; all'arrivo dell'informazione dal primary tier la sua supposizione sarà poi confermata oppure smentita. Siccome le decisioni del secondary tier sono tentativi, non possono essere affette da attacchi; per avere garanzie di consistenza è sufficiente attendere l'ordine fornito dal primary tier.

Infine si noti che le comunicazioni tra i nodi dell'inner ring sono autenticate da meccanismi a chiavi simmetriche, per via della loro velocità di computazione maggiore rispetto alle firme a chiave pubblica; queste ultime vengono invece utilizzate dal secondary tier, i cui nodi possono così verificare l'autenticità dei dati senza dover comunicare con il primary tier.

2.5 Conclusioni

La rete OceanStore rappresenta un notevole sforzo di realizzare un data store distribuito in grado di gestire il problema dell'affidabilità dei nodi. Una rete di questo tipo però necessita di una larga base d'utenza, senza la quale con abbastanza nodi si può facilmente sovvertire il Byzantine Agreement Protocol.

Attualmente OceanStore è, sebbene pienamente funzionante, ancora in fase di sviluppo e di testing; l'implementazione corrente evidenzia problemi di latenza nel Byzantine Agreement Protocol che rendono la rete lenta benchè, come illustrato, le operazioni avvengono localmente. Per rimediare a questo inconveniente ci si sta concentrando sullo sviluppo di un algoritmo di consistenza che richieda meno comunicazioni, pur assicurando una buona resistenza.

OceanStore inoltre è interessante per via del suo carattere commerciale. A parte le attuali problematiche prestazionali, l'interesse dei clienti potrebbe essere limitato proprio dalla natura P2P del

servizio, ancora oggi associata a fenomeni quali distribuzione di materiale illegale e violazione di copyright: gli utenti potrebbero non considerarla adatta per ospitare i propri contenuti sensibili.

Capitolo 3

Mnemosyne

Un altro storage distribuito di tipo peer-to-peer basato su Tapestry è Mnemosyne [23] [24] [25], ideato da Steven Hand e Timothy Roscoe. L'approccio rispetto ad OceanStore è però differente; si tratta di un file system steganografico, poiché utilizza i blocchi di dati dispersi sulla rete per nascondervi le informazioni utili.

La steganografia [26] è una tecnica che consente di nascondere un messaggio all'interno di un dato che funge da contenitore, in modo tale che solo il destinatario sia in grado di individuare la presenza dell'informazione nascosta. Un esempio classico è nascondere il testo di un messaggio all'interno della codifica di un'immagine. La steganografia non protegge il contenuto sensibile ma lo nasconde (è una forma di "*security through obscurity*"). Spesso il dato nascosto è comunque crittografato: si aggiunge quindi un ulteriore livello di protezione che garantisce la segretezza del contenuto nell'eventualità che l'attaccante riesca ad individuare ed estrarre l'informazione nascosta. In questo caso particolare, ad un utente che ignora la presenza di Mnemosyne oppure è privo delle chiavi corrette, il filesystem appare come un insieme di blocchi di dati non allocati e senza significato: non solo un attaccante non può accedere ai dati, ma non sa neppure se i dati esistano.

I principali vantaggi di un filesystem di questo tipo sono:

- Estraneità dei gestori dei nodi riguardo i contenuti dei dati, tecnicamente chiamata *negazione plausibile di responsabilità* (*plausible deniability*). I fornitori del servizio infatti non sono al corrente (e non possono esserlo in alcun modo) del contenuto dei dati, e quindi non possono essere considerati a tutti gli effetti responsabili delle informazioni che ospitano, compatibilmente con la legislazione locale.
- Resistenza ad interruzioni di servizio. Per un utente che vuole sicurezza in termini di accessibilità e disponibilità, un sistema distribuito è molto meno soggetto a guasti e ad attacchi di tipo Denial of Service.
- Trasmissione sicura di informazioni. Viene garantito un alto livello di riservatezza allo scambio di dati.

Ogni nodo di Mnemosyne agisce sia da client che da server: le funzioni di client comprendono l'accesso in lettura e scrittura di blocchi di dati, mentre la parte server fornisce uno spazio per la memorizzazione di blocchi di dati (non è necessariamente affidabile).

Comunemente i meccanismi per applicare la steganografia sono due:

- Creare dei file di copertura contenenti dati sensati sufficientemente entropici oppure del tutto casuali ed insensati, e memorizzare le informazioni da coprire effettuando uno XOR bit a bit con il file di copertura nelle posizioni indicate da una chiave.
- Scrivere dati casuali su tutti i blocchi di un disco; cifrare ogni blocco del file da memorizzare che viene scritto sui blocchi del disco scelti secondo un algoritmo pseudo-casuale.

L'implementazione steganografica utilizzata è la seconda, e parte dalle idee di StegFS [27], un noto filesystem steganografico locale per Linux; è adattato per essere comodamente gestito su una rete Tapestry. Tuttavia i meccanismi di Mnemosyne funzionano anche per un sistema locale, per cui verrà prima descritto il suo funzionamento in locale.

3.1 Steganografia locale a blocchi

Il filesystem è organizzato a blocchi di dati; in fase di formattazione tutti i blocchi sono colmati da dati casuali, in modo da massimizzare l'entropia sul filesystem. Il file da memorizzare è quindi diviso in parti, le quali sono cifrate con una chiave segreta e quindi scritte su blocchi sparsi del filesystem; il posizionamento è determinato da un hash calcolato a partire dal nome del file e dall'indice del blocco del filesystem. Se la chiave segreta e l'algoritmo di hashing sono buoni, i blocchi del file sono difficilmente distinguibili dai blocchi contenenti dati casuali.

Il problema di questo meccanismo sono le collisioni: infatti stessi blocchi possono essere sovrascritti da successive memorizzazioni di file. Per ovviare a questo problema si utilizza la replicazione, per cui ogni blocco è scritto in n ubicazioni differenti. La scelta di n è strategica, poiché se troppo piccolo o troppo grande il rischio di perdere blocchi a causa di sovrascritture aumenta notevolmente; gli sviluppatori indicano un numero di replicazioni n ottimale che va da 2 a 8.

Nello specifico la scrittura di un blocco di dati richiede, oltre al blocco stesso, una chiave segreta K, un valore di hash iniziale per il blocco h_0 ed una stringa da utilizzarsi come controllo di validità. Il primo passo è quello di calcolare una sequenza n di hash differenti (n è il numero di replicazione), ottenuta da una funzione di hashing H (attualmente si utilizza SHA-256) come:

$$h_0, h_1 = H(h_0), ..., h_n - 1 = H(h_n - 2)$$

Il passo successivo è quello di cifrare la copia i-esima del blocco (ovviamente $0 \leq i \leq n-1$), con la chiave $k_i = E_K(hi)$; la funzione di cifratura a blocchi $E_K()$ utilizzata è AES, con la chiave segreta K lunga 256 bit. Il blocco i-esimo è memorizzato alla posizione $bi = h_i mod(N)$, dati N blocchi totali. Si noti che l'utilizzo di una chiave k_i per ogni replica del blocco assicura l'omogeneità dei dati.

Per la lettura il processo è inverso: data la chiave K e il valore di hash h_0, si calcolano le varie posizioni delle copie del blocco e si decifrano man mano, finché un blocco passa il controllo di validità. Questo controllo di consistenza è sotto forma di stringa presente all'interno del blocco stesso; la sua struttura cambia a seconda del contenuto del blocco, che può accogliere dati, un inode oppure una directory.

La funzione principale del blocco directory è quella di raccogliere file che utilizzino la stessa chiave K; il blocco contiene il nome della directory stessa e la lista dei nomi dei file. h_0 è calcolato come l'hash del nome della directory, dopo averne calcolato lo XOR con la chiave K. Si noti che così facendo nomi di directory identici vengono scritti da utenti differenti in luoghi differenti; inoltre il controllo di validità si ottiene dal nome della directory, avendo h_0 e K.

Il blocco inode serve per raccogliere i collegamenti ai blocchi di dati che appartengono ad un file. Il valore $h0$ per questo blocco è calcolato dall'hash sulla directory del file concatenato con il nome

del file, e anche qui il risultato si ottiene dopo lo XOR con la chiave K. Il blocco contiene anche il nome del file, che serve al controllo di correttezza. Si noti che i blocchi directory sono opzionali alle operazioni sui file, a patto che si conosca già il nome della directory.

I blocchi inode contengono quindi una lista di puntatori ai blocchi di dati; ogni elemento contiene una coppia di valori da 256 bit: il valore di h_0 per il blocco dati e l'hash sul contenuto del blocco dati, da utilizzarsi per il controllo di validità.

Contrariamente all'implementazione di StegFS, il sistema descritto è del tutto piatto in quanto non vi è un elemento più importante che gestisce l'ubicazione dei blocchi; Mnemosyne è quindi scalabile a piacere e del tutto decentrato. Inoltre la replicazione permette di non dover fidarsi di tutto il dispositivo di memorizzazione, ma basta che la gran parte delle volte si comporti correttamente.

3.2 Steganografia distribuita

Lo schema descritto si estende facilmente all'utilizzo distribuito con Tapestry. Infatti il numero di blocchi totali del sistema ora è dato dalla somma dei nodi per il numero di blocchi disponibili per nodo; cambia il meccanismo per determinare la posizione del blocco i-esimo partendo dal suo hash h_i. Dato un insieme di M nodi Tapestry, supponendo che ognuno fornisca lo stesso spazio (per esempio 1 GB composto da 2^{20} blocchi da 1 KB), il modo più semplice per operare è riservare i primi 160 bit di h_i all'identificazione del nodo Tapestry M_i, ed per esempio i successivi 20 bit come b_i su quel nodo. Quindi quando un nodo vuole leggere un blocco, parte da h_0, ottenendo M_0 e b_0. Poi prova le altre copie del blocco fino a $h_n - 1$, utilizzando $M_n - 1$ e $b_n - 1$. Le richieste possono essere effettuate in parallelo; il primo blocco che arriva ed è corretto viene utilizzato.

Le primitive di accesso ai file sono due, lettura e scrittura; come accennato prima richiedono solo l'informazione sull'ubicazione del blocco. Tutta la parte di verifica del blocco ed estrapolazione di puntatori e nomi viene fatta dall'utente. Le informazioni in Mnemosyne sono nascoste a due livelli, in quanto sia i blocchi sono sparsi su diversi nodi non noti a priori, sia il blocco è all'interno di una lista di blocchi dai contenuti egualmente probabili.

Si noti che se un nodo vuole offrire meno spazio degli altri, deve mappare i bit di indirizzo sul suo numero ridotto di bit di indirizzo utilizzabili; se invece vuole offrire maggiore spazio, deve utilizzare un altro identificativo Tapestry.

3.3 Considerazioni e vulnerabilità

Il modello descritto è stato migliorato, in quanto il meccanismo di replicazione rimane il punto debole del sistema, poiché è troppo semplicistico. Si utilizza quindi un algoritmo di dispersione di informazioni (*Information Dispersal Algorithm*, IDA) che garantisce una migliore distribuzione dei blocchi e quindi anche una minore sovrascrittura dei dati.

L'adozione di un IDA costringe però all'utilizzo di un controllo di validità dei blocchi basato su cifratura a blocchi (in particolare si utilizza OCB); è quindi possibile aggirare un attaccante che controlla le richieste di un nodo tramite analisi passiva del traffico, richiedendo dei blocchi identificati dall'IDA che non c'entrano nulla con i dati voluti; i blocchi indesiderati saranno scartati all'arrivo grazie ad OCB.

Rimane l'inconveniente che ogni lettore di un file può modificarlo oppure cancellarlo. Per aggirare il problema, con l'IDA è possibile fornire al lettore l'indicazione di una parte minima delle repliche dei blocchi, per cui l'autore che è l'unico a conoscere tutte le locazioni dei blocchi non perde il contenuto.

Un altro problema è la necessità di effettuare un refresh dei vecchi file. Infatti più vengono aggiunti blocchi nuovi, più è possibile perdere il contenuto dei vecchi blocchi utili mai più riscritti. Perciò è

necessario decidere un periodo di rinfresco dei file, tuttavia difficile da stabilire perché l'utente non conosce l'attività generale del sistema. Inoltre nasce un'altra questione sulla vulnerabilità all'analisi del traffico, in quanto è possibile localizzare i blocchi aggiornati. Riscrivere blocchi non propri per confondere l'attaccante non è una soluzione efficace poiché può generare collisioni.

L'implementazione attuale è sperimentale e non è liberamente ottenibile; la parte client è scritta in linguaggio C ed utilizza SHA-256 e OCB-AES, mentre l'IDA è un polinomio con 2^{16} gradi di libertà. La parte server è scritta in Java e si basa su Tapestry, utilizzando UDP. Le prestazioni raggiunte in locale sono dell'ordine di 80 KB/s in scrittura e 160 KB/s in lettura.

Capitolo 4

Free Haven

Il progetto Free Haven [28] [29] nasce con lo scopo di realizzare un sistema distribuito anonimo di memorizzazione persistente; tra i suoi obiettivi principali vi è l'anonimato delle operazioni effettuate e dei contenuti dello storage, realizzati tramite una rete peer-to-peer in cui tutti i nodi collaborano in modo paritario. I nodi paritari offrono uno spazio di memorizzazione, in cui i dati contenuti sono frequentemente scambiati con gli altri; per migliorare l'affidabilità della rete è presente un sofisticato meccanismo di reputazione sulle transazioni. La riservatezza è più importante delle prestazioni del sistema, in quanto il suo scopo ultimo è resistere alla censura dei contenuti attuata da avversari potenti.

L'anonimato della rete Free Haven è in gran parte assicurato dall'utilizzo di canali di comunicazione basati su meccanismi di rete a cifratura multipla, come quello offerto dalla rete Mixminion oppure dalla rete Tor (The Onion Routing). La rete Mixminion offre un sistema per la trasmissione anonima di messaggi di posta elettronica; è l'implementazione del terzo tipo di remailer anonimi. L'anonimato delle comunicazioni è assicurato dal transito del messaggio attraverso una serie casuale di nodi remailer, dove ad ogni trasmissione il dato è decifrato con differenti chiavi asimmetriche. La rete Tor lavora in modo simile ma più sofisticato, opera a livello TCP e permette prestazioni nettamente migliori. Infatti Mixminion ha troppe latenze per essere impiegato efficacemente come livello di rete di uno storage distribuito; funziona bene con i messaggi di posta elettronica, che non hanno particolari esigenze di performance.

Il progetto Free Haven è iniziato nel dicembre 1999 presso il Massachussets Institute of Technology da Roger Dingledine e Nick Mathewson e ha ricevuto negli anni collaborazioni e supporto dall'US Navy e dall'Electronic Frontier Foundation. Le ultime attività riguardanti lo sviluppo vero e proprio della rete Free Haven sono datate dicembre 2004; più che alla progettazione dello storage stesso, il progetto Free Haven è stato molto utile allo sviluppo di Mixminion e Tor; attualmente è attivo sul fronte Tor. In appendice A e B si trovano le trattazioni esaustive dei sistemi Mixminion e Tor, con le relative analisi di sicurezza. Questo capitolo descrive quindi il funzionamento e le caratteristiche progettuali di Free Haven, analizzandone gli aspetti relativi alla sicurezza; si dà per scontata la conosenza del funzionamento del livello di rete sottostante per la comunicazione anonima.

4.1 Principi e funzionamento

Seppur implementato solo nel livello di trasporto delle informazioni, l'intero progetto Free Haven è sempre stato definito per quanto riguarda le sue caratteristiche. Gli autori hanno ritenuto infatti inutile implementare il sistema finché vi sono incertezze nella progettazione, esaminate alla fine del capitolo.

I principi che hanno guidato lo sviluppo del progetto Free Haven sono:

- Robustezza e anonimato, ottenuti grazie alla dispersione dei contenuto ed ad un livello di comunicazione anonima.
- Affidabilità, data da una rete di fiducia decentralizzata.
- Semplicità e modularità, in quanto i sistemi complessi portano sempre a debolezze di sicurezza. Questi concetti hanno favorito la definizione e lo sviluppo di Tor.
- Flessibilità, poiché l'operatore di ogni nodo può decidere quanto essere paranoico o fiducioso.
- Neutralità del contenuto. La popolarità o l'opinione su un contenuto non influenzano la permanenza sulla rete.
- Free e Open Source, visto che la ridistribuzione del codice e l'incoraggiamento alle modifiche favoriscono la scoperta di vulnerabilità e il miglioriamento del programma.

Il progetto si pone come strumento per impedire la censura o l'individuazione degli autori e dei fruitori dei documenti; l'attenzione è centrata più sullo sviluppo di meccanismi di riservatezza e di permanenza delle informazioni rispetto alle prestazioni, intese sia come tempi di accesso che banda occupata. La rete Free Haven evita la censura partendo dalla considerazione che le informazioni censurate in alcune zone tipicamente non lo sono in molte altre; favorendo lo scambio frequente dei dati cifrati fra nodi localizzati in molti luoghi diversi impedisce l'individuazione esatta dei contenuti sensibili. Il sistema essendo poco efficiente è pensato per memorizzare i contenuti in modo affidabile, per poi renderli disponibili tramite canali convenzionali, come per esempio pagine web situate in locazioni dove tali informazioni non sono soggette a censura.

Un file sulla rete Free Haven non è presente come tale, ma per assicurare robustezza è disperso fra i nodi utilizzando un *Information Dispersal Algorithm* (IDA). Tale algoritmo prevede che dal file da pubblicare si ottengano N parti, in modo tale che almeno un sottoinsieme k di tutte le parti permetta la ricostruzione corretta del dato. In particolare si è pensato all'IDA ideato da Rabin [30]; esso offre il *secret share*, in quanto con $i < k$ parti non si può ricostruire il messaggio. Questo fatto è importante in quanto garantisce la negazione plausibile di responsabilità (*plausible denyability*) per un nodo che contiene le parti di un contenuto. È possibile scegliere il livello di robustezza, dato da $r = N/k$, variando il valore di k, in quanto un valore piccolo rispetto ad N implica una maggiore robustezza, mentre un valore di k elevato significa minor spazio occupato.

4.1.1 Operazioni

Free Haven si compone di una rete di nodi peer-to-peer che offrono spazio di memorizzazione, ricevendone in cambio. Tuttavia l'autore o chi pubblica un contenuto non è costretto a mantenere un nodo, ma può contattare un nodo a sua scelta chiedendogli di pubblicare il contenuto sulla rete. Si possono quindi distinguere due livelli gerarchici, i *client* e la *servnet*.

L'identità di un client è nascosta grazie alla comunicazione tramite Mixminion o Tor con il nodo. Esso è scelto senza un criterio preciso, deve però essere predisposto con un reply block pubblico ed essere disponibile a pubblicare i dati di utenti anonimi. Per il client tutti i nodi sono uguali; per l'estrazione di un contenuto il client invece si interfaccia con tutti i nodi, in quanto ottenere un contenuto implica una ricerca in broadcast su tutta la servnet.

Un nodo della servnet invece si raffronta con gli altri nodi in modo paritario; d'ora in avanti si considereranno client e nodo come un'unica entità, in quanto è un nodo che effettivamente pubblica ed estrae i contenuti.

Le operazioni fondamentali sulla rete sono:

- Inserimento nella rete di un contenuto da parte di un utente anonimo.
- Estrazione di un contenuto con modalità anonime, che incorpora un meccanismo di verifica di autenticità ed integrità del contenuto.
- Meccanismo di scadenza dei documenti. All'inserimento deve essere imposta la durata della permanenza, durante la quale esso è immutabile.
- Meccanismo per l'aggiunta di nodi alla rete senza provocarne rallentamenti o disfunzioni.
- Meccanismo per individuare ed escludere dal sistema i nodi inattivi o malevoli.

4.1.1.1 Inserimento

Un nodo che vuole pubblicare un contenuto, deve per prima cosa creare le N parti ottenute con l'IDA. Quindi deve procedere generando una coppia di chiavi asimmetriche $K_p[dato]$ e $K_s[dato]$ da utilizzare per firmare le parti. Ogni frammento è presente sulla rete come un blocco di dati contenente la parte stessa, un hash della chiave pubblica $HASH(K_p[dato])$ per provare l'autenticità del dato (*message digest*), un timestamp con informazione sulla scadenza, il numero della parte dato dall'IDA e naturalmente la firma ottenuta con la chiave $K_s[dato]$ per verificare l'integrità.

Le parti di un file sono tenute nello spazio di memorizzazione locale. Per introdurre un nuovo contenuto è allora necessario incrementare il proprio spazio di memorizzazione condiviso in modo da poterlo contenere localmente. Il nodo chiede poi ad altri nodi scelti di memorizzare le parti che contiene. Avvengono quindi degli scambi di frammenti fra i nodi; questi scambi sono continui e frequenti, permettendo la dispersione statistica delle parti sulla rete; gli aspetti di una rete dinamica verranno analizzati nella sezione 4.1.2. L'algoritmo di IDA deve essere robusto, in quanto la perdita di fino a metà dei nodi non deve compromettere la presenza dei documenti. Inoltre si assicura che lo spazio totale presente su Free Haven è sufficiente dato che un nodo per immettere un documento, deve anche mettere a disposizione lo spazio necessario a contenerlo.

4.1.1.2 Estrazione

Quando un utilizzatore del sistema (client o nodo che sia) vuole accedere ad un oggetto, deve effettuare una richiesta a più nodi, possibilmente fidati; la richiesta contiene una locazione ed una chiave da utilizzarsi per la consegna dell'oggetto ricercato con modalità anonima. Nel dettaglio, la ricerca si effettua sull'hash della chiave $K_p[dato]$ utilizzata per firmare le parti. Il richiedente deve generare una coppia di chiavi asimmetriche $K_p[nodo]$ e $K_s[nodo]$ per la ricerca, ed un *reply block* costruito ad-hoc.

> Per il funzionamento del reply block si rimanda all'appendice A e B; in breve esso contiene una serie di livelli di cifrature racchiusi l'uno nell'altro che permettono di rispondere ad un messaggio anonimo senza conoscerne il mittente.

La richiesta si compone quindi della chiave $K_p[nodo]$, dell'hash della chiave $K_p[dato]$ e del reply block. Essa viene trasmessa ai nodi fidati; il nodo a cui è stata fatta la richiesta deve ritrasmetterla ai nodi da lui conosciuti, e così via. Le richieste essendo broadcast possono occupare una notevole banda; possono però essere accumulate ed inviate in blocco in momenti più opportuni.

Ogni nodo contattato della richiesta verifica se ha delle parti con l'hash uguale a quello della chiave del dato cercato ($HASH(K_p[dato])$), nel qual caso cifrano ogni parte con la chiave $K_p[nodo]$ inviata e la trasmettono con il reply block. Quando almeno k parti arrivano al nodo destinazione, è possibile ricostruire il contenuto desiderato.

4.1.1.3 Scadenza

Assieme ad ogni frammento è presente un timestamp ed un'indicazione di scadenza; essa è un riferimento assoluto del momento dopo il quale un nodo può cancellare senza problemi il dato. Si noti che diversamente da Freenet (capitolo 5), qui è l'utente che nel momento dell'inserimento del contenuto ne decide la scadenza, indipendentemente dalla popolarità del dato.

4.1.1.4 Gestione dei nodi

Un nodo che vuole far parte della rete inizialmente non conosce altri peer. Per questo alcuni nodi possono servire da 'iniziatore'; devono avere una buona reputazione e devono essere disposti a notificare agli altri peer che conoscono l'esistenza del nuovo nodo.

Inoltre i nodi possono considerare come dormienti nodi che non rispondono ai messaggi. In tal caso essi non vengono utilizzati per gli scambi e le ricerche.

4.1.2 Dinamicità della rete

Questo sistema si basa sulla fiducia fra i nodi data dalla loro *reputazione* [31], argomento approfondito nella sezione 4.2.1: un nodo che scarta sistematicamente gli oggetti prima della loro scadenza non sarà in grado di soddisfare le richieste come gli altri nodi e tendenzialmente verrà considerato poco affidabile dal sistema. Per reputazione di un nodo s'intende il suo grado di onestà calcolato secondo il parere dei nodi che hanno completato delle transazioni con esso. Ogni nodo mantiene quindi una lista di nodi fidati a cui inviare gli oggetti e trasmettere le ricerche; questa lista è continuamente aggiornata secondo dei punteggi attribuiti internamente ai nodi conosciuti. Le liste interne con i giudizi sono scambiate con gli altri nodi. È possibile variare il livello di fiducia minima richiesta scegliendo in modo più o meno generoso dalla lista interna i nodi con cui comunicare.

In questa rete quindi i nodi adiacenti o vicini sono i nodi fidati, indifferentemente dai tempi o dalla banda del link tra due nodi: si realizza una rete di fiducia, in cui un nodo ostile è progressivamente escluso.

Il meccanismo di scambio dei dati è centrale al funzionamento di Free Haven. I nodi non accettano qualsiasi scambio, viene stipulata una sorta di *contratto*, in cui una coppia di nodi si promettono di ospitare i dati scambiati. Ogni contratto concluso con successo incrementa la reputazione di un nodo, mentre un nodo disonesto che scarta i dati prima della scadenza o non rispetta il meccanismo di contratto la decresce. Un nodo con buona reputazione aumenta la sua probabilità di poter scambiare i suoi dati con gli altri peer.

La frequenza degli scambi interni di dati è un parametro deciso dall'amministratore del nodo. Frequenti scambi permettono maggiore anonimato, dato che non è possibile distinguere tra scambi dovuti all'immissione di nuovi dati e scambi interni regolari. Inoltre permettono di escludere in modo rapido i nodi disonesti, in quanto si hanno frequenti contratti con conseguenti report di fiducia dai nodi. Si noti infine che una rete dinamica quale è la servnet di Free Haven implica che per un attaccante (che vuole colpire l'anonimato od i contenuti) non c'è mai un unico specifico e statico obiettivo ed è costretto ad 'inseguire' i dati.

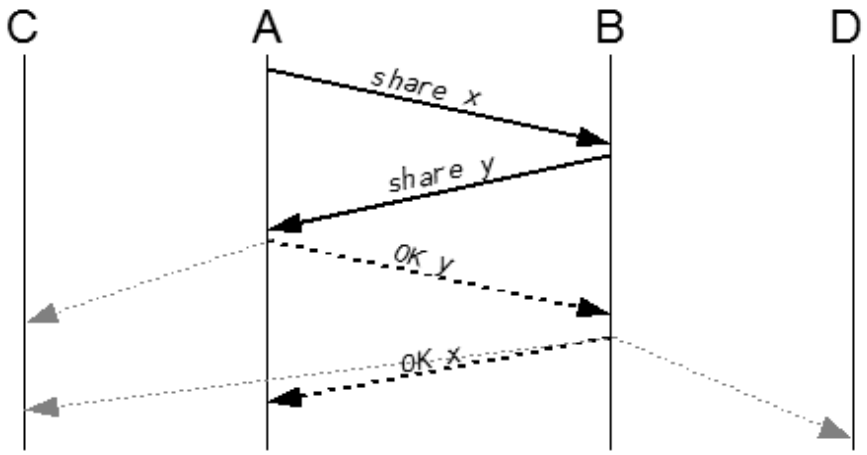

Figura 4.1: Four-way handshake

4.1.2.1 Four-way handshake

Si esamina ora come funziona il protocollo d'intesa per lo scambio di un dato. Come presentato in figura 4.1, il nodo A vuole provare uno scambio con il nodo B, scelto nella sua lista di nodi conosciuti fra quelli con buona reputazione. A invia quindi il blocco di dati contenente la parte di file con i suoi attributi, aspettandosi come risposta da B un altro blocco di dati. I blocchi di dati scambiati dovrebbero avere circa lo stesso valore. Una proposta di valore presente in [32] è il prodotto fra dimensione e tempo prima della scadenza; tuttavia dato il suo costo la dimensione è meno importante rispetto al tempo. Lo scambio non deve per forza avvenire, in quanto B può sia non ricevere il messaggio sia non avere un dato con un valore simile.

Il nodo B risponde con un suo blocco. Per completare la transazione, alla ricezione il nodo A deve inviare una ricevuta dell'avvenuto scambio a B. Il nodo B dovrebbe rispondere ad A con la sua ricevuta; tuttavia il nodo B potrebbe essere disonesto e a questo punto scartare il dato di A senza rispondere. Il nodo A è autorizzato a scartare il vecchio dato e tenere quello di B, in quanto la sua transazione è terminata dopo aver inviato la sua ricevuta; non ricevendo la ricevuta da B invia ai nodi conosciuti un reclamo per abbassare la sua reputazione. Più robustezza potrebbe essere aggiunta facendo in modo che il nodo A tenga il dato inviato fino alla ricezione della ricevuta di B; ciò ha però delle controindicazioni per esempio nel caso di collegamenti malfunzionanti che potrebbero portare al raddoppio dello spazio richiesto.

Il primo scambio di frammenti di dati ed il secondo scambio di ricevute prendono il nome di *four-way handshake*. Si noti che le ricevute non implicano la buona conclusione della transazione, ma solo della sua metà. Esse vengono diffuse non solo al nodo che partecipa allo scambio, ma anche agli altri nodi conosciuti; nell'esempio, i nodi che ricevono sia la ricevuta sia il reclamo da A hanno quindi una doppia conferma della disonestà di B. La reputazione gioca un ruolo molto importante in questi scambi, in quanto è il fattore che favorisce l'onestà dei nodi nell'ultima ricevuta.

Le ricevute sono composte dall'identificativo dei due nodi, dagli attributi ($HASH(K_p[dato])$, numero di parte, timestamp) delle due parti scambiate e dal timestamp dello scambio. Esse sono memorizzate a lungo dai nodi, come appunto ricevute della transazione.

È stato proposto un meccanismo (*buddy system*) per il quale i frammenti di un contenuto a due a due si 'cercano' l'un l'altro, verificandone la presenza. In caso di parte corripondente non trovata

un nodo genera un reclamo. Questo meccanismo potrebbe aumentare la robustezza sia in caso di nodo che scarta prematuramente i dati, sia in caso di four-way handshake fallito. Le ricevute vengono inviate ai nodi contenenti la parte associata, aumentando la cattiva reputazione del nodo disonesto. Con l'algoritmo di IDA utilizzato la parte associata ad una persa potrebbe attivamente creare una nuova parte così da rigenerare la coppia; anche questa soluzione è stata però scartata per evitare di congestionare la rete a causa di problemi del canale di comunicazione.

4.2 Anonimato in rete

Free Haven vuole essere uno storage permanente *anonimo*. L'anonimato può essere presente a differenti livelli; in questa rete quasi tutte le comunicazioni sono anonime, poiché include le seguenti proprietà:

- gli oggetti pubblicati sulla rete sono immessi da utenti che non devono poter essere rintracciati;
- gli utenti che recuperano gli oggetti presenti non devono doversi identificare in alcun modo;
- la posizione attuale di un oggetto non deve essere nota ad alcuno;
- un nodo della rete non conosce il contenuto dei suoi oggetti;
- le transizioni fra i nodi della rete devono essere effettuate in modo tale da fornire il minimo possibile di informazioni ad un osservatore esterno.

In una rete distribuita l'anonimato non è semplice da ottenere come la cifratura di un dato: non è un accordo dove bastano i due enti che fanno parte della comunicazione, ma è necessario avere un numero (il più vasto possibile) di enti intermediari in modo tale da far perdere le tracce del mittente e del destinatario. Tutti i sistemi di comunicazione realmente anonima, che quindi non affidano l'identità degli utenti ad un presunto server fidato (ad esempio un proxy anonimizzante), hanno come requisito una rete eterogenea di nodi che possano selezionare uno qualsiasi degli altri nodi per la trasmissione del messaggio.

L'anonimato prevede l'attribuzione di uno pseudonimo. Lo pseudonimo e la vera identità di un soggetto o nodo devono essere scorrelate. Per identità si intende sia il nome dell'utente, sia l'indirizzo IP e la locazione geografica di un nodo.

In Free Haven l'anonimato di chi pubblica è garantito dal fatto che sia l'autore che il nodo d'ingresso sono sconosciuti: l'autore grazie alla comunicazione basata sui remailer e il nodo d'ingresso poiché i continui scambi con gli altri nodi lo nascondono. L'anonimato di un lettore è protetto dalla comunicazione con il *reply block*. La posizione di un contenuto è sconosciuta data la sua scomposizione tramite l'IDA ed i continui scambi; l'IDA garantisce anche la negazione plausibile di responsabilità. Le transazioni fra i nodi sono rese anonime e poco correlabili ad un osservatore esterno grazie all'utilizzo di un canale di comunicazione anonimo; i sistemi che Free Haven può utilizzare sono Mixminion in modalità Forward Message (vedi sezione A.2 in appendice) oppure la rete Tor (vedi appendice B).

4.2.1 La reputazione ed i suoi limiti

Si noti che le transazioni tipicamente sono inaffidabili poiché non si conosce il destinatario; poche sono le transazioni fra gli stessi due nodi. Per migliorare il grado di affidabilità del destinatario di una transazione deve esistere una sorta di accordo per verificare l'onestà degli altri nodi, in modo da escludere quelli disonesti. Ciò implica o di poter conoscere lo stato di tutta la rete, impossibile dato l'anonimato richiesto, o di avere un modo per fidarsi del destinatario. Il meccanismo di Free Haven per

migliorare l'affidabilità della rete si basa sulla reputazione dei nodi, in quanto è un criterio di giudizio che mantiene l'anonimato del sistema. In Free Haven, tramite la lista di nodi fidati, la reputazione di un nodo viene diffusa per la rete.

L'anonimato e la reputazione potrebbero sembrare proprietà contrastanti. Tuttavia è stato osservato che il sistema di fiducia fondato sulla reputazione degli pseudonimi funziona, si guardi ad esempio ai servizi web quali eBay, Yahoo! Auction ed ai commenti di Amazon [33].

La reputazione è però un elemento critico nei sistemi anonimi. Prima di tutto in un ambiente anonimo e così dinamico, è comunque difficile essere certi della disonestà di un nodo. Alcuni accorgimenti potrebbero migliorare il sistema, ad esempio il *Buddy system*, però implicano tutti l'aggiunta di notevole complessità ed overhead su una rete già non prestazionalmente buona.

Un'altra questione riguarda eventuali reclami ingestibili: se un nodo onesto viene maliziosamente screditato da un nodo malevolo, non è possibile verificare l'esattezza di un reclamo da parte del nodo erroneamente screditato. Inoltre un attaccante che possiede abbastanza nodi può cercare di screditare gli altri nodi ed accreditare i propri; questo potrebbe essere opportuno per condurre attacchi passivi sui dati, perché permette di esaminare una maggior quantità di traffico.

Il meccanismo di reputazione ha inoltre alcuni problemi di anonimato. Un nodo non può verificare tutti i nodi e, se ne misura alcuni, rende noto ad un osservatore esterno l'informazione su quali nodi conosce; per questo in Free Haven l'elenco di nodi fidati è continuamente scambiato con gli altri e l'affidabilità di un nodo nell'elenco è aggiornata dalle informazioni passivamente ricevute dalla rete. Avere uno o alcuni nodi centrali che gestiscono la reputazione dei nodi non risolve il problema poiché si torna alla situazione di partenza in quanto ci si lega ad un presunto nodo fidato che può rispondere a suo piacimento ad ogni richiesta.

Gli attacchi atti a modificare la reputazione di un nodo possono principalmente avere come obiettivo l'accumulo di notorietà o lo screditamento di un nodo attaccato. Nel secondo caso si può interpretare come un attacco di tipo DoS: il nodo attaccato assume una cattiva reputazione, diventa isolato dalla rete che è in effetti ragionevole al fine di proteggere l'intero sistema. La rete Free Haven risponde quindi automaticamente ad un attacco di questo tipo considerando un nodo attaccato non affidabile.

Vi sono altri attacchi possibili contro il sistema di reputazione:

- Tradimento. Un nodo ostile che fa parte della rete accumula una buona reputazione per poi cancellare tutti i dati che contiene lasciando sulla rete i suoi file. In verità un attacco di questo tipo non può avvenire in quanto per accumulare abbastanza credito per memorizzare i suoi dati deve mantenerne di conseguenza. Un nodo che agisce in questo modo non nuoce alla rete, ma in realtà le giova.

- Manomissione delle ricevute degli scambi. Le ricevute sono firmate ed includono un timestamp. Tuttavia un attaccante che dispone di più nodi potrebbe effettuare manomissioni della reputazione di un nodo inviando delle ricevute ad-hoc.

La sezione 6.2.2 approfondisce le problematiche legate alla reputazione.

4.3 Vulnerabilità

4.3.1 Anonimato

Diversi tipi di attacchi possono essere condotti al fine di rivelare l'identità di alcune entità della rete:

- Identità di chi estrae.
  Un attaccante potrebbe sviluppare e pubblicare sulla rete P2P un malware che contatta un server ostile, il quale registra le connessioni. Questo malware potrebbe semplicemente essere nascosto, per esempio in un documento ipertestuale, ed attivarsi al momento dell'apertura del documento.

  Un nodo della rete Free Haven che fa parte anche della rete del canale di comunicazione (per esempio Tor), potrebbe cercare di correlare le connessioni per individuare l'identità dei nodi, per esempio tenendo traccia delle tempistiche del four-way handshake.

  Un nodo sofisticato potrebbe cercare di correlare i contenuti estratti da un utente e costruire dei profili statistici e cercarne la corrispondenza con individui fuori da Free Haven. Questo problema dovrebbe essere risolto dall'utilizzo di reply block sempre differenti per ogni estrazione.

- Identità del nodo.
  Un nodo attaccante può creare delle parti eccezionalmente grandi e cercare quindi di ridurre il numero di nodi che possono permettersi lo scambio di moli di dati così consistenti. Questo è un problema di *anonimato parziale*, in quanto non è possibile essere certi di una corrispondenza ma vi è solo un sospetto dato dal numero esiguo di possibilità.

  Sempre un nodo malevolo potrebbe immagazzinare tutte le informazioni sull'entrata ed uscita dei nodi dalla rete, inferendole dalle liste di fiducia dagli altri nodi; avendo un'ottima conoscenza delle caratteristiche di molte sottoreti di Internet, il nodo malevolo potrebbe cercare di costruire la topologia della rete Free Haven.

  Una variante della vulnerabilità precendente potrebbe essere la diffusione su Internet di un codice malevolo (come un worm) che cerca i nodi Free Haven e pubblica la lista delle parti presenti sul nodo infettato.

Riguardo l'anonimato non bisogna sottovalutare gli attacchi a livello sociale; per esempio il pagamento in cambio di qualsiasi informazione utile è una costante minaccia. Questo problema dovrebbe essere fortemente limitato dall'utilizzo di un canale di comunicazione adeguatamente anonimo.

4.3.2 I nodi ed i loro contenuti

Gli attacchi ai nodi possono avere il fine di limitare il loro funzionamento oppure eliminare i dati contenuti:

- Attacco fisico. Un nodo può semplicemente essere distrutto. L'IDA permette di scegliere il livello di resistenza a questo tipo di attacco, in quanto per la perdita dell'informazione è necessario distruggere più di $n - k$ frammenti.

- Azione legale. Se si riesce a conoscere l'identità di un nodo Free Haven, si può intraprendere un'azione legale al fine di rimuoverne i contenuti. Questo pericolo è spesso risolto dalla negazione plausibile di responsabilità, dove applicabile, in quanto l'amministratore di un nodo non ne conosce i contenuti.

- Pressione sociale. Un amministratore oppure l'intera rete Free Haven potrebbe essere citata per contenuti illegali, immorali o che violino le leggi sul copyright; un amministratore potrebbe essere quindi costretto a rimuovere tutti i contenuti del suo nodo. Per questo si suppone che la rete Free Haven sia abbastanza vasta da coprire molti Paesi con differenti giurisdizioni, tale da permettere alla rete di sopravvivere anche in mancanza di molti nodi di un Paese.

- Denial of Service. La rete di Free Haven potrebbe essere attaccata da continue ricerche di dati o richieste di ingresso nella rete, in modo da sovraccaricarla. Questo tipo di attacco non dovrebbe funzionare se il canale di comunicazione è bufferizzato, come il canale anonimo Tor o Mixminion.
- Data flooding. Si potrebbe cercare di riempire tutto lo spazio di memorizzazione. Come si è visto, questo non è permesso dal sistema di scambi.
- Accumulo di dati. Un attaccante con alcuni nodi dotati di notevole spazio può cercare di accumulare molti dati di Free Haven scambiando in modo intelligente dati spazzatura che trasferisce agli altri nodi. Si presuppone che la rete sia abbastanza vasta da rendere proibitivo un attacco di questo tipo. In caso contrario l'attaccante riesce sia ad eliminare dalla rete gran parte dei contenuti, sia a ricostruire i file.

4.4 Osservazioni

I problemi irrisolti di Free Haven relativi al meccanismo di reputazione, il primo dei quali quello dei reclami, ne hanno probabilmente frenato lo sviluppo, portando quindi gli sviluppatori a concentrarsi sul canale di comunicazione, prima Mixminion e poi Tor. Il sistema di reputazione probabilmente dovrà essere scartato per evitare troppa complessità, ripensando Free Haven da capo con un modo per accertare anonimamente l'affidabilità di un nodo senza che ci possano essere reclami.

Un ulteriore grosso problema è proprio l'efficienza, in quanto per l'utente finale medio l'usabilità è comunque più importante dell'elevato livello di anonimato. Un sistema inusabile porta ad avere una base d'utenti ridotta, il che è diametralmente opposto al requisito principale di Free Haven, un largo numero distribuito di nodi. Quindi prima di implementarla, la rete deve essere ripensata in modo da risultare maggiormente efficiente.

Capitolo 5

Freenet

Freenet è un sistema per lo storage distribuito e il prelievo di informazioni progettato per risolvere i problemi legati alla privacy degli utenti e alla sopravvivenza dei dati. Il sistema realizza un filesystem location-independent distribuito attraverso molti nodi paritetici che donano le proprie risorse hardware per permettere di inserire, memorizzare e prelevare in modo anonimo i file, ma senza garantire che i dati rimangano permanentemente nello storage.

Gli obiettivi primari di Freenet sono cinque:

- Garantire l'anonimato degli utenti che producono e consumano l'informazione.
- Permettere agli operatori dei nodi di esercitare la negazione plausibile di responsabilità (*plausible deniability*).
- Impedire la censura dei documenti ad opera di terze parti.
- Memorizzare e distribuire documenti in maniera efficiente.
- Decentralizzare tutte le funzioni di rete, eliminare qualsiasi *single point of failure* o di controllo centralizzato.

Freenet è descritta nella pubblicazione "A distributed decentralized information storage and retrieval system" redatto da Ian Clarke nel Luglio 1999 quando si trovava come studente presso l'Università di Edinburgh in Scozia. Freenet è un Free Software rilasciato sotto licenza GPL e scritto in linguaggio Java, pubblicamente distribuito per la prima volta in versione 0.1 nel Marzo 2000.

La metodologia impiegata nello sviluppo di questo programma ricalca l'architettura distribuita e decentralizzata della rete stessa: i principali componenti del team (Ian Clarke, Matthew Toseland, Oskar Sandberg, Florent Daignière e Scott Miller) collaborano da molti Paesi diversi del mondo in modo delocalizzato attraverso Internet. Il progetto fa affidamento in gran parte all'impegno volontaristico di queste persone e alle donazioni degli utenti; i finanziamenti più significativi sono stati erogati da John Gilmore dell'Electronic Frontier Foundation e da Google nell'ambito dei Summer Of Code.

Attualmente le versioni di Freenet che gli utenti possono utilizzare sono due:

- Freenet Light: versione stabile (ramo 0.5, ultima release del Settembre 2005), trattata nella sezione 5.1 di questo testo.
- Freenet Dark: versione in sviluppo (ramo 0.7, ultima release del Settembre 2006), discussa separatamente nella sezione 5.2 dal momento che costituisce un radicale cambiamento di design della rete.

Il codice sorgente di Freenet è consultabile online presso [34].

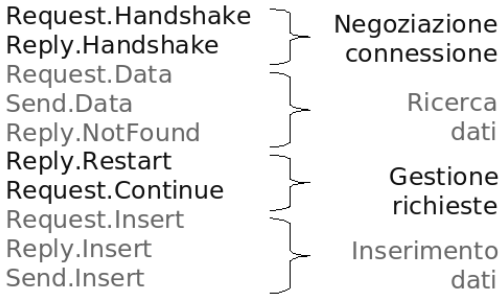

Figura 5.1: Messaggi in Freenet Light.

5.1 Freenet Light

5.1.1 Panoramica

Freenet consiste di un insieme di nodi che scambiano tra di loro messaggi. Un nodo è semplicemente un calcolatore che sta eseguendo il software Freenet e tutti i nodi della rete sono identici e trattati allo stesso modo. Ogni nodo mette a disposizione una certa quantità di spazio di memorizzazione per mantenere una propria porzione locale del data store distribuito. Gli ideatori di Freenet desiderano offrire a qualsiasi autore la possibilità di pubblicare i propri documenti a prescindere dalle sue disponibilità finanziarie e di spazio libero su disco e per questo motivo il sistema non richiede né di condividere una soglia minima di memoria di massa né alcun pagamento per fruire del servizio. Questo giustifica il fatto che Freenet non garantisce che i documenti siano presenti in modo permanente nello storage ed implementa invece una politica probabilistica per lo scarto dei file meno richiesti quando è necessario liberare spazio per l'inserimento di nuovi dati.

I nodi di Freenet Light seguono il Freenet Network Protocol (FNP) per organizzarsi spontaneamente in una configurazione di rete efficiente. In Freenet Light pertanto i nodi stabiliscono automaticamente connessioni gli uni con gli altri: una rete di questo tipo è definita Opennet. Gli indirizzi dei nodi sono composti da un metodo di trasporto ed un identificativo specifico a livello di trasporto (tipicamente un indirizzo IP e un numero di porta), ad esempio `tcp/192.168.1.1:19114`. Il FNP è un protocollo packet-oriented basato sullo scambio dei messaggi riportati in figura 5.1.

Ogni messaggio include i seguenti tre campi:

- `UniqueID`: 64 bit generati in modalità casuale che consentono ai nodi di tenere traccia dello stato degli inserimenti e delle richieste.
- `Hops-To-Live` (`HTL`): valore intero impostato dal mittente di un messaggio e decrementato a ogni hop per prevenire l'inoltro infinito dei messaggi.
- `Depth`: valore intero incrementato ad ogni hop che consente al destinatario di un messaggio di impostare nella risposta un valore di `HTL` sufficientemente alto da raggiungere il mittente.

Una transazione in Freenet ha inizio con l'invio di un messaggio `Request.Handshake` da un nodo ad un altro, specificando quale indirizzo di ritorno per il nodo mittente si desidera. Il nodo destinazione risponde con un messaggio `Reply.Handshake` in cui indica il numero di versione del protocollo. I nodi ricordano gli handshake per un certo periodo di tempo, durante il quale comunicazioni successive tra gli stessi due nodi non necessitano ripetere la fase di handshake.

I messaggi `Request.Data`, `Send.Data` e `Reply.NotFound` sono legati ai meccanismi di richiesta dei file dal data store descritti in sezione 5.1.3.

I messaggi `Reply.Insert` e `Send.Insert` permettono di pubblicare documenti come spiegato in sezione 5.1.4.

5.1.2 Protezione dei dati: chiavi

In Freenet ogni documento è identificato da una chiave binaria ottenuta applicando la funzione di hash SHA-1 a 160 bit. Freenet definisce un Uniform Resource Indicator (URI) nella forma seguente:

```
freenet:keytype@data
```

dove il dato binario è codificato usando una variante dello schema Base64. Ogni tipo di chiave interpreta a suo modo la parte *data* dell'URI, la cui spiegazione è quindi legata al *keytype*.

In Freenet Light si possono usare tre tipi di chiavi per indicizzare un documento:

- *Keyword-Signed Key* (KSK).
- *Signed-Subspace Key* (SSK).
- *Content-Hash Key* (CHK).

5.1.2.1 Keyword Signed Key (KSK)

Il tipo più semplice di chiave è la KSK, viene derivata da una breve stringa di testo facile da ricordare (ad esempio `text/books/1984.html`) scelta dall'utente al momento della pubblicazione di un file nel data store distribuito ed appare così:

```
freenet:KSK@text/books/1984.html
```

La figura 5.2 illustra il procedimento di creazione di una chiave KSK:

- si calcola l'hash SHA-1 della stringa descrittiva;
- si utilizza il valore di hash della stringa descrittiva come chiave privata;
- a partire dalla chiave privata si genera una chiave pubblica secondo il *Digital Signature Algorithm* (DSA);
- si ottiene la chiave KSK associata al file inserito calcolando l'hash SHA-1 della chiave pubblica.

La chiave privata è utilizzata per firmare il digest del file inserito, in modo da offrire un controllo minimo di integrità garantendo che un documento ritirato corrisponda alla sua chiave KSK associata. Il documento è inoltre cifrato in modalità simmetrica usando proprio la stringa descrittiva come chiave.

Per permettere agli altri di ritirare un documento, chi inserisce il file deve pubblicare la relativa stringa descrittiva; questo risulta evidente se si percorre a ritroso il diagramma di figura 5.2 dal blocco "Chiave KSK del file": è sufficiente conoscere la stringa descrittiva per calcolare la chiave KSK che identifica il file.

Nonostante il formato convenzionale di una KSK ricordi l'organizzazione in directory di un filesystem, questo non deve indurre a pensare che in Freenet vi sia una gerarchia all'interno del data store: in Freenet *non esiste una gerarchia dei documenti*.

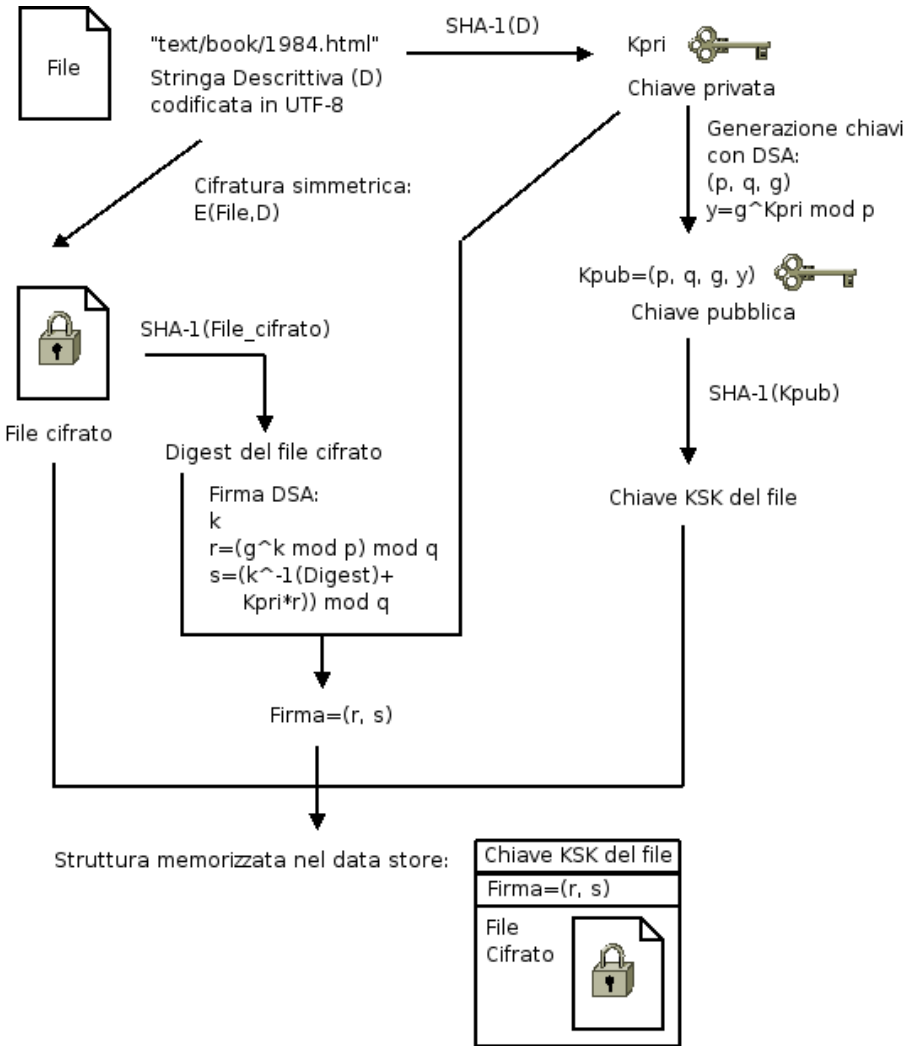

Figura 5.2: Schema di generazione di una KSK.

Un documento inserito tramite KSK apparirà sui nodi nel data store come una chiave binaria seguita da dati opachi (firma digitale e file cifrato). L'operatore di un nodo non può ricostruire la stringa di testo descrittiva dalla chiave binaria dal momento che è ottenuta calcolando l'hash della chiave pubblica DSA della stringa e l'operazione non è invertibile; d'altra parte allora non può neppure risalire al contenuto del file perché esso è simmetricamente cifrato usando per chiave la stringa descrittiva.

Le KSK sono tuttavia il tipo più debole di chiave perché la stringa di testo viene scelta per essere facilmente memorizzabile da un utente ed è formata da una concatenazione di parole di senso compiuto; un attaccante può sfruttare questa peculiarità della stringa per sferrare un attacco *brute-force*: con un nodo ostile genera molte stringhe comuni fino a quando trova quella che produce la KSK vittima ed a questo punto è in grado di inviare false risposte quando riceve richieste per quella KSK. Inoltre nulla vieta a due utenti di scegliere in modo indipendente la stessa stringa descrittiva per pubblicare file diversi e quindi un attaccante può eseguire un attacco di *key-squatting* in cui inserisce molti dati fasulli indicizzati da descrizioni popolari provocando collisioni al momento di richiesta dei documenti.

5.1.2.2 Signed-Subspace Key (SSK)

Le SSK si basano sullo stesso sistema a chiave pubblica DSA delle KSK, ma sono puramente binarie in quanto non sono derivate dalla sola stringa descrittiva human-readable.

Per risolvere i problemi propri dell'impiego delle KSK, le SSK consentono la gestione di *namespace* (detti anche *subspace*) personali per controllare un insieme di chiavi. Un utente definisce un proprio personale namespace generando in modo casuale un paio di chiavi pubblica/privata che serviranno da identificativo.

La figura 5.3 illustra il procedimento di creazione di una chiave SSK per inserire un file:

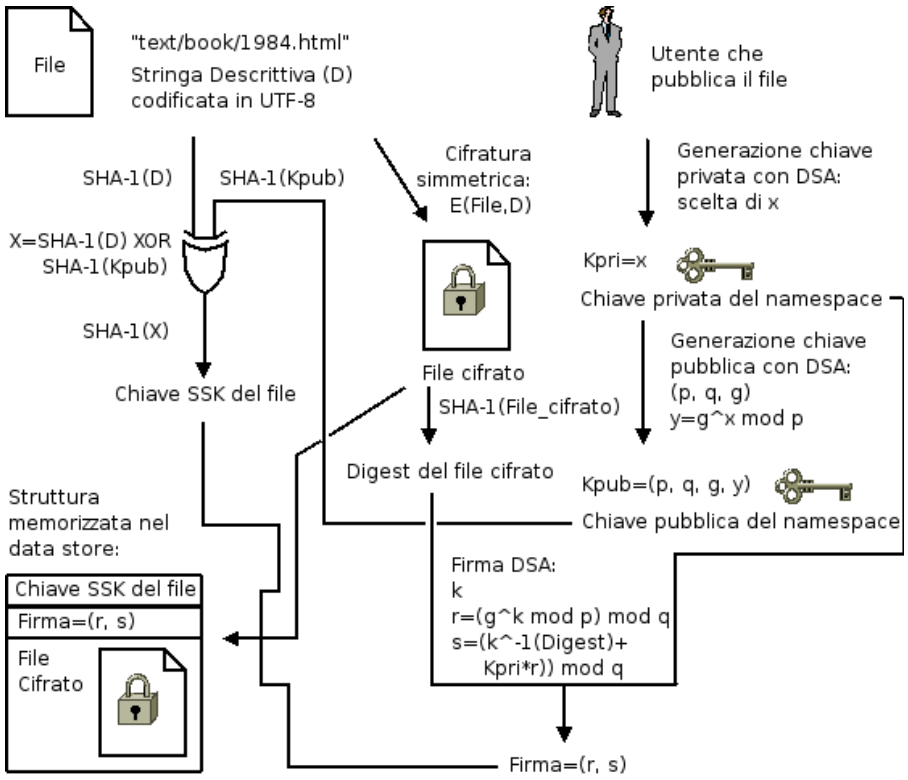

Figura 5.3: Schema di generazione di una SSK.

- si sceglie una breve stringa descrittiva come nel caso delle KSK;
- si calcolano separamente l'hash della chiave pubblica del namespace e l'hash della stringa descrittiva;
- si esegue sui due valori di hash ottenuti l'operazione XOR;
- si applica l'hash SHA-1 al risultato per ottenere la chiave binaria SSK del documento.

In analogia con le KSK, si usa la metà privata della coppia di chiavi asimmetriche per firmare il documento. Questa firma, ottenuta da un paio di chiavi generate in modalità pseudo-casuale dall'utente che pubblica il file, è più sicura di quella usata nelle KSK dal momento che in quel caso le chiavi erano derivate in modo deterministico a partire dalla sola stringa descrittiva. Il documento è anche cifrato servendosi della stringa descrittiva come chiave, come già visto per le KSK.

Per consentire agli altri di prelevare un documento, chi pubblica rende note la stringa descrittiva insieme alla chiave pubblica del subspace: percorrendo a ritroso il diagramma di figura 5.3 dal blocco "Chiave SSK del file" si vede come a partire da queste sole due informazioni sia possibile ottenere la chiave SSK che identifica il file. L'inserimento del documento invece richiede la conoscenza della chiave privata e pertanto solo il proprietario del subspace può aggiungere o modificare i file in esso contenuti.

Il meccanismo impiegato nelle SSK permette di raggiungere due obiettivi importanti. In primo luogo con le SSK un utente ora può organizzare un proprio namespace in cui simulare ad esempio una struttura gerarchica dei documenti e creare tramite un'opportuna sintassi interpretabile dai client file simili a directory che contengono metadati che puntano ad altri file. Ad esempio una directory inserita sotto la chiave `text/cryptome` potrebbe includere una lista di chiavi come `text/cryptome/freedom-of-speech`, `text/cryptome/cryptography` e `text/cryptome/surveillance` che a loro volta sono liste di chiavi che corrispondono a documenti.

Il secondo significativo risultato è conseguenza del primo: se un utente può creare un proprio sottospazio e controllare un insieme di chiavi, allora è in condizione di mantenere un proprio pseudonimo su Freenet. Quando gli utenti si fidano del proprietario di un namespace, si fidano anche dei documenti in esso contenuti e nel contempo l'anonimato di chi pubblica rimane protetto.

La possibilità di fidarsi delle pubblicazioni di un utente anonimo rende vani attacchi di *key-squatting* in cui la rete viene inondata di falsi documenti (problema presente in sistemi per il file sharing che non godono di fiducia anonima come Gnutella e Napster).

Una URI di SSK appare così:

```
freenet:SSK@U7MyLlomHrjm6443k1svLUcLWFUQAgE/text/books/1984.html
```

5.1.2.3 Content Hash Key (CHK)

La CHK di un documento è semplicemente ottenuta calcolando direttamente l'hash del contenuto del file, come illustra la figura 5.4. Dal momento che la funzione di hash è tale da rendere computazionalmente molto difficile trovare due input che forniscano lo stesso output, la chiave CHK costituisce un identificativo pseudo-unico di ogni file.

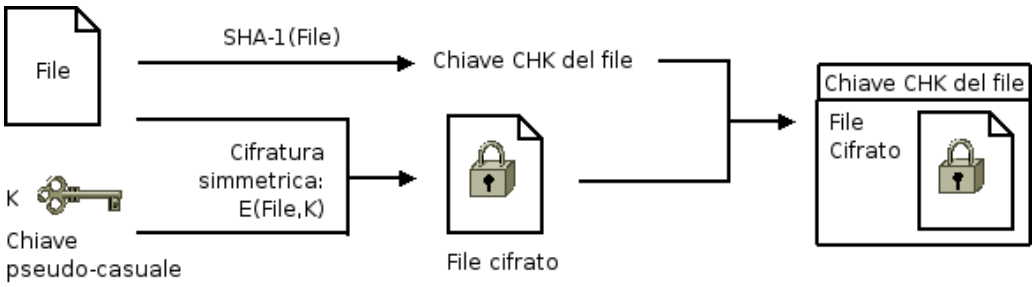

Figura 5.4: Schema di generazione di una CHK.

Il dato inoltre è cifrato con una chiave generata in modalità casuale. Affinché gli altri utenti possano prelevare e visualizzare il documento, chi inserisce il file deve pubblicare insieme alla sua CHK anche la chiave simmetrica per la decrittazione. Anche in questo caso la chiave di cifratura non viene mai memorizzata sui nodi insieme al documento, ma solo pubblicata insieme alla CHK binaria.

Dal momento che un documento è inviato in risposta ad una richiesta che include la sua CHK, un nodo è in grado di verificare l'integrità dei dati eseguendo un redundancy check tra message digest e CHK contenuta nella richiesta: se i valori di hash coincidono allora il documento è quello corretto. Le

CHK forniscono una chiave unica e inalterabile e per questo la maggior parte dei dati in Freenet sono memorizzati sotto una CHK. Le CHK inoltre riducono la ridondanza delle informazioni perché dati uguali avranno la medesima CHK e al momento dell'inserimento si verificherà una collisione.

Servendosi sia delle CHK che delle SSK si ottiene un sistema che unisce i benifici di entrambe: è possibile verificare l'integrità dei documenti ritirati e fidarsi di chi li ha pubblicati, anche se questi rimane anonimo e protetto da uno pseudonimo. Per pubblicare un documento con una CHK congiunta ad una SSK (lo stesso vale per una KSK, nel caso non interessi l'aspetto di fiducia anonima) il client è in grado di attuare un meccanismo di indirezione: pubblica il file sotto una CHK e poi inserisce una SSK che ridirige a quella CHK.

L'URI di una CHK appare così:

```
freenet:CHK@DtqiMnTj8YbhScLp1BQoW9In9C4DAQ,2jmj7l5rSwOyVb-vlWAYkA
```

5.1.2.4 Osservazioni sulle chiavi

Si è visto come tutti i documenti in Freenet sono crittografati prima dell'inserimento. La chiave è sia casuale che distribuita dal richiedente insieme all'URI (CHK) o basata su dati che il nodo non può conoscere (ad esempio la stringa di una KSK o SSK). In ogni caso un nodo non può sapere quali informazioni siano contenute in un documento. Questo ha due conseguenze: in primo luogo gli operatori dei nodi non possono impedire ai propri nodi di memorizzare in cache o inoltrare contenuti per loro indesiderabili perché non hanno nessun modo per dire che cosa contenga un documento; il secondo effetto è che l'amministratore di un nodo non può ritenersi responsabile dei dati presenti sul suo nodo (*plausible deniability*).

La verifica di integrità dei documenti è offerta sia dalla corrispondezza tra message digest del file e chiave binaria (CHK), sia dalla verifica della firma DSA (KSK e SSK).

5.1.3 Anonimato di chi accede ai dati: richieste

Un utente, per ritirare un file dal data store distribuito di Freenet, deve conoscere o calcolare la chiave binaria del file. Quindi deve inviare una richiesta in cui specifica la chiave desiderata e il numero di Hops-To-Live. Le richieste sono messaggi `Request.Data` che possono essere inoltrati attraverso numerosi nodi diversi. All'inizio un utente spedisce la richiesta a un nodo a lui noto e di cui si fida (solitamente questo nodo è in esecuzione sul suo stesso calcolatore).

Quando un nodo riceve una richiesta, controlla se possiede nel proprio data store locale il documento cercato e in caso positivo lo invia come risposta in un messaggio `Send.Data` indicando se stesso come sorgente dei dati nel campo `DataSource`. Se al contrario il nodo non dispone del documento richiesto allora cerca nella sua tabella di routing la chiave più "vicina" (in base alla distanza lessicografica) a quella desiderata e inoltra la richiesta al nodo associato a tale chiave.

In questo modo i messaggi vanno a formare una catena dal momento che ogni nodo inoltra la richiesta al prossimo nodo. Il meccanismo di conteggio degli Hops-To-Live, simile al Time-To-Live dei datagrammi IP, fa scadere i messaggi dopo aver attraversato un certo numero di nodi, così si limita la lunghezza massima di una catena. La catena di messaggi termina quando un messaggio scade oppure quando un nodo risponde col documento richiesto.

Se la richiesta viene completata con successo, il documento viene inviato percorrendo a ritroso la catena formata dai nodi che avevano precedentemente inoltrato la richiesta, fino a raggiungere il nodo iniziale. Ogni nodo della catena può memorizzare una copia della risposta nella sua porzione locale del data store, così potrà rispondere immediatamente a qualsiasi futura richiesta per quello specifico documento; inoltre ogni nodo inserisce nella sua tabella di routing una nuova entry in cui

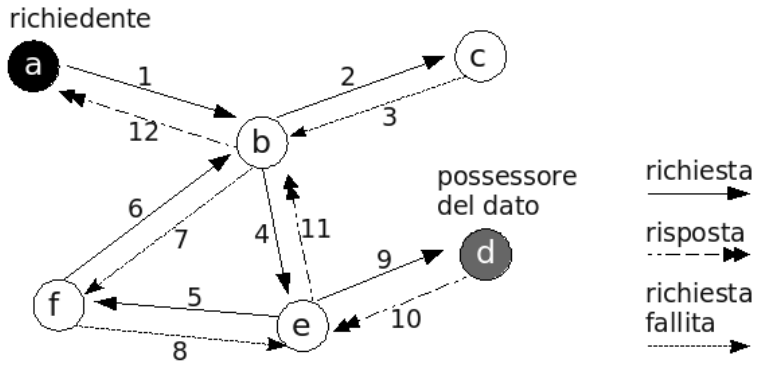

Figura 5.5: Routing in Freenet

associa la chiave richiesta con l'attuale sorgente di risposta. Questo significa che i documenti ritirati più frequentemente sono memorizzati nelle cache di molti nodi e si evita un sovraccarico dell'unico server che inizialmente ha pubblicato il documento richiesto.

Questo sistema fa sì che una richiesta successiva per la medesima chiave venga soddisfatta più rapidamente (perché il documento è ora presente in molti data store locali); richieste di chiavi "vicine" (ovvero di ridotta distanza lessicografica) verranno instradate con probabilità alta verso una medesima sorgente di dati.

La risposta ad una richiesta contiene un indirizzo di uno dei nodi che ha attraversato, così col tempo i nodi possono acquisire informazioni a proposito di altri nodi. Questo significa che l'interconnessione dei nodi della rete Freenet aumenta col trascorrere del tempo. In questo modo a un nodo può accadere di ricevere dati da un nodo sconosciuto: ogni nodo ottiene informazioni solo riguardo ai nodi con cui scambia messaggi direttamente e al più un altro nodo della catena, ma nel momento in cui riceve una risposta ad una richiesta è possibile che non sia ancora a conoscenza del nodo che risponde. Tuttavia mantenere sui nodi una tabella delle sorgenti dei dati costituirebbe un potenziale rischio per la sicurezza e per evitarlo ogni nodo della catena può arbitrariamente decidere di cambiare il messaggio di risposta per dichiarare sé stesso o un altro nodo scelto a caso come sorgente dei dati.

Se un nodo non può inoltrare una richiesta al nodo associato alla chiave più vicina perché la destinazione è irraggiungibile o si verrebbe a creare un ciclo infinito, proverà a inviare al nodo la cui chiave è seconda in ordine di vicinanza, eventualmente al terzo e via di seguito. Se un nodo esaurisce tutti i candidati a cui inviare la richiesta, riporterà in un messaggio `Reply.NotFound` la notifica di fallimento al nodo che lo precede nella catena, il quale tenterà di inoltrare la richiesta al suo secondo nodo più vicino alla chiave desiderata, e così via. In questo modo, il meccanismo di richiesta opera come una ricerca *steepest-ascent-hill-climbing* [35] con backtracking. Se si eccede il limite di Hops-To-Live, viene propagata a ritroso per tutta la catena una notifica di fallimento fino all'origine della richiesta, senza provare altri nodi.

Il sistema di routing di Freenet basato sulle chiavi è sintetizzato nella figura 5.5, dove viene illustrata una tipica sequenza di messaggi di richiesta. Un utente avvia una richiesta al nodo a. Il nodo a inoltra la richiesta al nodo b, che a sua volta la spedisce al nodo c. Il nodo c non è in grado di contattare altri nodi e risponde a b con un messaggio di fallimento. Il nodo b a questo punto tenta con la sua seconda scelta, e, che inoltra la richiesta a f. Il nodo f invia la richiesta a b, che rileva il loop e ritorna un messaggio di fallimento. Il nodo f non può contattare alcun altro nodo e restituisce il controllo ad e. Il nodo e inoltra la richiesta alla sua seconda scelta, d, che possiede il dato sul proprio data store locale. Il documento è inviato in risposta da d attraverso e e b fino ad a, il quale la spedisce all'utente.

Il dato a questo punto è stato memorizzato anche nelle cache di e, b ed a.

Dal momento che un nodo non è in grado di vedere il cammino percorso da una richiesta al di là del nodo che gliel'ha direttamente inoltrata, è molto difficile individuare la persona che ha avviato la richiesta. Questo meccanismo di inoltro e caching tra i nodi offre pertanto l'anonimato per gli utenti che ritirano documenti da Freenet.

5.1.4 Anonimato di chi pubblica i dati: inserimenti

L'inserimento di un documento in Freenet segue una strategia parallela a quella delle richieste. Per inserire un file, l'utente prima calcola una corrispondente chiave binaria applicando una delle procedure descritte nella sezione 5.1.2. Quindi invia un messaggio di inserimento `Request.Insert` al nodo in esecuzione sul proprio calcolatore in cui specifica la chiave binaria proposta e un certo valore per il campo `HTL` (questo indicherà il numero di nodi in cui si vuole memorizzare il documento).

Quando un nodo riceve una proposta di inserimento, per prima cosa verifica se nel proprio data store locale la chiave è già utilizzata. Se trova la chiave binaria, allora il nodo manda per risposta il file preesistente per quella chiave in un messaggio `Send.Data`, proprio come se avesse ricevuto una richiesta per quel documento. In questo modo l'utente viene a sapere che si è verificata una collisione e può riprovare usando una chiave diversa. Se il nodo che ha ricevuto la richiesta non ha invece trovato una chiave locale preesistente allora individua la chiave più vicina nella sua tabella di routing e inoltra la richiesta di inserimento al nodo corrispondente. Se questo messaggio di inserimento causa una collisione il nodo riceve il documento preesistente, si comporta come se fosse stata effettuata una richiesta (cioè memorizza localmente il file e crea una entry per la sorgente dei dati nella tabella di routing) e trasmette a ritroso i dati al suo predecessore nella catena degli inserimenti.

Se viene raggiunto il limite di `HTL` senza che si rilevi alcuna collisione di chiave, viene propagato un messaggio `Reply.Insert` all'indietro fino all'utente che ha avviato l'inserimento: questa condizione indica che tutto è pronto per effettuare l'inserimento vero e proprio.

A questo punto l'utente manda in un messaggio `Send.Insert` i dati da inserire che verranno propagati attraverso il percorso stabilito dalla richiesta iniziale di inserimento e saranno memorizzati in ogni nodo del cammino.

Ciascun nodo crea anche una entry nella sua tabella di routing in cui associa la sorgente dei dati, cioè chi inserisce, con la nuova chiave. Come nel caso delle richieste, ogni nodo della catena può decidere in modo unilaterale di cambiare il messaggio di inserimento per annunciare sé stesso o un altro nodo scelto arbitrariamente come sorgente dei dati.

Se un nodo non può inoltrare un inserimento al suo nodo preferito giù lungo la catena perché la destinazione è irraggiungibile o si verrebbe a creare un loop, il nodo che sta inserendo prova a inviare al nodo associato alla seconda chiave più vicina, quindi alla terza, e così via con le stesse modalità di backtracking viste nella sezione precedente per le richieste. Se la procedura di backtracking ritorna indietro fino ad arrivare all'utente che ha inserito per primo il file, questo indica che è stato possibile contattare meno nodi di quelli desiderati. Come per le richieste, i nodi possono ridurre valori eccessivi di `HTL` ed eventualmente ignorare inserimenti che rimangono in coda di attesa per un periodo di tempo eccessivo.

Analogamente al caso delle richieste, anche per quanto riguarda gli inserimenti un nodo non è pertanto in grado di vedere il cammino percorso da un inserimento al di là del nodo che gliel'ha direttamente inoltrato ed è molto difficile individuare la persona che ha inserito il dato in rete.

Gli utenti che pubblicano documenti in Freenet, oltre che rimanere anonimi grazie al meccanismo appena visto di inoltro e caching tra i nodi, possono usare le chiavi SSK per gestire i propri pseudonimi e realizzare un modello di fiducia anonima degli utenti che ricevono i dati nei confronti di chi li pubblica.

Chiave	Dato	Indirizzo
8e0109xb87wkhkujhs98k	99usbkjhgd7333khjgs763	tcp/5.34.27.4:6473
uushs89763kjhx7w732722	yy6254231gsyw4GGcwhgs	tcp/89.34.36.1:24855
kjhks872228x0982876jjhd	TTRos384hgygduybv1111n	tcp/194.44.62.66:9897
878772kx762776xbv8622		tcp/64.28.67.48:43653
222764kjhx8f63wkbkjs77w		tcp/4.18.49.35:65466
57765xkjhd72729jnbck01kj		tcp/55.18.4.1:3895

Tabella 5.1: Chiavi note ad un nodo

5.1.5 Resistenza alla cancellazione da parte di terzi

Il tentativo di cancellare un documento dal data store di Freenet può essere condotto in parallelo su due fronti:

- In rete attraverso l'impiego di nodi ostili controllati dall'attaccante che lavorano per "spingere" al di fuori del data store distribuito i dati da censurare.
- Localmente eliminando il file dai data store locali dei nodi su cui è memorizzato.

Per impedire la censura delle informazioni pubblicate, Freenet implementa una combinazione di soluzioni a diversi livelli del sistema:

- Politica di gestione dello spazio di storage.
- Dispersione e replicazione del documento.
- Negazione plausibile di responsabilità da parte dell'operatore del nodo.

5.1.5.1 Politica di gestione dello spazio di storage

Freenet non mira a offrire un servizio di data store permanente e quindi tutte le informazioni inserite devono confrontarsi con la capacità di storage limitata della rete. Gli operatori dei nodi decidono in fase di configurazione quanto spazio su disco offrire al data store distribuito.

I documenti memorizzati nello storage locale di un nodo sono gestiti secondo un algoritmo di caching *Least Recently Used* (LRU) in cui i documenti sono mantenuti in ordine decrescente in base al tempo dell'ultima richiesta ricevuta per essi (o in base al tempo di inserimento, se un file non è ancora mai stato richiesto). In questa struttura dati simile ad uno stack, illustrata in figura 5.1, ogni chiave binaria di cui il nodo è a conoscenza è associata in una entry al relativo documento nel data store locale e al corrispondente indirizzo sorgente.

Una copia dei documenti richiesti con maggiore frequenza è pertanto memorizzata nel data store locale al nodo e le relative entry sono spostate verso la cima dello stack mentre quelle dei file meno richiesti vengono scalate verso la base. Per incentivare gli utenti a comprimere i file prima di inserirli in Freenet, le entry dei documenti si muovono sullo stack di una distanza variabile legata alla dimensione del dato: i documenti più grandi sono penalizzati e percorrono sullo stack una distanza minore di quella dei documenti di piccole dimensioni. Al di sotto di una certa posizione nello stack il nodo non memorizza più in locale i dati accoppiati ad una chiave, ma solo l'indirizzo sorgente ad essa associato, di fatto elimininandoli dal nodo.

Il metodo per scartare i dati impiegato in Freenet ha quindi l'effetto di rimuovere i documenti meno richiesti (cioè per i quali si è registrata una bassa quantità di richieste da parte degli utenti),

non quelli più impopolari (cioè quelli sgraditi a molte persone). Se fossero i dati più impopolari ad essere rimossi, allora questo potrebbe essere usato per censurare i documenti. Il design di Freenet è molto attento affinché questo sia evitato.

Questo modello fa in modo che lo scarto di un documento dal data store di un nodo sia un'azione intrapresa individualmente dal nodo stesso in base alle richieste che ha veicolato; nel protocollo di rete Freenet non esiste alcun messaggio per cancellare un file dal data store: un documento non può essere volontariamente rimosso da Freenet neppure dall'utente che l'ha inserito e questo rende vano qualsiasi attacco censorio nei confronti di chi ha pubblicato documenti.

Se un autore vuole assicurarsi che un documento permanga in Freenet, tutto quello che deve fare è richiederlo o reinserirlo molto spesso.

5.1.5.2 Dispersione e replicazione del documento

Nonostante tutti questi accorgimenti, la capacità limitata di contenere informazioni rende Freenet vulnerabile ad un attacco in cui la rete venga inondata di documenti spazzatura allo scopo di cancellare quelli presenti. L'algoritmo di routing key-based impiegato in Freenet fornisce una certa resilienza a un attacco di questo tipo. Infatti i nuovi file vengono segregati in prossimità dei nodi che li inseriscono: in questo modo la parte di Freenet rimanente non viene interessata dall'inondazione e le informazioni in essa presenti non vengono cancellate.

Le chiavi che guidano il meccanismo di inoltro e caching dei documenti sono hash e quindi file simili nel contenuto o nella stringa di descrizione o appartenenti a uno stesso subspace saranno associati a chiavi binarie non correlate tra loro: le chiavi non trasportano informazioni sul contenuto semantico dei relativi documenti. Questo significa che documenti che trattano argomenti simili saranno sparsi su nodi diversi e lontani nella rete e per un censore non sarà sufficiente agire contro un singolo nodo per cancellare le informazioni a lui sgradite.

5.1.5.3 Plausible deniability per l'operatore del nodo

Si è visto quanto Freenet renda difficile individuare su quali nodi sia presente un documento, ma si immagini ora che un attaccante abbia comunque localizzato tali nodi. Per far fronte a questa evenienza, Freenet è progettata per ampliare al massimo la negazione plausibile di responsabilità da parte di chi la usa.

Come spiegato in sezione 5.1.2.4, tutti i file memorizzati su un certo nodo sono crittografati e quindi inaccessibili al suo operatore a meno che egli non conosca le chiavi di richiesta e di cifratura. Inoltre la presenza di un dato su uno specifico nodo può essere dovuta sia al suo inserimento o lettura, sia allo spontaneo ed automatico funzionamento della rete Freenet, riducendo le possibilità che un operatore risulti legalmente perseguibile per contenuti questionabili ospitati sul suo data store.

In conclusione Freenet impedisce totalmente attacchi di tipo censorio realizzati costringendo gli autori o gli operatori a cancellare documenti: nessuno, neanche chi l'ha inserita, può volontariamente cancellare una informazione da Freenet.

5.1.6 Ricerche sui dati

La messa a punto di un meccanismo che consenta agli utenti di cercare in rete documenti di loro interesse, ma di cui non conoscono le relative chiavi è un problema aperto di Freenet. Nel web esistono alcuni soluzioni efficaci, ad esempio l'impiego di siti *spider* che visitano i nodi e indicizzano i file oppure il mantenimento di siti che ospitano elenchi di segnalibri. Tuttavia questi approcci hanno la caratteristica di essere centralizzati e questo non soddisfa appieno i requisiti di Freenet. Inoltre

le chiavi binarie non forniscono informazioni sul contenuto del corrispondente file e quindi l'entità migliore candidata a catalogare un documento è soprattutto l'utente che lo ha inserito.

Un metodo semplice per realizzare un vero sistema di ricerca in Freenet potrebbe consistere nel creare uno speciale subspace pubblico per inserirvi chiavi che puntino indirettamente ad un documento. Quando gli autori pubblicano i file, potrebbero inserire in rete anche varie chiavi che per indirezione rimandino ai documenti e associno ad essi le relative parole chiave (*keyword*). Ad esempio questo testo potrebbe essere inserito e indirettamente indicizzato dalle chiavi `keyword:anonimato` e `keyword:sicurezza`.

Il sistema consentirebbe a numerose chiavi binarie diverse di coesistere per una stessa keyword (a differenza che per i file normali dove una chiave indica un solo documento) e richieste per questa chiave ritornerebbero corrispondenze multiple. Gestire un gran numero di file indiretti per le keyword più comuni sarebbe tuttavia difficile perché il meccanismo di routing le attrarrebbe tutte sugli stessi nodi.

5.1.7 Vulnerabilità ed attacchi

Per la sicurezza di Freenet l'obiettivo primario è proteggere l'anonimato degli utenti che richiedono e inseriscono i file. È anche importante proteggere l'identità degli operatori che ospitano i nodi dove risiedono i file. Anche se chiunque può in modo banale fare in modo che un nodo memorizzi un file richiedendolo attraverso di esso, "identificando" quindi il nodo come uno di quelli che ospitano il file, ciò che è importante è che rimangono pur sempre altri nodi, non identificati, che mantengono il file: in tal modo un avversario non può rimuovere un file attaccando tutti i nodi che lo ospitano se non riesce a conoscerli tutti.

I file devono essere protetti rispetto a modifiche malevole ed infine il sistema deve essere resistente ad attacchi di tipo Denial-of-Service.

Reiter e Rubin [36] presentano una utile tassonomia delle proprietà di comunicazione anonima su tre assi.

Il primo asse è il tipo di anonimato: *sender anonimity* (anonimato del mittente) o *receiver anonimity* (anonimato del ricevente), che significano rispettivamente che un avversario non può determinare chi ha originato il messaggio o a chi è stato inviato.

Il secondo asse è l'avversario in questione: un *local eavesdropper* (un ascoltatore locale, in grado di monitorare il traffico in modalità passiva), un nodo ostile o un insieme di nodi ostili che collaborano o un web server (non applicabile a Freenet).

Il terzo asse è il grado di anonimato, che cattura la forza di ogni tipo di anonimato su uno spettro che spazia da *absolute privacy* (privacy assoluta: per l'avversario non è possibile neanche avvertire la presenza della comunicazione) a *provably exposed* (comunicazione comprovatamente esposta: l'avversario può provare chi siano mittente e destinatario). Di particolare interesse su questo spettro sono i punti beyond suspicion (sotto sospetto) e probable innocence (probabile innocenza). L'anonimato del mittente è sotto sospetto se, nonostante l'attaccante possa rilevare l'evidenza di un messaggio inviato, il mittente non appare un candidato più probabile ad essere l'origine di quel messaggio che ogni altro potenziale mittente del sistema. Una garanzia di anonimato più debole è la probabile innocenza: il mittente è probabilmente innocente, dal punto di vista dell'attaccante, se ogni mittente appare con la stessa probabilità tanto essere l'originatore del messaggio quanto non esserlo. La figura 5.6 visualizza lo spettro che descrive il grado di anonimato.

Dal momento che la comunicazione in Freenet non è diretta verso specifici destinatari, l'anonimato del ricevente è visto in modo più preciso come anonimato dell'interrogazione (*query anonymity*), cioè il nascondere la chiave che sta venendo inserita o richiesta. Sfortunatamente, dal momento che il routing

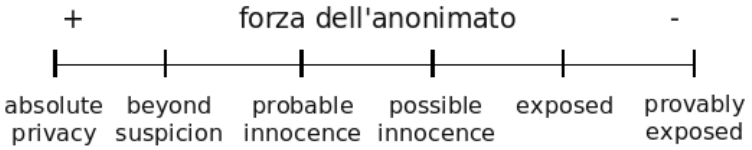

Figura 5.6: Grado di anonimato

dipende dalla conoscenza della chiave, l'anonimato della chiave non è possibile nello schema base di Freenet.

Contro un ascoltatore locale non c'è protezione per i messaggi tra l'utente e il primo nodo contattato perché la comunicazione avviene in chiaro per mezzo dell'insicuro *Freenet Client Protocol* (FCP). Siccome il primo nodo contattato può agire come un ascoltatore locale, il team di Freenet raccomanda che l'utente usi solo un nodo sul proprio calcolatore come primo punto di ingresso nella rete Freenet. I messaggi tra i nodi seguono il FNP e sono cifrati per contrastare l'intercettazione passiva.

Contro una collaborazione di nodi malevoli, l'anonimato del mittente è preservato sotto sospetto in quanto un nodo in una catena di una richiesta non può affermare se il suo predecessore abbia iniziato la richiesta o stia meramente inoltrandola. Tuttavia un'analisi del traffico, tramite correlazione delle statistiche dei messaggi che entrano ed escono dai nodi e dei valori dei campi `Depth` e `HTL`, può offrire indicazioni importanti per localizzare il nodo che ha generato un messaggio.

L'anonimato della chiave e un anonimato più forte per il mittente possono essere ottenuti aggiungendo all'algoritmo di instradamento una fase di pre-routing che realizza una variante della Mix-Net di Chaum [37]. Questa può essere effettuata utilizzando una rete Mixmion, descritta in appendice A.

In questo schema, i messaggi del modello base di Freenet sono cifrati con una successione di chiavi pubbliche che determinano la rotta che il messaggio cifrato percorrerà (sovrascrivendo così il normale meccanismo di routing). I nodi disclocati su questo cammino sono incapaci di determinare sia chi abbia originato il messaggio sia il suo contenuto (inclusa la chiave della richiesta), grazie alle proprietà anonimizzanti di una Mixnet. Quando il messaggio raggiunge il punto terminale della fase di pre-routing, sarà iniettato nella normale rete Freenet e si comporterà come se quest'ultimo nodo fosse il mittente del messaggio; questa possibilità verrà esplorata ulteriormente in sezione 5.1.7.4.

La protezione delle sorgenti dei dati è offerta dall'occasionale reset del campo `DataSource` nelle risposte. Il fatto che un nodo sia elencato come sorgente del documento per una particolare chiave non implica necessariamente che esso abbia effettivamente fornito quel dato e neppure che sia mai stato davvero contattato nel corso della richiesta. Non è possibile dire se un nodo precedente in una catena abbia fornito il file o abbia semplicemente inoltrato una risposta inviata da qualche altro nodo.

È certo invece che il solo atto di richiedere e ottenere con successo un file memorizza una copia del file sul nodo precedente se non ancora presente, quindi un'analisi successiva di quel nodo posto sotto indagine non rivela nulla dello stato precedente delle cose e puo' costituire una base legale per affermare che il dato non fosse già lì presente prima che l'atto investigativo ce l'abbia messo.

Richiedere un file specifico con un Hops-To-Live impostato a 1 non rivela direttamente se il nodo possedeva precedentemente il documento in questione oppure no, dal momento che i nodi continuano a inoltrare messaggi che presentano un HTL pari a 1 con probabilità finita. L'esito positivo di un gran numero di richieste di file simili, tuttavia, potrebbe gettare una base per sospettare che questi file fossero già memorizzati sul nodo.

In base a queste considerazioni, gli ideatori di Freenet inquadrano le sue proprietà di anonimato secondo lo schema illustrato in tabella 5.2.

Sistema	Attaccante	Sender anonymity	Key anonymity
Freenet Light	local eavesdropper	exposed	exposed
	collaborazione di nodi	beyond suspition	exposed
Freenet Light	local eavesdropper	exposed	beyond suspition
+ pre-routing	collaborazione di nodi	beyond suspition	exposed

Tabella 5.2: Proprietà dell'anonimato

```
Reply.Data  UniqueID=C24300FB7BEA06E3
            Depth=10
            HopsToLive=54
            Source=tcp/127.0.0.1:2386
            DataSource=tcp/192.235.53.175:5822
            Storable.InfoLength=0
            DataLength=131
            Data
            'Twas brillig, and the slithy toves
            Did gyre and gimble in the wabe:
            All mimsy were the borogoves
            And the mome raths outgrabe.'
```

Tabella 5.3: Esempio di dato

Nelle prossime sezioni si approfondiranno le osservazioni esposte in questa sezione considerando in dettaglio le diverse tipologie di attacchi [38] che possono essere condotti contro Freenet Light.

5.1.7.1 Eavesdropping

Per *eavesdropping* si intende l'azione di un ipotetico ascoltatore passivo, che verrà indicato col nome di Eve. Nello scenario considerato Eve è in grado di osservare le connessioni tra i nodi, ma non può cambiare, rimuovere o aggiungere messaggi extra.

Come si è visto nella precedente sezione, se Eve può monitorare i messaggi tra l'utente e il primo nodo contattato, allora è in grado di leggere il loro contenuto perché la comunicazione è in chiaro e priva quindi di qualunque protezione; è pertanto auspicabile che il primo nodo contattato dall'utente risieda sullo stesso calcolatore dove è in esecuzione il client.

I messaggi tra i nodi sono cifrati per contrastare l'intercettazione locale, essi compiono un normale scambio di chiavi seguendo il protocollo Diffie-Hellman per concordare la chiave con la quale cifrare le comunicazioni. Quindi i messaggi tra i nodi sono cifrati ed al riparo da Eve.

Tuttavia il protocollo Freenet è pubblico ed il codice sorgente è disponibile ed Eve potrebbe realizzare nodi modificati destinati a sovvertire il funzionamento della rete.

Si definisce *rouge* un nodo in grado di funzionare regolarmente, ma allo stesso tempo anche di registrare informazioni sugli inserimenti e sulle richieste per correlare questi dati tra loro al fine di rivelare l'identità di chi pubblica e chi scarica i documenti. Eve può quindi installare un nodo rouge per "ascoltare" qualsiasi messaggio venga inoltrato attraverso di lei.

I campi all'interno di tutti i messaggi sono *human-readable*, ad eccezione del campo `Data` (cioè il documento vero e proprio) che è cifrato. Ad esempio, in figura 5.3 è rappresentato un messaggio che contiene un documento di testo (per maggiore comprensibilità il corpo del documento è riportato in chiaro).

L'uso di hash come chiavi offre una misura di oscuramento contro ascoltatori casuali, ma nel caso si usino KSK è vulnerabile a un attacco *brute-force* basato su dizionari visto che le loro versioni in chiaro precedenti all'hashing devono essere largamente conosciute per essere di una qualche utilità. Inoltre la conoscenza della chiave KSK include una prova del fatto che chi ha richiesto il dato conosce la stringa di testo descrittiva.

Se gestisce un nodo e osserva i messaggi in transito su di esso, Eve può registrare informazioni sui nodi che richiedono chiavi, sui nodi che rispondono e con quali dati, sui nodi che inseriscono chiavi e sui nodi che replicano con messaggi `Reply.NotFound` o `Reply.Restart` (inviato alla scadenza del timer di attesa di risposta). Queste informazioni possono agevolare molto un tentativo di analisi del traffico, come illustrato in sezione 5.1.7.4.

5.1.7.2 Man in the Middle

Senza autenticazione tra i nodi, le connessioni cifrate tra di essi sono vulnerabili ad attacchi di tipo attivo; si ipotizzi uno scenario in cui l'attaccante, ora chiamato Mallory, sia in grado di interferire nel protocollo che il nodo Alice segue per concordare la chiave con un altro nodo Bob e si inserisca come *Man In The Middle* (MITM) tra di loro, convincendo Alice di essere Bob e Bob di essere Alice. Mallory può costringere Alice a possedere materiale compromettente richiedendolo attraverso di lei e facendo così in modo che il documento venga ospitato nella cache del suo nodo. Può anche far sembrare che Alice stia inserendo specifici documenti in Freenet.

Una minaccia particolarmente grave è un *all sides man in the middle attack*, cioè un attacco condotto su tutti i lati da Mallory che si inserisce tra tutte le connessioni di Alice con Freenet. Questo offre a Mallory il controllo totale sulle comunicazioni di Alice ed una maggiore capacità di intervenire sul suo data store. Mallory può verificare quali richieste lasciano il nodo di Alice e le risposte che vi entrano. Uno dei nodi di Mallory può richiedere un documento attraverso Alice, il quale può essere servito da un nodo upstream anch'esso sotto controllo di Mallory, ed in questo modo obbligare Alice a memorizzare una copia del documento sul proprio nodo. Dal momento che Mallory può far sembrare che il documento non esista da nessun'altra parte distruggendo le copie in suo possesso e facendo fallire ulteriori richieste del documento provenienti da altri nodi, può accusare Alice di essere l'autrice del file. Analogamente può inserire documenti in Freenet che sembrino essere originati da Alice. Può anche monitorare tutto il traffico di Alice, anche se non è in grado di determinare i contenuti che sta inserendo o richiedendo senza conoscere le chiavi binarie implicate nell'operazione.

Per difendersi da Mallory i nodi devono autenticare gli altri nodi. Questo si può ottenere ricevendo la chiave pubblica di un nodo attraverso un canale sicuro *out-of-band* (ad esempio una terza parte affidabile quale un fornitore di certificati digitali o per mezzo di un *web of trust* stile PGP [39]) e validando i loro fingerprint. Alcuni hanno suggerito di usare Freenet come key server, tuttavia questo approccio lascia aperto il problema di come validare le chiavi. Chiunque può inserire una chiave sostenendo che essa appartenga a un nodo arbitrario. L'implementazione di un web of trust funzionante per assicurare queste validazioni si è dimostrata molto difficile da realizzare.

Una soluzione intermedia che Freenet potrebbe adotttare per elevare la protezione nei confronti di Mallory è di permettere ai nodi di trasmettere la propria chiave pubblica (o il suo fingerprint) insieme al loro indirizzo. Quando un nodo inoltra l'indirizzo di un altro nodo (ad esempio come contenuto del campo DataSource in una risposta), può includere il corrispondente fingerprint della chiave pubblica (ad esempio: `tcp/123.123.123.123:19114/ <public key fingerprint>`).

Si potrebbe aggiungere un campo PK (Public Key) ai messaggi `Request.Handshake` per richiedere la chiave pubblica di Bob nel caso Alice non la conoscesse. Bob risponderebbe con la sua chiave pubblica nella `Reply.Handshake`. Alice può controllare che la chiave pubblica inviata da Bob combaci con il fingerprint. Alice può inoltre accertarsi della consistenza di questa informazione quando riceve un

riferimento a un nodo che già conosce assicurandosi che il campo PK sia lo stesso. Alice può anche richiedere ai suoi nodi adiacenti di provare la consistenza delle loro informazioni sulla mappatura tra IP e chiave pubblica di Bob. Se un certo numero di nodi adiacenti conferma la bontà di questa corrispondenza allora Alice può fidarsi di Bob. Se qualcuno di essi non è d'accordo sull'identità di Bob allora rifiuterà le sue connessioni.

In questo scenario, Mallory può ancora fare in modo che Alice riceva una chiave pubblica falsa per Bob. Tuttavia, se Mallory avesse voluto fornire una chiave falsa, poteva più facilmente anche solo annunciare un IP falso (per vari motivi fornire un IP falso rappresenta un attacco più semplice da portare a termine perché non si ha più a che fare con i consueti problemi in caso di MITM a livello IP: intercettare i pacchetti IP in tempo reale, modificarli, gestire la frammentazioni, l'ordinamento, la ritrasmissione e così via). Ancora una volta Mallory riesce così a fornire una chiave pubblica che possa essere completamente autenticata e l'attacco non è prevenuto.

Un vantaggio di questa soluzione tuttavia è che incrementa le possibilità per Alice di accorgersi della presenza di Mallory. Ora infatti, per evitare di essere scoperta da Alice, Mallory deve non solo essere in azione dall'inizio della negoziazione della chiave, ma deve anche individuare ed alterare ogni singolo riferimento che la riguarda ricevuto da Alice durante l'utilizzo di Freenet.

Se Alice inizia a usare Freenet prima dell'arrivo di Mallory, allora è a conoscenza di qualche indirizzo di nodo fidato e di alcune chiavi pubbliche valide. Mallory non potrebbe fare nulla rispetto alle connessioni che Alice stabilisce con questi nodi. Questi nodi le fornirebbero ulteriori chiavi pubbliche valide ogni volta che le comunicano un indirizzo di un nodo. Mallory non potrebbe sostituire chiavi false per i contatti precedenti di Alice (perché sono inviati su connessioni cifrate che lei non riesce a leggere).

A questo punto la sfida per Mallory diventa corrompere i nodi con cui Alice sta comunicando ed in qualche modo far sì che essi le inviino nuovi indirizzi di nodi con chiave falsa. Questo però può esporre Mallory al rischio di essere scoperta se non conosce i nodi già noti ad Alice. Infatti se Mallory prova ad inviarle una chiave falsa per un nodo di cui lei possiede già una chiave pubblica valida, allora il tentativo di attacco potrebbe essere rilevato.

Se Mallory non riesce a circondare completamente Alice, allora lei ha una possibilità di individuarla anche se parte con un solo indirizzo valido. Se Mallory invece circonda completamente Alice, l'unico modo in cui può essere scoperta è attraverso un controllo out-of-band del fingerprint. Gli utenti più paranoici potrebbero ancora fare affidamento a parti fidate per ottenere i loro indirizzi e chiavi attraverso canali sicuri.

5.1.7.3 Node discovery

Mallory potrebbe risparmiare molti sforzi se il suo indirizzo fosse largamente diffuso come un buon punto di partenza nella rete. Se ci riuscisse, i nodi che vengono avviati si ritroverebbero presto connessi con uno o più nodi controllati da Mallory. A quel punto Mallory sarebbe agevolata nel portare a termine l'attacco MITM descritto nella sezione precedente.

I nodi Freenet Light vengono a conoscenza degli altri nodi in quattro modi:

- In fase di installazione ogni nuovo nodo scarica da `http://downloads.freenetproject.org/seednodes/` il file `seednodes.ref`, una lista di indirizzi di nodi Freenet Light che è l'unione delle informazioni sui nodi presenti in rete fornite da un certo numero di nodi selezionati dal team di sviluppo e attivi da lungo tempo. Un nuovo nodo usa gli indirizzi dei nodi contenuti nel file `seednodes.ref` per inizializzare la propria tabella di routing, ereditando di fatto la conoscenza di nodi che possiedono maggiore esperienza. Questo meccanismo pone almeno tre problemi:

- La lista è pubblicamente accessibile e di fatto diffonde a chiunque informazioni sugli indirizzi di nodi Freenet senza che i loro operatori ne siano necessariamente a conoscenza.
- Nonostante l'URI del file `seednodes.ref` ridiriga il client a una serie di mirror per il download, Mallory potrebbe attuare uno sniffing delle richieste HTTP GET rivolte al server principale freenetproject.org o a questi siti mirror e stilare un elenco degli indirizzi IP dei nuovi nodi installati che hanno originato le richieste.
- L'integrità del file `seednodes.ref` scaricato può solamente essere verificata rispetto all'hash MD5 contenuto nel file `CHECKSUMS` ospitato presso freenetproject.org. Mallory, se fosse in grado di agire come MITM tra i nodi e questo server, potrebbe modificare la lista dei nodi a suo piacimento e fornire un MD5 valido per essa.

  Si deve inoltre tenere presente che la lista non contiene solo gli indirizzi dei nodi, ma anche informazioni statistiche su tempi di risposta alle richieste di determinate chiavi, il rapporto di richieste servite con successo, il tempo impiegato per stabilire una connessione e così via. Queste metriche sono utilizzate nella versione più recente dell'algoritmo di routing di Freenet Light (*Next Generation Routing* [40]) per migliorare le prestazioni della rete. Mallory quindi può modificare `seednodes.ref` non solo inserendo gli indirizzi dei nodi ostili che controlla, ma anche associare ad essi statistiche che facciano sì che i nuovi nodi li preferiscano rispetto agli altri, dirottando così una maggiore quantità di traffico verso di essi e aumentando l'efficacia dell'attacco.

- Quando un nuovo nodo si unisce alla rete annuncia la sua presenza ai nodi preesistenti. L'algoritmo impiegato tiene conto di due requisiti che in un certo qual modo sono in conflitto l'uno con l'altro. Da un lato è desiderabile che tutti i nodi siano consistenti nella scelta di quali chiavi binarie inviare a un nuovo nodo (ovvero si dimostrino concordi nella scelta della chiave da assegnare al nuovo nodo nelle proprie tabelle di instradamento), ma d'altra parte si creerebbe un problema di sicurezza se un nodo potesse scegliere la chiave da associare a sé stesso nelle tabelle degli altri nodi.

  Scartata quindi la possibilità di delegare autonomamente al nuovo nodo la decisione della sua chiave binaria iniziale, Freenet Light soddisfa entrambi i requisiti adottando un protocollo crittografico. Quando un nuovo nodo si collega alla rete sceglie un seed casuale e invia un messaggio di announcement ad alcuni tra i nodi preesistenti contenente il proprio indirizzo e l'hash del seed. I nodi raggiunti dall'announcement generano a loro volta un seed casuale, ne calcolano lo XOR con l'hash ricevuto nel messaggio e ricavano l'hash del risultato; quindi inoltrano come commitment il nuovo hash a qualche nodo scelto a caso dalle proprie tabelle di routing. Questo processo continua fino a che l'`HTL` dell'announcement scade. L'ultimo nodo a ricevere l'announcement genera solo un seed. A questo punto tutti i nodi della catena rivelano i propri seed e la chiave binaria associata al nuovo nodo è il risultato dello XOR di tutti i seed. Controllando i commitment i nodi possono fidarsi dei seed rivelati.

  Questo meccanismo porta a concordare una chiave casuale consistente per il nuovo nodo e tutti i nodi coinvolti aggiungono l'indirizzo di tale nodo nelle proprie tabelle di routing associandolo a quella chiave binaria.

- Quando un nodo riceve un messaggio `Send.Insert` o `Send.Data` che racchiude un file, esso aggiunge l'indirizzo di un altro nodo contenuto nel campo `DataSource` alla sua tabella di routing. Il nodo a questo punto può iniziare a stabilire nuove connessioni con questo nodo se necessario. Come descritto nella sezione 5.1.3. ogni nodo di una catena di inserimento o risposta può arbitrariamente decidere di indicare il proprio indirizzo o quello di qualunque altro nodo nel

campo `DataSource`; non c'è modo di verificare la validità dell'indirizzo annunciato e quindi esso potrebbe puntare ad un nodo ostile.

- Un operatore può inserire manualmente nel file `nodes.config` del proprio nodo gli indirizzi di altri nodi a cui è a conoscenza. In questo modo è possibile per un operatore fare in modo che il proprio nodo ignori gli indirizzi contenuti nei campi `DataSource` dei messaggi in transito e comunichi solo con nodi conosciuti e fidati contenuti in `nodes.config`. Tuttavia se tutti i nodi scegliessero di operare in questa configurazione, Freenet Light smetterebbe di funzionare. In quanto Opennet, in Freenet Light è fondamentale che tutti comunichino con tutti per beneficiare del meccanismo adattativo di *path compression* attraverso cui i nodi vengono a conoscenza gli uni degli altri attraverso il campo `DataSource` e stabiliscono così nuove connessioni nella rete.

In base alle caratteristiche di node discovery analizzate, capiamo come un attaccante di profilo sufficientemente elevato possa dirigere gran parte dei messaggi della rete attraverso i propri nodi ostili.

Ipotizziamo che Mallory controlli centinaia di nodi modificati di tipo *cancer*, cioè che impersonano nodi legittimi, ma modificano attivamente in modo malevolo le informazioni che veicolano. Nel caso peggiore Mallory può controllare l'inizializzazione delle tabelle di routing di gran parte dei nuovi nodi manipolando i download del file `seednodes.ref` e posizionare i propri nodi come preferiti per l'inoltro dei nuovi messaggi. Ogni nodo ostile poi può manipolare i campi `DataSource` di tutti i messaggi che inoltra assicurandosi che tutti i riferimenti ai nodi puntino ad indirizzi di nodi controllati da Mallory. In questo modo gli utenti che fanno transitare una richiesta attraverso uno dei nodi di Mallory col tempo si troveranno a conoscere e comunicare con un numero sempre più ampio di nodi malevoli.

Se tutti gli utenti di Freenet Light comunicassero solo con nodi gestiti da persone di cui si fidano e verso cui si autenticano su un canale sicuro, allora Mallory non costituirebbe più una minaccia. Tuttavia Freenet Light in quanto Opennet smetterebbe di funzionare correttamente in questo scenario.

Per trovare i nodi che non hanno volutamente annunciato la propria presenza, Mallory può ancora creare un nodo Freenet che esegua un *portscan* inviando un messaggio `Request.Hanshake`. I nodi infatti rispondono automaticamente a questo messaggio con un `Reply.Handshake` svelando la loro presenza.

Infine, dal momento che lo scambio iniziale di messaggi non avviene in formato binario, per un Service Provider è possibile scovare facilmente i propri utenti che eseguono nodi Freenet Light cercando nel proprio traffico i messaggi testuali del FNP (ad esempio: `Freenet v1.0 DH KeyExchange`) anziché dover attivamente enumerare le porte TCP.

5.1.7.4 Analisi del traffico

Un'attaccante, Mallory, può monitorare la rete installando nodi in luoghi strategici e analizzando il traffico cifrato tra i nodi. Osservando i campi `HTL` e `Depth` dei messaggi, la vicinanza della chiave binaria di ricerca e inviando richieste ad altri nodi, Mallory può ottenere informazioni rilevanti.

[37] descrive un attacco probabilistico che potrebbe compromettere l'anonimato del mittente, conducendo un'analisi statistica della probabilità che una richiesta in arrivo al nodo a sia inoltrata oltre o servita direttamente, e la probabilità che a scelga un particolare nodo b a cui inoltrarla. Questa analisi non è tuttavia immediatamente applicabile a Freenet perché il percorso delle richieste non è costruito con modalità probabilistica. L'inoltro dipende dal fatto che a possegga o meno il dato richiesto nel proprio data store locale, piuttosto che da una probabilità. Se un messaggio di richiesta è inoltrato, le tabelle di routing determinano dove vada inviato, e può succedere che a inoltri ogni richiesta a b, o non ne inoltri nessuna a b, o una via di mezzo tra queste alternative. D'altro canto, i valori dei

campi `Depth` e `HTL` possono offrire alcune indicazioni riguardo a quanti hop dista chi ha originato il messaggio, anche se questo è celato dall'inizializzazione con un valore casuale e di mezzi probabilistici per incrementarlo.

Come si è già accennato, i messaggi non scadono automaticamente quando il loro `HTL` arriva a 1, ma possono continuare ad essere inoltrati con probabilità finita, mantenendo il loro valore di `HTL` pari a 1. Il campo `Depth` è incrementato ad ogni hop e serve al nodo che risponde a impostare un `HTL` sufficientemente grande affinché la risposta giunga al mittente della richiesta. Gli utenti che richiedono il file possono inizializzare `Depth` a un piccolo valore casuale per occultare la propria posizione nella rete e un `Depth` uguale a 1 non è automaticamente incrementato, ma è inoltrato intatto con probabilità finita.

Anche se queste accortezze aggiungono casualità ai valori di `HTL` e `Depth`, questi campi forniscono comunque un aiuto nel restringere lo spazio di ricerca del mittente originale del messaggio. Alcuni hanno sostenuto che si potrebbe fare a meno del campo `HTL` perché i messaggi di risposta ereditano già il valore del campo `UniqueID` dalle corrispondenti richieste. I nodi quindi potrebbero evitare di dipendere dal campo `HTL` e fare invece affidamento sullo `UniqueID` per instradare le risposte indietro sulla corretta direzione seguita dalle richieste.

Eliminare sia il campo `Depth` che quello `HTL` non risolverebbe comunque completamente questo problema. Si è visto come il design del routing di Freenet fa sì che le richieste di una chiave vengano inoltrate a nodi associati a chiavi lessicograficamente sempre più vicine a quella cercata man mano che si procede di hop in hop sulla catena; inoltre le chiavi tendono a organizzarsi in cluster sui nodi, ovvero chiavi vicine sono servite da uno stesso nodo. Esaminando la distanza lessicografica tra la chiave richiesta e il cluster tipicamente servito dal nodo, l'attaccante può elaborare un'ipotesi sulla posizione che il nodo occupa nella catena di comunicazione. Nel caso più semplice, se un nodo riceve una richiesta per una chiave binaria che è molto distante da qualsiasi chiave abbia servito fino a quel momento, allora può supporre che la richiesta sia originata da un nodo direttamente connesso, o quanto meno da uno dei primi nodi nella catena.

Se Mallory invia una richiesta attraverso un nodo e riceve una risposta senza osservare alcun messaggio in uscita, allora capisce che il nodo o possedeva una copia locale del documento o ha inserito per primo il file nel data store. L'esito positivo di un grande numero di richieste per file simili e le tempistiche delle risposte potrebbero far sospettare che quei documenti risiedessero già sul nodo. Poiché i nodi intermedi non alterano i messaggi, Mallory può ottenere un grande aiuto nella sua opera di correlazione del traffico inviando messaggi strategici attraverso dei nodi e osservando la dimensione, l'ordinamento e la latenza dei messaggi che entrano ed escono da specifici nodi.

Sono pertanto molte le sfide che Freenet Light deve vincere per garantire un anonimato forte contro un attacco di analisi del traffico [41]. Un accorgimento ovvio è quello di consentire ai nodi di alterare l'ordine in cui i messaggi sono inviati e la loro latenza, estendere la lunghezza dei messaggi per renderli di dimensione costante o spezzettare i messaggi grandi e inviare un flusso costante di traffico tra nodi (traffico reale quando presente, più traffico fittizio durante i periodi di inattività ([42], [43]); l'appendice A contiene maggiori informazioni su questa tecnica).

Come anticipato in sezione 5.1.7, una soluzione contro questi attacchi di correlazione potrebbe essere l'aggiunta di un livello di *premix routing* (*Onion Routing* [44] [45], si faccia riferimento all'appendice B), un metodo che consente di stabilire una connessione anonima attraverso vari nodi.

Un limite nell'adottare questo sistema per Freenet è che richiederebbe al client di conoscere in anticipo gli indirizzi e le chiavi dei nodi e questo renderebbe ancora più complesso il problema della node-discovery all'avvio, perché il client dovrebbe conoscere non solo uno, ma molti nodi.

Un secondo aspetto negativo è il peggioramento dell'efficienza globale di Freenet causato dall'onion routing. Infatti i nodi in onion routing non conoscono il contenuto dei messaggi che inoltrano e quindi

si verrebbe ad aggiungere carico computazionale e di comunicazione al nodo senza poter attivare il sistema di memorizzazione dei documenti nel suo data store locale, punto cardine dell'efficienza di Freenet. Un compromesso è quindi permettere solo ad una parte dei nodi di funzionare come onion router. Ad esempio un nodo potrebbe scegliere un instradamento onion per i primi hop, e in seguito il messaggio verrebbe iniettato nella rete Freenet normale per essere instradato con l'algoritmo consueto.

Un'altra strategia per gli utenti più sensibili alla privacy è accedere a Freenet attraverso altre reti di comunicazione anonima esistenti [46] [47]. In questo scenario Freenet si concentrerebbe sulla memorizzazione del documento sui data store locali dei nodi e sulle caratteristiche adattative della rete, mentre altri gateway verso Freenet potrebbero offrire un servizio robusto per proteggere l'anonimato degli utenti.

5.1.7.5 Attacchi al routing

Un attaccante può adoperarsi per sovvertire l'efficienza dell'algoritmo di routing di Freenet Light in diversi modi:

- Nel caso di richieste che impiegano chiavi binarie KSK anziché CHK o SSK, Mallory risponde con dati falsi e inquina così le associazioni tra chiave binaria e indirizzo per i nodi nelle tabelle di routing del richiedente.

  *M*allory imposta il campo `DataSource` di ogni messaggio in transito sul proprio nodo in modo che punti al nodo ostile da lui controllato.

  In particolare, l'attaccante può attuare un *routing table takeover attack* [48]. In questo attacco il nodo ostile finge di essere molti nodi con indirizzi diversi, si connette ad un nodo vittima e ogni volta che soddisfa per esso una richiesta per un documento, scrive nel `DataSource` l'indirizzo di uno dei suoi nodi fittizi. Il nodo malevolo ottiene rapidamente un grande numero di connessioni, molte delle quali verso lo stesso nodo vittima. Il nodo obiettivo dell'attacco mantiene tante connessioni con diversi nodi virtuali che in realtà corrispondolo tutti allo *ubernode* di Mallory e questo rende vano anche l'eventuale impiego del premix routing ed espone maggiormente la vittima ad altri attacchi.

- Mallory specifica un `DataSource` diverso per ogni richiesta di documento servita, obbligando i nodi ad aggiornare continuamente le loro tabelle di routing e contrastando il fenomeno spontaneo di raggruppamento dei documenti con chiavi binarie vicine sugli stessi nodi (*clustering*).

- Mallory dirotta sul proprio nodo porzioni di traffico che originariamente verrebbero instradate verso chi ospita il file vittima. A questo scopo dapprima per mezzo di una ricerca esaustiva crea molti file che producono un hash lessicograficamente vicino a quello del file obiettivo e quindi li inserisce in Freenet specificando se stesso come `DataSource`. In questo modo aumenta la probabilità che le richieste e gli inserimenti per quel file vengano instradate verso il nodo di Mallory, di fatto sovvertendo il routing ed esponendo più informazioni sui nodi interessati al documento.

5.1.7.6 Attacchi DoS

Infine, si possono ipotizzare un certo numero di attacchi DoS.

La minaccia più significativa è che un attaccante provi a saturare tutta la capacità di storage della rete inserendo un grande numero di file spazzatura. Una possibilità interessante per contrastare questo attacco è utilizzare uno schema di micropagamenti che richiede a chi inserisce di eseguire un

calcolo piuttosto lungo come "pagamento" prima che l'inserimento sia accettato, in modo da rallentare l'attacco. Per ulteriori informazioni sui micropagamenti si faccia riferimento alla sezione 6.2.1.

Un'altra alternativa è quella di dividere il data store in due sezioni, una per i nuovi inserimenti e una per i file "stabili" (definiti come i file per i quali si è superata una soglia minima di richieste). I nuovi inserimenti possono solo prevaricare su altri nuovi inserimenti, lasciando intatti i file stabili. In questo modo un'inondazione di inserimenti malevoli potrebbe paralizzare temporaneamente le operazioni di inserimento, ma non farebbe sparire i file stabili. Per un attaccante è difficile legittimare artificiosamente i propri documenti spazzatura richiedendoli molte volte, perché le sue richieste sarebbero servite dal primo nodo che possiede il dato e non procederebbero oltre; non può inviare richieste direttamente ad altri nodi più avanti nella catena che possiedono i suoi file dal momento che le loro identità sono a lui ignote. Tuttavia l'adozione di questo schema potrebbe rendere difficile ai nuovi inserimenti legittimi sopravvivere abbastanza a lungo per essere richiesti da altri utenti e trasformarsi in stabili.

Gli attaccanti potrebbero tentare di scartare documenti esistenti inserendo versioni alternative sotto le stesse chiavi. Questo attacco non è facilmente realizzabile contro CHK o SSK, perché richiede di trovare una collisione del valore di hash o di forgiare una firma digitale. Un attacco contro KSK d'altra parte potrebbe risultare nel far coesistere entrambe le versioni nella rete. Ci si aspetta che il modo in cui i nodi reagiscono alle collisioni negli inserimenti renda questi attacchi più difficili da portare a termine con successo.

Il buon esito di un attacco di sostituzione (*replacement attack*) può essere misurato valutando il rapporto tra versioni corrotte e quelle integre del documento presenti nel sistema. Tuttavia, più copie corrotte l'attaccante tenta di mettere in circolazone (impostando un alto HTL durante l'inserimento), più cresce la probabilità che si verifichi una collisione di inserimento, che causerebbe un aumento del numero di copie integre del file. Per superare questo ostacolo l'attaccante potrebbe pur sempre iniettare i documenti falsi (con la stessa KSK del file originale) in molti nodi malevoli disconnessi dalla rete. Quando questi nodi si ricollegano a Freenet, può effettivamente accadere che ci siano più copie corrotte del file di quelle genuine e quindi l'attacco abbia successo.

Anche se non conosce la posizione di un nodo che ospita il file vittima, l'attaccante può tentare di isolare il nodo lanciando contro di esso un attacco *Distributed DoS* (DDoS).

Innanzitutto, per ricerca esaustiva, trova una chiave binaria che risulta essere più vicina a quella del documento vittima di tutte le altre presenti in Freenet. Poi avvia da tantissimi nodi diversi migliaia di richieste per quella chiave; per ridurre la complessità dell'attacco non è necessario che tutti i nodi malevoli dai quali iniettare le richieste stiano fisicamente su calcolatori diversi, ma si possono aprire connessioni da uno stesso host utilizzando un grande numero di porte diverse e impersonare così molti nodi virtuali. Le richieste corrispondono ad un documento inesistente che non verrà trovato su alcun nodo, quindi viaggiano fino ai nodi dove risiede il file obiettivo, saturandone le risorse e rendendoli irraggiungibili.

Questo attacco è meno efficace verso documenti popolari che sono presenti nel data store locale di molti nodi sparsi nella rete, ma funziona bene contro informazioni meno diffuse e che risiedono solo su un numero limitato di nodi specializzati.

Un attacco distribuito analogo al precedente, ma diretto contro un elenco di nodi anziché un documento, è naturalmente possibile a livello di protocollo di trasporto inondando di richieste e inserimenti direttamente le porte di rete degli host bersaglio (questa minaccia naturalmente non costituisce un problema specifico di Freenet, ma interessa piuttosto ogni Privacy Enhancement Technology (PET)).

Questi attacchi distribuiti richiedono una quantità di risorse tale da essere realizzabili solo da attaccanti di profilo alto e per periodi di tempo limitati.

Infine, un attaccante potrebbe ancora sopprimere tutti i messaggi che attraversano il proprio nodo,

generare falsi messaggi `Send.Data`, `Reply.NotFound` o `Reply.Restart` in risposta ai `Request.Data` o inviare messaggi contenenti quantità enormi di dati.

5.1.8 Conclusioni

Dall'analisi condotta si evince che Freenet Light non offre un "anonimato perfetto" (come quello della rete Mixminion, si veda l'appendice A) perché cerca di bilanciare la paranoia con l'efficienza e l'usabilità. Se qualcuno vuole scoprire esattamente cosa un utente sta facendo può riuscirci se dispone delle risorse necessarie. Freenet tuttavia persegue la finalità di fermare la sorveglianza di massa ed indiscriminata delle persone.

Un attaccante potente (ad esempio un governo) che sia in condizione di eseguire un attacco globale di analisi e correlazione del traffico della rete potrebbe effettivamente essere in grado di compromettere l'anonimato di chi pubblica e chi richiede documenti, ma per portare a termine un attacco di questo genere dovrebbe disporre di ingenti risorse di calcolo, connettività e storage per poter dislocare un gran numero di nodi di medie dimensioni in punti diversi della rete oppure pochi nodi, ma enormi. In fin dei conti, un attaccante che disponga di tali mezzi potrebbe trovare metodi migliori, più efficaci e meno dispendiosi, di spiare gli utenti.

Un attaccante che controlli solo un numero esiguo di nodi non molto grandi non potrebbe capire chi sta richiedendo i documenti e non sarebbe in grado di forgiare documenti falsi. Tali nodi non riuscirebbero a raccogliere informazioni sugli utenti ed a censurare documenti ed è questa categoria di attaccanti che Freenet cerca di rendere innocua.

I problemi di sicurezza più gravi che rimangono da risolvere in Freenet Light sono quelli relativi alla mancata autenticazione tra nodi e alla capacità dei nodi di apprendere l'esistenza di altri peer per mezzo del campo `DataSource`, caratteristica che permette di usare nodi *rogue* per fare *node harvesting*, cioè scoprire quali calcolatori fanno parte della rete.

L'impatto di quest'ultimo attacco si è rivelato nell'Agosto del 2005, quando il firewall nazionale cinese (anche noto come "Grande Firewall della Cina" o "Scudo Dorato") ha iniziato a bloccare il traffico proveniente da indirizzi di nodi Freenet Light individuati proprio tramite node harvesting.

5.2 Freenet Dark

La versione 0.7 di Freenet, rilasciata nell'Agosto 2006, costituisce una riscrittura completa di gran parte del codice della precedente versione stabile e rappresenta una svolta architetturale che vede il passaggio dal modello Opennet a quello Darknet. Freenet Dark non è compatibile con Freenet Light, ma le due reti separate possono coesistere sugli stessi nodi.

Nel documento scritto nel 2001 da Peter Biddle ed altri dipendenti Microsoft [49], una Darknet è descritta come "una collezione di reti e tecnologie usate per condividere contenuti digitali. La Darknet non è una rete fisicamente separata, ma piuttosto un livello protocollare e applicativo che si appoggia alle reti esistenti". Oltre alla condivisione delle informazioni sintetiche, che fa quindi ricadere nella categoria "Darknet" anche i sistemi peer-to-peer di storage distribuito, la caratteristica peculiare di una Darknet Peer-to-Peer è che gli utenti comunicano direttamente solo con altri utenti di cui si fidano (per questo le Darknet sono anche indicate come reti Friends-to-Friends). Nelle sezioni seguenti si analizzeranno le novità introdotte in Freenet Dark dal punto di vista del loro impatto sulla sicurezza del sistema.

5.2.1 Referenze

In Freenet Dark un nodo comunica solo con i pochi nodi di cui si fida. Rispetto alla versione 0.5 sono quindi stati eliminati i meccanismi di node discovery basati sulla lista preimpostata `seednodes.ref` e sull'inserimento nelle tabelle di routing degli indirizzi prelevati dal campo `DataSource` dei messaggi.

Per stabilire un rapporto di fiducia reciproca, ovvero una connessione tra due nodi, ognuno dei due operatori deve scambiare con l'altro una serie di informazioni sul nodo che amministra. Una referenza (*reference*) di un nodo appare nel modo seguente:

```
identity=zq1qf564lJKUIYdFAzG-l1234LFUYjM-s5d1vj8anEs
location=0.39841354643039973
testnet=false
myName=This is my node!
lastGoodVersion=Fred,0.7,1.0,1010
sig=594b174dc670e3555a12345693ed45806c9f2351ec289b271cecd6fd87456d96,303d[..]
version=Fred,0.7,1.0,1016
dsaPubKey.y=LlWomcYWo312345Hx76zr8xxwH5y2234pb6AxdEnIbUm7TUto5YUpVNWXcpC6[..]
physical.udp=72.194.102.78:31592
oldark.pubURI=SSK@N7SoSa8MApo0tj39k5S051Zjs5a3GiyuAL652B46-jQ[..],AQABAAE/ark
oldark.number=178
dsaGroup.g=UaRatn13245QvTlaaAXTMzn1Z15LDTXe-J~gOqXCv0zpz83CVngSkb--bVRuZ1[..]
dsaGroup.q=ALF123456R9Y1kQNVBc5kzmk0VvvCWosXY5t9E9S1tN5
dsaGroup.p=AIYIrE132hM38qPjirGGT-PJjWZBHY0q-JxSYyDFQfZQeOhrx4SUpdc~SppnWD[..]
ark.pubURI=SSK@47DisjrMpI0e3NDv6J8lBmmnbRblaq3m3qhsff31osw,EV[..],AQACAAE/ark
ark.number=11
End
```

La referenza di un nodo contiene tutti i dati necessari affinché i nodi fidati possano identificarlo e mantenere connettività con esso; in particolare, la referenza include il suo indirizzo IP, la porta UDP e una particolare chiave detta Address Resolution Key (ARK). L'ARK è una Updateable Subspace Key (USK), cioè una SSK dotata di un numero di versione che le consente di essere aggiornata. Quando un nodo cambia indirizzo di rete inietta in Freenet Dark il suo nuovo IP sotto l'ARK specificata nella sua referenza. In questo modo al nodo è sufficiente rimanere collegato anche ad un solo peer in seguito ad un cambio di indirizzo: gli altri nodi amici verranno a conoscenza del suo nuovo indirizzo ritirando la sua ARK (il meccanismo implementa una sorta di UDP Hole Punching [50] senza bisogno di fare affidamento su una terza parte e quindi evitando di esporre la topologia di rete a soggetti non fidati).

Per stabilire una connessione, ognuno dei due operatori dei nodi deve aggiungere la referenza dell'altro al proprio nodo. Quindi i due nodi scambiano i messaggi necessari a configurare la connessione; in essi non vi sono byte di sessione in posizioni prefissate perché il traffico è già cifrato usando chiavi simmetriche effimere usate solo in questa fase. Questo accorgimento aumenta l'invisibilità del taffico Freenet Dark agli occhi di un eavesdropper, ma rimane possibile individuare le connessioni in base a profili che tengano conto della dimensione dei pacchetti e del fatto che le connessioni sono UDP. Al momento infatti il traffico di Freenet Dark appare composto di tanti datagrammi UDP piccoli scambiati tra peer con elevata frequenza: i messaggi possono essere accorpati tra loro prima dell'invio, ma non viene aggiunto padding e quindi un attaccante di alto livello potrebbe portare a termine un'analisi del traffico.

Dopo aver stabilito la connessione, i due nodi reperiscono nelle referenze i parametri del protocollo Diffie-Hellman (DH) e concordano la chiave simmetrica crittografica. Il traffico tra i nodi amici è quindi cifrato e non è vulnerabile ad un attacco di eavesdropping, tuttavia un attacco MITM è ancora efficace; gli sviluppatori intendono risolvere presto questo problema impiegando un protocollo Station-to-Station, ovvero un DH autenticato da entrambe le parti che usa le chiavi pubbliche/private

recentemente incluse nelle referenze [51]. Un'alternativa resistente anche ad attacchi DoS è il protocollo *Just Fast Keying* (in particolare la variante JFKi) [52], ma questa resta una soluzione piuttosto complessa da implementare.

Lo scambio delle referenze in Freenet Dark è un aspetto cruciale dal punto di vista del funzionamento e delle garanzie di anonimato dell'intera rete. È importante che lo scambio avvenga solo tra utenti che nutrono reciproca fiducia e che le referenze siano comunicate su un canale out-of-band sicuro, per esempio un messaggio di posta elettronica cifrato e firmato con PGP.

L'adozione del modello Darknet è una difesa in più contro attacchi atti a ledere l'anonimato degli utenti, perché blocca la minaccia del node harvesting e rende inefficace l'uso di supernodi *cancer* per fare analisi del traffico. Tuttavia, ora viene delegata più responsabilità agli utenti. Un dissidente in Cina che scambi *reference* con un agente della Polizia Armata del Popolo e lo aggiunga nella sua rete di amici fidati metterebbe a repentaglio la privacy delle sue comunicazioni Freenet Dark e probabilmente esporrebbe a rischi anche quella dei suoi peer. Inoltre se l'agente riuscisse a capire con quali nodi il dissidente comunica allora potrebbe ricostruire la sua rete sociale e sospettare delle sue parti fidate. L'agente potrebbe ancora costruirsi un'opportuna identità falsa al di fuori di Freenet per attirare i dissidenti fino a godere della loro fiducia ed essere da loro inserito nel gruppo di peer amici così da portare a segno più efficacemente qualsiasi tipo di attacco alle loro comunicazioni.

5.2.2 Routing greedy distribuito in una rete small world

In una Darknet la rete sociale basata sulla fiducia reciproca tra utenti amici tende a configurarsi in gruppi di piccole dimensioni: una rete con questa caratteristica si può studiare con il modello *small world network*. Una small world network è una classe di grafi in cui i nodi di solito non sono vicini tra di loro, ma la maggior parte dei nodi può essere raggiunta da qualsiasi altro nodo compiendo un piccolo numero di salti da un nodo all'altro.

In una small world network in cui i nodi rappresentano le persone e gli archi le relazioni tra individui si osserva il cosidetto *small world phenomenon* descritto da Stanley Milgran nel 1967 [53]. Milgram provò come nella rete sociale della popolazione americana le persone fossero in grado di trovare con un certo grado di efficienza percorsi brevi (mediamente di 6 salti) che li mettessero in comunicazione con altri individui sconosciuti anche molto distanti. Studi più recenti hanno concluso che il fenomeno vale anche nel caso di Internet [54].

L'algoritmo di instradamento impiegato dalle persone in una rete small world è di tipo *greedy*, cioè consiste nel valutare ad ogni passo a quale individuo inviare il messaggio in base ad un criterio di "vicinanza" o similitudine tra soggetti: persone che condividono il luogo in cui lavorano, dove vivono, gli interessi che nutrono e via dicendo, hanno più probabilità di essere in contatto tra loro piuttosto che persone con caratteristiche dissimili.

Il problema di implementare questo algoritmo di routing sui nodi di una Darknet anonima è l'impossibilità per i nodi di conoscere le informazioni degli utenti che permetterebbero nella vita reale di scegliere il nodo preferito a cui instradare i messaggi. I nodi non conoscono né la propria posizione nella rete, né quanto distano i nodi di cui si fidano.

Oskar Sandberg del progetto Freenet è riuscito a mettere a punto un algoritmo [55] [56] che permette a Freenet Dark di superare il problema dei nodi ignari delle proprie coordinare e di calcolare in modalità collaborativa e distribuita le informazioni necessarie ad instradare il traffico. Il metodo si basa sul risultato di Kleinberg: la possiblità di ottenere un instradamento efficace in una rete small world dipende dal rapporto tra le connessioni di lunghezza diversa rispetto alla "posizione" dei nodi. Se si immaginano i nodi come punti disposti su una circonferenza, il numero di connessioni che legano due nodi sulla circonferenza deve essere inversamente proporzionale alla lunghezza della connessione; questa distribuzione suggerisce la presenza di molte connessioni brevi e poche connessioni lunghe

(*power law distribution*). Se questa condizione è verificata allora un algoritmo di routing greedy ha successo in $O(log^2 N)$ passi.

In Freenet Dark il metodo enunciato è implementato in due fasi:

- Quando i nodi si collegano alla rete scelgono una posizione casuale sull'anello.
- A intervalli casuali cominciano a scambiarsi di posto con altri nodi in modo da minimizzare le distanze tra nodi sull'anello (equivalenti alle lunghezze degli archi di un grafo). Due nodi concordano lo scambio di posto (*swap*) inviandosi query sul tunnel stabilito attraverso la catena di nodi fidati che li unisce.

L'attuale implementazione dell'algoritmo di scambio tra nodi non è sicura. Se la topologia della rete è esplicitamente resa nota è possibile forgiare opportune query e forzare uno swap ed al momento essa è accidentalmente esposta per permettere all'algoritmo di funzionare. Gli sviluppatori di Freenet pensano di risolvere in futuro questo problema esponendo volutamente la topologia, piuttosto che tentare di offuscarla ulteriormente, di forzare lo scambio tra nodi ed aggiungere una fase di *premix* routing.

Il routing *greedy* costituisce la base del routing in Freenet Dark e permette a due nodi qualunque di scambiare messaggi tra loro, nonostante ogni nodo della catena attraverso cui il traffico è instradato continui a comunicare solo con i suoi nodi fidati. Al di sopra di questo algoritmo di instradamento opera il consueto *key-based routing* già adottato in Freenet Light.

5.2.3 Data store

In Freenet 0.5 la probabilità che tutti i documenti richiesti o inseriti da un nodo vengano memorizzati nel suo data store locale è alta, soprattutto se dispone di spazio libero. Questo significa che se un attaccante riesce ad accedere al data store di un nodo può probabilmente identificare cosa è presente sul nodo, in particolare nel caso di grandi file distribuiti sotto forma di tanti file piccoli. In Freenet 0.7 è presente un'opzione per scegliere tra due comportamenti del nodo:

- Non memorizzare nulla: le richieste e gli inserimenti non sono salvati nel data store locale del nodo che li effettua. Il vantaggio è che l'analisi della memoria di massa del nodo non fornisce prove del materiale ospitato. Lo svantaggio è che un peer malevolo, in seguito ad un inserimento proveniente dal nodo vittima, potrebbe richiedergli il documento inserito per verificare se la richiesta provenisse inizialmente proprio da quel nodo.
- Memorizzare tutto: le richieste e gli inserimenti sono sempre memorizzati nel data store locale. Questa strategia mette al riparo dai controlli effettuati dai peer malevoli, ma espone ad un'analisi dei contenuti del data store nel caso l'attaccante riesca ad accedervi.

5.2.4 Conclusioni

Per le sue caratteristiche Freenet Dark è potenzialmente più sicura di Freenet Light. Al momento la vecchia rete è ancora più stabile e ricca di contenuti di quella nuova, ma gli sviluppatori del progetto Freenet desiderano incentivare il più possibile la migrazione degli utenti.

In questa fase di transizione può accadere che la maggior parte degli utenti non disponga di un numero sufficiente di amici fidati che mantengono nodi e a cui connettersi. Per ovviare a questo problema il team di sviluppo ha creato un canale IRC (*Internet Relay Chat*) dove gli operatori possono scambiare referenze tra di loro, anche in modo automatico attraverso $refbot$ (un IRC bot scritto in

Python). Questa soluzione tuttavia è deleteria per la sicurezza e il funzionamento della rete perché se la gente si connette a persone scelte casualmente o comunque non fidate di fatto si costruisce una Opennet su un sistema progettato per funzionare come Darknet.

Per risolvere questo serio problema gli sviluppatori intendono implementare nella versione 0.8 un'architettura mista Opennet/Darknet. Gli utenti che utilizzeranno Freenet in modalità Opennet non dovranno porsi il problema dello scambio delle referenze coi nodi fidati. Si cercherà di educare gli utenti per far capire loro quanto una Darknet sia più sicura di una Opennet ed è auspicabile che sempre più operatori inizino a stabilire relazioni di fiducia con altre persone attraverso il meccanismo delle referenze, che si cercherà di rendere più comodo da usare. Il routing sarà preferenziale: i nodi preferiranno instradare i messaggi attraverso i nodi della Darknet piuttosto che attraverso l'Opennet. L'obiettivo finale sarà far migrare gli utenti alla sola Darknet, che a quel punto supererà in dimensione la Opennet.

Infine, per rendere la presenza di connessioni Dark ancora più difficile da rilevare, la prossima versione di Freenet includerà semplici capacità steganografiche; ad esempio, tramite plugin sarà possibile selezionare il protocollo di trasporto desiderato (UDP o TCP) e adattare il profilo del traffico FNP in modo da simulare una comunicazione VoIP o HTTP.

5.3 Riepilogo

Freenet rappresenta lo stato dell'arte dei sistemi anonimi P2P per il data store distribuito ed è l'unica rete di questo tipo che gode di una grande base di nodi installati ed utenti. Sebbene Freenet Light si trovi in uno stato stabile e maturo ed ospiti un'alta quantità di contenuti, la nuova Freenet Dark introduce sostanziali miglioramenti per la protezione dell'anonimato degli utenti e degli operatori. Freenet Dark rende vani gli attacchi di *eavesdropping*, di *node harvesting* e l'analisi del traffico condotta tramite supernodi e rafforza le opportunità di esercitare la negazione plausibile di responsabilità.

La fase attuale di transizione da Freenet Light a Freenet Dark dimostra che le minacce ad un sistema per la protezione dell'anonimato non sussistono solo a livello tecnico, legale e politico, ma anche sociale. Condizione necessaria a garantire l'anonimato in Freenet è che l'anonymity set costituito dalla base di nodi installati sia grande: se la migrazione alle nuove versioni di Freenet non ha successo e non si attira un numero sufficiente di nuovi utenti la rete diventa insicura e l'anonimato è messo a repentaglio. Per far fronte a questo rischio è necessario da un lato che il team di sviluppo trovi un giusto compromesso tra funzionalità offerte e facilità di partecipazione alla rete, dall'altro che gli utenti sviluppino una maggiore sensibilità e motivazione verso le tecnologie di protezione della privacy e siano disposti a rinunciare ad un po' di facilità d'uso in cambio di libertà di comunicazione per i propri documenti e quelli degli altri. A questo proposito si ricorda il Progetto Winston Smith [57] che da anni in Italia promuove iniziative quali il convegno E-Privacy [58] destinate ad educare gli utenti all'uso di Freenet ed altre tecnologie di difesa della privacy.

Come riepilogo, nelle tabelle 5.4, 5.5, 5.6, 5.7, 5.8, 5.9 e 5.10 si sintetizzano i meccanismi impiegati in Freenet per raggiungere i suoi principali obiettivi di sicurezza.

Obiettivo	Anonimato della query
Implementazione in Freenet	Chiavi binarie per identificare i documenti: • KSK: stringa descrittiva → chiave pubblica/privata DSA generata dalla stringa descr. → SHA-1(chiave pubblica). • SSK: stringa descrittiva e chiave pubblica/privata del namespace → SHA-1[SHA-1(stringa descr.) XOR SHA-1(chiave pubblica)]. • CHK: SHA-1(documento).
Risultato	Operatori di nodi o attaccanti non possono risalire dalla chiave binaria richiesta o inserita alla descrizione del documento.
Attacchi possibili	Contro KSK: Ricerca in modalità brute-force della stringa per risalire alla stringa descrittiva.

Tabella 5.4: Anonimato della query.

Obiettivo	Integrità dei dati
Implementazione in Freenet	• KSK: firma DSA del documento con chiave privata. • SSK: firma DSA del documento con chiave privata. • CHK: verifica dell'hash SHA-1 del documento.
Risultato	• KSK: controllo minimo di integrità, forgiare documenti falsi richiede solo la conoscenza della stringa descrittiva. • SSK: controllo di integrità più affidabile, forgiare documenti richiede la conoscenza della chiave privata del namespace. • CHK: controllo affidabile di integrità.
Attacchi possibili	Contro KSK: • Ricerca in modalità brute-force della stringa per rispondere alle richieste con documenti falsi. • Key-squatting: l'attaccante inonda la rete di documenti indicizzati da stringe descrittive popolari per provocare collisioni.

Tabella 5.5: Integrità dei dati.

Obiettivo	Segretezza dei dati
Implementazione in Freenet	• KSK: cifratura simmetrica, la chiave è la stringa descrittiva. • SSK: cifratura simmetrica, la chiave è la stringa descrittiva. • CHK: cifratura simmetrica, la chiave è pseudo-casuale.
Risultato	Operatori di nodi o attaccanti non possono risalire dal documento memorizzato al contenuto del documento.
Attacchi possibili	• Ricerca brute-force della stringa descrittiva. • Crittoanalisi.

Tabella 5.6: Segretezza dei dati.

Obiettivo	Fiducia anonima
Implementazione in Freenet	Gestione di namespace tramite SSK.
Risultato	Fiducia nei confronti di chi pubblica, possibilità di organizzare gerarchicamente i documenti di un namespace.

Tabella 5.7: Fiducia anonima.

Obiettivo	Anonimato di chi accede o pubblica documenti
Implementazione in Freenet Light	• Catena di nodi che scambiano messaggi per servire la richiesta o l'inserimento di un documento. • Offuscamento della sorgente del documento attraverso reset arbitrario del campo `DataSource`. • Nomadismo e replicazione dei documenti tramite memorizzazione nei data store locali guidata da richieste e inserimenti. • Probabilità finita di inoltro di messaggi con `HTL` e `Depth` uguali ad 1, inizializzazione di `Depth` a un piccolo valore casuale.
Risultato	I messaggi scambiati e le tabelle di routing dei nodi non forniscono informazioni attendibili sul nodo che origina la richiesta o l'inserimento di un documento.
Attacchi possibili	• FCP insicuro: eavesdropping tra utente e primo nodo contattato. • Eavesdropping con nodi malevoli che operano nella Opennet. • MITM grazie all'assenza di autenticazione tra nodi. • Analisi del traffico tramite correlazione di informazioni ottenute da campi `HTL` e `Depth`, messaggi scambiati, vicinanza della chiave binaria al cluster tipicamente servito dal nodo, ecc. • Node harvesting via `seednodes.ref`, portscan, campo `DataSource`.
Contromisure di Freenet Dark	Darknet, mutua autenticazione tra nodi fidati basata su referenze: vanifica eavesdropping e node harvesting, impedisce di condurre analisi del traffico con supernodi.

Tabella 5.8: Anonimato di chi accede o pubblica documenti

Obiettivo	Negazione plausibile di responsabilità da parte dell'operatore del nodo
Implementazione in Freenet Light	• Opacità del data store: chiavi binarie e documenti cifrati. • Nomadismo dei documenti tramite memorizzazione nei data store locali guidata dalle richieste.
Risultato	• Un operatore di nodo non può conoscere e accedere ai contenuti del proprio data store locale. • Un documento può essere presente nel data store di un nodo anche se non è mai stato richiesto o inserito direttamente da un utente tramite quel nodo.
Attacchi possibili	• Richiesta diretta di un documento compromettente o riconoscibile attraverso il nodo vittima: provoca la memorizzazione del documento nel data store del nodo. • Facile ricostruzione dello storico dei documenti inseriti o richiesti attraverso il nodo vittima perchè sono memorizzati con alta probabilità nel data store locale.
Contromisure di Freenet Dark	Configurazione della strategia (tutto/niente) di memorizzazione dei documenti inseriti o richiesti localmente

Tabella 5.9: Negazione plausibile di responsabilità da parte dell'operatore del nodo.

Obiettivo	Resistenza alla cancellazione da parte di terzi
Implementazione in Freenet	• Assenza di funzionalità per la cancellazione volontaria di un documento dal data store. • Inserimenti che provocano collisioni contribuiscono a diffondere il documento preesistente. • Politica di gestione LRU del data store. • Diffusione key-based e replicazione del documento. • Offuscamento delle sorgenti del documento attraverso reset arbitrario del campo `DataSource`. • Negazione plausibile di responsabilità da parte dell'operatore del nodo.
Risultato	• I documenti più richiesti sopravvivono. • Resilienza del data store a inondazioni di documenti falsi. • Difficoltà di individuare quali nodi ospitano un documento: le tabelle di routing dei nodi non contengono associazioni attendibili e globali tra nodi e chiavi/documenti ospitati e i documenti formano cluster in base alle chiavi binarie anzichè al contenuto.
Attacchi possibili	• DoS. • Attacchi al routing volti a impedire il clustering dei documenti o a dirottare le richieste verso nodi malevoli che negheranno il servizio.

Tabella 5.10: Resistenza alla cancellazione da parte di terzi.

Capitolo 6

Analisi conclusiva

L'idea alla base dei sistemi P2P per il data store distribuito è piuttosto semplice: sostituire lo storage centralizzato di calcolatori locali con gruppi di dati memorizzati su un grande numero di nodi paritetici sparsi e che interagiscono tra loro attraverso Internet.

I sistemi studiati nei capitoli precedenti aggiungono a questo obiettivo di base una serie di proprietà generiche desiderabili, elencate di seguito:

- Disponibilità (*availability*):
  possibilità di accedere alle informazioni in qualunque momento e da qualsiasi punto ci si connetta alla rete.
- Persistenza (*durability*):
  le informazioni inserite nel sistema devono avere una durata virtualmente eterna.
- Controllo degli accessi:
  le informazioni sono protette sia a livello di segretezza (entità non autorizzate non possono leggere le informazioni) che di integrità (entità non autorizzate non possono modificare le informazioni).
- Autenticità (*authenticity*):
  un attaccante non può rispondere con un documento fasullo ala richiesta di un documento.
- Resilienza ai DoS:
  è difficile per un attaccante compromettere la disponibilità.
- Scalabilità di massa:
  il sistema funziona bene persino con miliardi di nodi.
- Prestazioni:
  in varie situazioni le prestazioni devono essere comparabili a quelle di uno storage locale.

In aggiunta, si possono perseguire nuovi obiettivi specializzati che sono raggiungibili grazie all'adozione del modello P2P:

- Anonimato (*anonymity*):
  è impossibile o molto difficile per un attaccante determinare chi ha pubblicato un documento e chi l'ha richiesto.

- Negazione plausibile di responsabilità (*plausible deniability*):
  l'operatore di un nodo è messo in condizione di poter negare di conoscere i documenti ospitati sul proprio calcolatore.

- Resistenza alla censura:
  Nessuno può censurare le informazioni, cancellandole o negandone l'accesso, una volta che esse siano inserite nel data store.

Sistemi quali OceanStore si prefiggono il primo gruppo di obiettivi generici sopraelencati e propongono anche un modello di business per il data store distribuito P2P.

Il servizio è erogato da una confederazione di società ed ogni utente paga una quota ad uno specifico provider anche se sfrutta le risorse di storage e banda di molti provider diversi. I provider comprano e vendono risorse tra di loro: quando una particolare regione del mondo ha storage e banda insufficienti, investitori sono incoraggiati ad aggiungere nuove risorse online. I provider, grazie alla capacità di questo sistema P2P di ripararsi autonomamente e riorganizzarsi in caso di fallimento, traggono vantaggio dalla gestione e manutenzione semplificata.

Idealmente, un utente affida tutti i propri dati all'infrastruttura di data store distribuito; in cambio, l'economia di scala del servizio offre livelli di disponibilità, performance e affidabilità che altrimenti non sarebbe possibile conseguire.

6.1 Routing

Nei sistemi P2P per il data store distribuito i nodi costruiscono una *overlay network* che poggia sulla rete IP sottostante.

Se il sistema P2P assegna un identificativo globalmente univoco (GUID, Global Unique IDentifier) ad ogni specifico documento o endpoint, allora localizzare un documento all'interno dell'*overlay network* e definire il percorso nell'*underlay network* necessario per raggiungerlo possono essere visti come problemi legati al routing: i nodi che richiedono un'operazione su un file del data store costruiscono messaggi che includono come indirizzo di destinazione un GUID anziché un indirizzo IP e li inviano attraverso la rete P2P che instraderà da un peer all'altro questi messaggi fino a che non raggiungono la destinazione.

Il GUID di un file in sola lettura è comunemente generato calcolando il message digest del file (GUID di documenti read-only in OceanStore, CHK in Freenet) o l'hash del nome del file concatenato alla chiave pubblica del suo proprietario (GUID delle *replica* in OceanStore, SSK in Freenet). In questo modo il GUID non contiene alcuna informazione sulla locazione dell'oggetto a cui si riferisce.

Le reti P2P *strutturate* (*structured network*) impiegano un protocollo globalmente consistente che assicura a qualsiasi nodo di poter instradare con efficienza un messaggio destinato ad un GUID verso un peer che ospiti il file desiderato; le decisioni che riguardano il routing sono prese in modo collaborativo e distribuito da una pluralità di nodi. Il servizio è quindi chiamato *Decentralized Object Location and Routing* (*DOLR*). Il sistema deve garantire che un file esistente nel data store possa essere sempre trovato e necessita perciò di un'organizzazione dei collegamenti tra i nodi dell'*overlay* piuttosto rigida e globalmente nota.

Il DOLR costituisce un nuovo livello di astrazione che abilita il sistema P2P a disaccoppiare la ricerca di un oggetto nella rete dalla ricerca a livello IP di un peer che ospita tale oggetto.

Al di sopra dell'interfaccia del DOLR, i sistemi P2P possono implementare in modo trasparente i meccanismi di replicazione, rimozione e nomadismo dei documenti per soddisfare i requisiti di disponibilità e gli altri obiettivi del livello applicazione.

Al di sotto del DOLR, il routing IP riesce a ottenere affidabilità, buone prestazioni e un uso ottimizzato delle risorse servendosi di percorsi multipli.

Quando si verifica un malfunzionamento su un collegamento tra peer, il fallimento di un nodo o un'inconsistenza su un percorso di rete, il DOLR è in grado di aggirare il guasto rinviando o replicando i messaggi.

Inoltre se il DOLR è in grado di preferire l'uso di risorse locali rispetto ad altre risorse globali (proprietà di *locality*), le prestazioni e la disponibilità della rete P2P migliorano in modo significativo poiché la probabilità che la comunicazione tra due nodi fallisca diminuisce quando la distanza tra i due peer si riduce.

Grazie ad una conoscenza globale distribuita dei documenti presenti nel data store e della posizione dei nodi che li ospitano è possibile cercare di minimizzare la distanza nell'*underlay network* che separa utenti e dati. Infine, la scelta del DOLR influisce anche sulla resilienza agli attacchi DoS perché può essere vista come la capacità del sistema di dissipare in modo efficiente il traffico dai nodi che lanciano l'attacco.

OceanStore e Mnemosyne utilizzano rispettivamente OpenDHT e Tapestry, due DOLR entrambi basati su tabelle di hash distribuite. Questa scelta permette a Mnemosyne di localizzare in modo affidabile i blocchi di dati sufficienti a ricostruire il file desiderato memorizzato nel data store steganografico distribuito. OceanStore impiega un DOLR che con le sue proprietà di località, scalabilità, localizzazione deterministica degli oggetti ed autenticazione certificata dei nodi consente di raggiungere elevate prestazioni ed affidabilità garantita.

Un approccio diverso al routing è quello adottato nelle reti P2P *non strutturate* (*unstructured network*), dove i collegamenti tra peer nell'*overlay network* possono essere stabiliti in modo arbitrario e non esistono vincoli rigidi sulla topologia.

In sistemi di questo tipo è semplice per i nuovi nodi entrare a far parte del sistema, anche se dispongono di una connessione *on-demand*: un nuovo peer quando si unisce alla rete esegue una fase di *bootstrap*, in cui eredita le informazioni topologiche da qualche altro nodo e col tempo crea nuovi collegamenti con altri peer.

Dal momento che a livello globale non esiste una correlazione tra un nodo e i contenuti ospitati su di esso, la ricerca di un documento nel data store si traduce nell'inoltro del messaggio di richiesta attraverso la rete di hop in hop fino a che non si trova un nodo che ospita una copia dei dati. In Freenet Light questa operazione avviene tramite un routing di tipo *key-based*.

Lo svantaggio principale del routing in reti *unstructured* è il mancato determinismo della ricerca: se il dato richiesto è largamente diffuso sui nodi P2P allora l'operazione ha successo, ma nel caso si cerchi un contenuto raro può accadere che esso non venga trovato. Un altro limite è la difficoltà di migliorare le prestazioni ottimizzando la rete in base a criteri quali la località delle risorse.

Da un altro punto di vista, la mancanza di un protocollo che fornisca ai nodi una conoscenza consistente e globale della topologia della rete P2P e la conseguente relativa autonomia del singolo peer nelle decisioni di routing, permette di raggiungere l'obiettivo dell'anonimato per gli utenti che operano sul data store distribuito, come è stato illustrato in dettaglio nel capitolo su Freenet. Al contempo però la significativa autonomia dei peer e la loro visione parziale della rete P2P significa esporli maggiormente ad attacchi di tipo *Sybil* od *Eclipse*, portati a termine ad esempio con nodi ostili *cancer* che operano per effettuare *takeover* delle tabelle di routing delle vittime.

Per proteggersi allora da queste minacce, sistemi quali Freenet Dark estremizzano il concetto di limitare la conoscenza della topologia di rete e giungono a costruire reti "Friends-to-Friends" dove i peer si collegano solo ad altri peer fidati e coi quali si è preventivamente eseguita una mutua autenticazione attraverso un canale sicuro out-of-band. Il prezzo da pagare in questo caso è una diminuzione della

disponibilità dei dati in caso di malfunzionamento dei nodi "amici" e una maggiore vulnerabilità ad attacchi all'anonimato condotti da nodi "amici" compromessi.

6.2 Affidabilità, fiducia ed accountability

Una differenza sostanziale ai fini della sicurezza tra un data store di rete centralizzato ed un servizio P2P per il data store distribuito risiede nella natura inaffidabile e non fidata dell'infrastruttura P2P.

Nel caso centralizzato (si pensi a NFS e le sue evoluzioni AFS, CODA, ecc.) i documenti sono memorizzati in remoto su server gestiti professionalmente in sale macchine chiuse. Amministratori e tecnici dei server possono intervenire prontamente per risolvere guasti o problemi di sicurezza; per far fronte a cali di prestazioni vengono aggiornate ed aumentate le risorse centralizzate. Come risultato, è ragionevole nutrire fiducia nel buon comportamento dei sistemi centralizzati.

Al contrario, i sistemi P2P possono essere gestiti da non professionisti e sono *inaffidabili* perché in qualsiasi istante può verificarsi un guasto o un malfunzionamento; dal momento che il tasso di fallimento cresce linearmente con la dimensione del sistema, in grandi reti P2P è praticamente certo che vi siano componenti mal funzionanti.

Inoltre i partecipanti alle reti P2P *non sono fidati* poiché potrebbero agire sfruttando vulnerabilità del sistema per sovvertire o interrompere il suo funzionamento.

Dall'analisi dei software condotta nei capitoli precedenti, si capisce come utenti ostili dispongano di numerosi vettori di attacco che agiscono a diversi livelli e sfruttano le vulnerabilità dei numerosi meccanismi implementati in un complesso sistema P2P per il data store distribuito.

Le minacce tuttavia non provengono solo da utenti malevoli. In assenza di controlli sull'*allocazione delle risorse* di storage, l'utente ha un incentivo a sfruttare tutto il valore del servizio allo scopo di massimizzare il proprio guadagno; se ad un utente è permesso comportarsi in questo modo, allora tutti gli utenti faranno lo stesso. Questo fenomeno è descritto con l'espressione "Tragedy of the Commons" [59]: una risorsa condivisa viene sovrasfruttata fino al suo degrado e conseguente distruzione finale, se tutti gli utenti perseguono per prima cosa il proprio interesse personale.

Il progetto di un sistema P2P per il data store distribuito deve pertanto soddisfare requisiti di allocazione e protezione delle risorse, dato che la banda e lo storage messi a disposizione dalla somma di tutti i nodi sono comunque grandezze finite.

Se ad un utente è permesso di inondare lo storage inserendovi un'enorme quantità di dati fino a saturare lo spazio di memorizzazione disponibile, gli altri utenti non possono più scrivere nello storage i propri file o, nel caso il servizio di data store non sia permanente, i vecchi documenti possono essere addirittura cancellati dal data store.

In alternativa, un utente malintenzionato potrebbe richiedere ai nodi della rete di eseguire un grande numero di operazioni di inserimento, modifica, estrazione, ricerca di file nel data store al fine di saturare le risorse dei peer. Ancora, potrebbe sfruttare il servizio di data store senza a sua volta mettere a disposizione degli altri utenti le proprie risorse.

Questi attacchi possono essere contrastati tenendo traccia delle operazioni sullo storage effettuate dagli utenti e rendendoli responsabili del loro comportamento, realizzando cioè la cosiddetta *accountability*. Lo scopo principale dell'*accountability* è di massimizzare l'utilità dell'operazione di un utente nei confronti dell'intero sistema minimizzando il suo potenziale rischio.

La soluzione a questo problema si articola tipicamente in due parti:

- Restrizione dell'accesso:
  per limitare il numero e la frequenza delle operazioni compiute da un utente sul data store distribuito è possibile servirsi di tecnologie dette *micropagamenti*. Prima di effettuare una operazione

sul data store l'utente deve "pagare" il servizio offrendo qualcosa di valore. Il rischio (in termini di banda o storage ad esempio) a cui il sistema va incontro erogando il servizio all'utente ha una dimensione pressapoco equivalente al beneficio conseguito dalla ricezione del micropagamento.

- Selezione degli utenti da favorire:
  mantenendo una *reputazione* per ogni utente si possono concedere meno risorse agli utenti a bassa reputazione o addirittura non fidarsi e negare loro di operare sul data store distribuito. Questo approccio accetta un rischio proporzionale alla reputazione che il sistema nutre nei confronti dell'utente.

6.2.1 Micropagamenti

Con i micropagamenti la decisione di garantire all'utente l'accesso al servizio è basata su poche informazioni collezionate nell'immediato della transazione e non richiede che i peer dispongano di informazioni precedenti riguardo l'un l'altro; questo modello quindi si adatta bene a reti P2P a partecipazione dinamica dove gli utenti possono unirsi o abbandonare la rete in qualsiasi istante e con frequenza arbitraria.

Inoltre i micropagamenti non richiedono che i peer si identifichino e questo è un punto di forza in una rete dove sia tecnicamente difficile identificare in modo permanente gli utenti, sia per motivi tecnologici come la transitorietà delle connessioni e della topologia della rete, sia sociali come la volontà degli utenti di nascondere informazioni personali, e ancora legali, come nel caso non vi siano contratti che vincolino gli utenti all'uso del servizio.

È naturale quindi considerare i micropagamenti una soluzione adatta per i sistemi P2P per il data store distribuito volti a garantire l'anonimato dei partecipanti alla rete, come Free Haven e Freenet.

I micropagamenti si dividono in due grandi schemi: *non fungibili* (*nonfungible*) e *fungibili* (*fungible*).

Si consideri un nodo, Alice, che effettua un micropagamento ad un altro nodo, Bob, per pubblicare un documento attraverso di lui. Se il micropagamento è di tipo non fungibile, Alice paga Bob con qualcosa che non costituisce per Bob un valore da reinvestire: lo scopo è semplicemente impedire attacchi DoS rallentando il ritmo con cui Alice può effettuare operazioni attraverso il data store e tipicamente il pagamento consiste per Alice nel provare a Bob che ha svolto un'operazione computazionalmente impegnativa (*POW, Proof Of Work*).

In Hashcash [60] [61] e Client Puzzles [62] tale operazione è un calcolo brute-force per ottenere l'input che provoca una collisione parziale rispetto ad un certo hash.

Un limite di queste soluzioni è la possibilità di parallelizzare il calcolo: nel caso in cui l'accesso alla risorsa tramite Bob sia pubblico, per evitare attacchi DDoS è allora necessario usare dei Time Lock Puzzles [63] che richiedono di attendere un certo lasso di tempo (*POT, Proof Of Time*), piuttosto che svolgere un calcolo.

Se invece il micropagamento è fungibile, Bob riceve un pagamento che racchiude un valore intrinseco e può essere a sua volta utilizzato per acquistare qualcos'altro: questo genere di pagamento è anche noto come *digital cash* e può avvenire sia in modo anonimo che palese. Un micropagamento fungibile per il data store distribuito è quello prospettato in Free Haven volto ad assicurare che gli utenti donino quantità di risorse proporzionali a quelle che consumano.

In base alle considerazioni fatte i micropagamenti si dimostrano una tecnologia adatta a prevenire attacchi di *flooding* transitori e che implicitamente consente di mitigare condizioni di congestione del sistema dovute a picchi di richieste di operazioni sul data store distribuito; sono invece meno efficaci

contro attacchi e situazioni di congestione cumulativi, ad esempio tentativi di saturare il data store che agiscono per un lungo lasso di tempo.

Come effetto intrinseco dell'uso di micropagamenti gli utenti che dispongono di maggiori risorse sono favoriti; l'entità del pagamento richiesto agli utenti va scelta con accortezza per evitare che pagamenti eccessivi escludano una parte significativa dell'utenza dal servizio nel caso non disponga di sufficienti risorse o la fruizione sia frustrante (inefficienza sociale).

Un utente può voler pubblicare nel data store in modo anonimo un documento con la garanzia da parte del nodo attraverso cui effettua l'inserimento che i dati rimarranno memorizzati per il tempo desiderato. A questo fine un semplice micropagamento generalmente espone chi pubblica al rischio di essere truffato dall'altro peer; d'altro canto un sistema che permetta a chi inserisce di rinfrescare periodicamente i propri dati perché permangano nel data store può essere problematico perché richiederebbe di connettersi ripetutamente alla rete incrinando potenzialmente l'anonimato dell'utente.

Finora si sono considerati micropagamenti che coinvolgono solo due nodi: Alice che richiede un'operazione alla rete e per questo effettua un micropagamento a Bob in cambio dell'accesso alle risorse sulle quali desidera operare.

Nei sistemi P2P per il data store distribuito esaminati nei capitoli precedenti un'operazione coinvolge tipicamente più dei due nodi che rivestono il ruolo di endpoint nella transazione, si pensi ad esempio al percorso compiuto dai messaggi in Freenet e Free Haven che attraversano una pluralità di nodi intermedi prima di giungere a destinazione.

Alcuni modelli possibili per usare i micropagamenti in un percorso di comunicazione multinodo P2P stabilito tra Alice e Bob sono i seguenti:

- Modello *end-to-end*.
  L'approccio più semplice è di imporre ad Alice un micropagamento a Bob, senza considerare i nodi intermedi.

  Questa scelta in un contesto P2P equivale ad un sistema privo di micropagamenti perché non risolve nessun problema. L'azione di Alice non è moderata dal micropagamento e lei è libera di attaccare qualsiasi nodo intermedio inoltrando attraverso di essi i suoi messaggi, sfruttando la loro banda o (ad esempio in Freenet) riempiendo i loro data store locali.

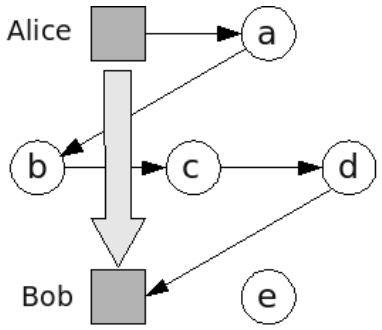

Figura 6.1: Modello di micropagamento end-to-end.

- Modello *pairwise*.
  Il percorso tra Alice e Bob può essere visto come una catena di coppie di nodi ed è possibile inserire un micropagamento per ognuna di esse.

Se si sceglie di usare micropagamenti di tipo fungibile, ogni nodo della catena guadagna dal suo predecessore un'unità da spendere con il suo successore. Tutti i segmenti della catena hanno costo eguale, Alice è l'unico debitore, Bob l'unico creditore e i nodi intermedi non sono né creditori né debitori.

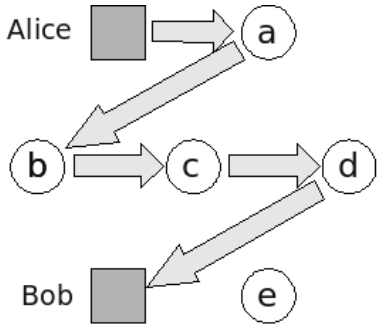

Figura 6.2: Modello di micropagamento pairwise.

Questo schema è tuttavia vulnerabile: un attaccante che impersoni sia Alice che Bob può sfruttare a suo piacimento le risorse dei nodi intermedi senza pagare nulla.

Se si impiegano micropagamenti non fungibile Alice rimane un debitore, ma anche tutti i nodi intermedi diventano debitori. I problemi in questo caso aumentano perché Alice, se dispone di sufficienti capacità di calcolo, può lanciare un attacco DoS contro un gruppo di nodi intermedi sfidandoli con moltissime POW, obbligandoli ad eseguire un calcolo come micropagamento per l'hop successivo e saturando così le loro risorse.

- Modello *amortized pairwise*.
  Per superare i rischi del modello *pairwise* si può pensare un sistema in cui il costo sostenuto da Alice viene ammortizzato durante il percorso, riducendo progressivamente ad ogni hop il costo del micropagamento richiesto tra le coppie di nodi intermedi.

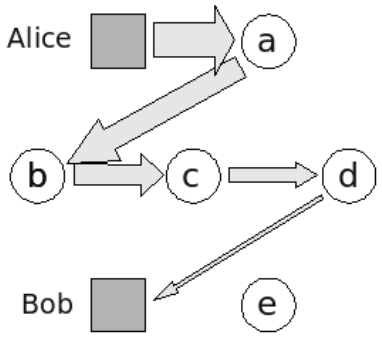

Figura 6.3: Modello di micropagamento amortized pairwise.

Ad esempio in figura 6.3 Alice paga 5 unità di micropagamento ad a, a paga 4 unità a b, b paga 3 unità a c, c paga 2 unità a d ed infine d paga 1 unità a Bob.

Se si usano micropagamenti non fungibili di tipo POW, lo schema è ancora vulnerabile perché Alice può sferrare un attacco come per il modello *pairwise*: inonda i nodi intermedi di richieste

di POW e, nonostante il carico diminuisca mano a mano che ci si avvicina alla destinazione, può comunque riuscire a saturare le risorse degli ultimi hop.

Se si scelgono micropagamenti fungibile lo schema sembra invece funzionare meglio e tutti i nodi intermedi sono creditori nei confronti della rete perché le loro risorse vengono pagate da una parte della somma complessiva pagata inizialmente da Alice.

Questo modello tuttavia pone ancora un problema: Alice deve conoscere il numero di hop che compongono il percorso verso Bob. Nei sistemi P2P considerati, si pensi a Freenet, questo non è in genere possibile e se Alice sbaglia la stima della lunghezza del percorso c'è il rischio che i micropagamenti richiesti ai nodi intermedi siano troppo pesanti e non soddisfacibili.

In Free Haven anche se Alice conosce i nodi della mixnet da attraversare, per implementare lo schema *amortized pairwise* di micropagamenti fungibili è necessario che ogni nodo intermedio conosca l'entità del pagamento ricevuto dal suo predecessore e la funzione di decremento delle unità di micropagamento: questo non è desiderabile perché i mix router potrebbero estrapolare da queste informazioni la quantità di nodi presenti nel percorso scelto e le loro posizioni relative nella catena.

- Modello *all points*.
  I problemi dei modelli precedenti possono essere risolti se Alice paga direttamente Bob e ogni nodo intermedio coinvolto nella transazione. Alice diventa l'unico debitore e tutto il costo dell'accesso alle risorse ricade su di lei.

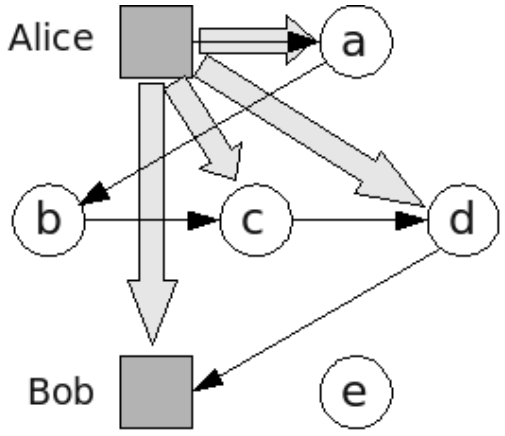

Figura 6.4: Modello di micropagamento all points.

Il limite di questa soluzione sta nel fatto che Alice deve conoscere tutti i nodi intermedi che compongono il percorso verso Bob e quindi si presta bene ad essere impiegata in sistemi P2P per il data store distribuito in cui i nodi sono completamente identificati, ad esempio Oceanstore, ma non in quelle reti anonime quali Freenet e Free Haven.

In Free Haven non si può adottare un approccio interattivo per questo modello: solo il primo nodo della mixnet conosce l'identità di Alice, mentre gli altri nodi intermedi non possono contattarla e sfidarla a eseguire un micropagamento. Per questo i micropagamenti in questo contesto devono essere di tipo non interattivo.

Inoltre, per evitare che Alice possa spendere più di una volta il micropagamento, è opportuno che nel pagamento sia codificata l'indicazione del destinatario (come nel caso di Hashcash); tale

informazione deve essere nascosta per evitare che un osservatore passivo possa ricostruire il percorso della transazione attraverso l'analisi dei micropagamenti. La cifratura del micropagamento di tipo *recipient-hiding* [64] promette di non divulgare informazioni riguardo il destinatario dei dati.

6.2.2 Reputazione

Un approccio diverso per realizzare l'*accountability* è la reputazione. Con reputazione si intende la memoria ed il sommario dei comportamenti di un utente nelle passate transazioni.

In un sistema di reputazione un peer esprime un giudizio (*rating*) sui nodi con cui effettua transazioni; perché il giudizio acquisisca un significato è necessario associare ad esso un punteggio (*score*). Compito del sistema di reputazione è aggregare questi giudizi per produrre uno o più punteggi e pubblicarli affinché i peer del sistema possano usufruirne.

Una differenza sostanziale tra gli schemi per l'*accountability* basati sulla reputazione e quelli fondati su micropagamenti è che nei primi i peer prendono le loro decisioni in parte facendo affidamento a informazioni fornite da terzi, mentre nei secondi non si richiede l'intervento di altri attori nella transazione tra i due peer.

Un buon sistema di punteggio alla base di un sistema di reputazione dovrebbe possedere molte delle seguenti caratteristiche:

- Accurato riguardo le prestazioni sul lungo periodo: il sistema distingue una nuova entità di cui non è nota la qualità del comportamento da un'entità che ha mostrato prestazioni scadenti in un lungo periodo di tempo. La scelta del sistema di punteggio riflette il livello di confidenza che si può assumere per una valutazione.

- Pesato a favore dei comportamenti recenti: il sistema riconosce e riflette le tendenze più recenti delle prestazioni dell'entità. Ad esempio un nodo che si è comportato bene per un lungo periodo di tempo, ma improvvisamente precipita verso scarse prestazioni, viene rapidamente individuato e catalogato come non fidato.

- Efficiente: è positivo che il sistema sia capace di ricalcolare velocemente un punteggio, in modo da produrre una valutazione incrementale.

- Resiliente agli attacchi: è bene che il sistema non permetta di influenzare il punteggio in nessun modo eccetto che comportandosi più onestamente o dimostrando una qualità di servizio superiore.

- Valutabile statisticamente: deve essere semplice trovare anomalie che possano provocare una differente valutazione del punteggio da parte del sistema.

- Privato: nessuno deve poter conoscere come un'entità ha giudicato un'altra eccetto il giudicante stesso.

- Lineare (*smooth*): l'aggiunta di un singolo giudizio o di un piccolo numero di giudizi non fanno variare di molto il punteggio.

- Comprensibile: deve essere facile spiegare agli utenti il significato del punteggio, non solo per far capire loro come funziona il sistema, ma soprattutto che effetto ha sul punteggio il loro comportamento.

- Verificabile: un punteggio contestato può essere accertato attraverso alcuni dati tangibili.

Alcuni di questi requisiti sembrano tuttavia contraddittori e probabilmente non sono conciliabili tutti in un unico sistema.

In sistemi anonimi quali Free Haven è desiderabile permettere all'utente di mantenere uno pseudonimo a lungo termine e ricompensarne il buon comportamento offrendogli prestazioni migliori e maggiori risorse. Il fatto che l'identità pseudonima non sia indissolubilmente associata alla persona fisica apre le porte a minacce quali il furto o la vendita dello pseudonimo: un utente malevolo può appropriarsi della reputazione collezionata da un utente legittimo.

Consentire l'uso di pseudonimi comporta anche il rischio dello *pseudospoofing*: una persona crea e controlla più identità false. Questo è particolarmente dannoso per i sistemi di reputazione. Se un utente scorretto acquisisce una cattiva reputazione, può servirsi dello pseudospoofing per rientrare nella rete P2P con una nuova identità, azzerando così la sua storia passata.

Ancora, una persona potrebbe forgiare un gran numero di identità e fare in modo che esse cooperino tra loro per incrementare reciprocamente il loro grado di reputazione. Questa tecnica è detta *shilling* e consiste nell'esprimere giudizi falsi o svianti per aumentare la reputazione dei nodi amici o diminuire quella dei nemici.

Una tecnica per limitare lo pseudospoofing e lo shilling consiste nel progettare il sistema di reputazione per tenere conto solo delle transazioni per le quali c'è una prova della loro esistenza; un esempio di sistema che adotta questo approccio è il "Customers who bought this book also bought..." di Amazon [33]. Se il giudizio è espresso in modo automatico ed incorporato nella transazione è molto più difficile che un attaccante faccia shilling, annunciando ad esempio una serie di giudizi positivi su una serie di nodi, perché ognuno di questi dovrebbe essere strettamente associato ad una specifica transazione.

Il problema fondamentale dello pseudospoofing non è che qualcuno possa operare sotto falso nome, ma proprio che una persona possa agire rivestendo i panni di molteplici utenti diversi.

L'impiego di una infrastruttura a chiave pubblica (PKI) consente di associare un'identità legale ad un utente della rete P2P e rappresenta un ostacolo per l'attaccante, ma non risolve completamente il problema. La conoscenza dell'identità legale dell'utente non garantisce che egli non sia libero di comportarsi in modo ostile perché potrebbe ad esempio trovarsi in una giurisdizione diversa da quella della vittima ed essere sì identificabile, ma non perseguibile; anche nel caso fossero nella stessa giurisdizione, il comportamento ostile potrebbe essere perfettamente legale.

Avversari di alto profilo potrebbero comunque disporre di molte identità certificate per sferrare attacchi e non bisogna tralasciare che i nodi onesti possono essere violati da un attaccante che sfrutta vulnerabilità dell'applicativo o del sistema operativo.

Per scoraggiare ulteriormente lo pseudospoofing si possono progettare sistemi P2P dove i nuovi nodi sono abilitati ad usufruire di poche o nessuna risorsa fino a che diano prova della loro onestà dando un contributo al sistema. Se è molto oneroso per un avversario dimostrare numerose volte la propria onestà allora lo pseudospoofing non diventa un vettore di attacco appetibile. Free Haven segue questo approccio e richiede ai nuovi nodi di donare risorse per entrar a far parte della rete.

Un'altra contromisura è il monitoraggio del comportamento degli utenti alla ricerca di prove di pseudospoofing. Gli account che sembrano controllati da un unico utente possono essere bloccati o segnalati agli altri peer.

Un sistema di reputazione potrebbe assegnare un punteggio ad un utente secondo una combinazione lineare di giudizi positivi e negativi, uno per ogni transazione. In questo caso la reputazione di un peer che abbia concluso molte transazioni e si sia comportato in modo disonesto un numero limitato di volte può risultare più alta di quella di un altro utente che si sia sempre dimostrato onesto, ma abbia

intrattenute solo poche transazioni. Il rischio in un sistema di questo genere è che un nodo si comporti bene in molte transazioni di piccola entità, guadagnando così una buona reputazione, ma poi agisca in modo disonesto in un'ultima transazione molto importante.

Una soluzione ovvia come associare ai giudizi un punteggio pesato in base all'importanza della transazione può essere d'aiuto, ma non elimina completamente il problema; l'importanza di una transazione potrebbe essere regolata dalla quantità di dati inseriti, considerando anche il tempo di permanenza nella rete richiesto. In questo scenario le transazioni più grandi influirebbero molto sul punteggio di reputazione di un peer e le transazioni piccole avrebbero solo un impatto minimo: l'utente non avrebbe un incentivo reale per completare onestamente le transazioni piccole, perché non comporterebbero per lui né un calo né un miglioramento significativo della sua reputazione.

Una soluzione più efficace potrebbe essere suddividere la reputazione in una serie di dimensioni diverse, ognuna delle quali rappresenti il comportamento di un peer in una specifica categoria di transazioni; un nodo ha quindi diversi indici di reputazione, uno per le richieste di inserimento, uno per l'estrazione di contenuti, uno per le operazioni di ricerca e cosi‘ via.

Un aspetto importante da considerare nel progetto di un sistema di reputazione è la sua fase di *bootstrap*. Se i nodi agiscono in base ai punteggi a loro disposizione, ma il sistema non ha ancora raccolto una quantità di dati sufficiente per esprimere punteggi utili, i nodi non sanno come comportarsi.

Il design di Free Haven prevede di gestire questo problema grazie alla generosità di alcuni partecipanti disposti a intrattenere relazioni con nuovi nodi al solo fine di verificarne la stabilità e robustezza: questi partecipanti rischiano parte delle proprie risorse e reputazione per valutare nodi sconosciuti, svolgendo così un servizio pubblico importante per l'economia della rete P2P.

Il *bootstrap* è un problema più sentito nei sistemi centralizzati rispetto a quelli decentralizzati. In un sistema decentralizzato, ogni nodi costruisce la propria visione della rete: si fida degli altri nodi in base alle transazioni che ha portato a termine e ai rapporti redatti da altri nodi fidati. In questo modo per ogni nuovo nodo l'istante iniziale di *bootstrap* è quello in cui si unisce alla rete; a partire da quel momento inizia a valutare la reputazione dei nuovi nodi.

In un sistema centralizzato invece il *bootstrap* è un momento unico per tutte le entità di rete e i giudizi sono accumulati tramite moltissime transazioni e per un lungo periodo di tempo. Visto che i nuovi utenti si affidano al sistema centralizzato per valutare la reputazione degli altri in base a transazioni avvenute in un arco temporale in cui essi non facevano ancora parte della rete, allora l'inquinamento della sorgente centrale di reputazione altera la fiducia di tutti i nuovi nodi.

Il progetto di un sistema di reputazione deve tener conto di quale punteggio assegnare ai nuovi nodi che si uniscono alla rete.

Se è possibile creare nuove identità tramite pseudospoofing e un nuovo utente inizia con un livello medio di reputazione, allora gli utenti che collezionano una cattiva reputazione sono incoraggiati ad abbandonare le loro vecchie identità per ricominciare con altre nuove. Una strada per gestire questo problema è di assicurarsi che tutti gli utenti inizino la propria vita nella rete con il punteggio di reputazione minimo possibile, cosicché anche utenti con cattiva reputazione siano motivati a mantenere le proprie identità attuali.

In una rete P2P il sistema di reputazione deve essere decentralizzato. Un primo approccio possibile è quello adottato da Free Haven, che consiste nell'eseguire su ogni nodo in modo indipendente il sistema di acquisizione delle informazioni di reputazione.

Nella rete vi è un gruppo di nodi indipendenti abilitati ad assegnare i punteggi (*scorer*). Quando si svolge una transazione, chi offre il servizio sceglie un sottoinsieme di *scorer*, generando ed inviando loro un gruppo di *ticket*. Ogni *ticket* rappresenta una ricevuta che consente a chi ha richiesto il servizio di esprimere attraverso uno specifico *scorer* un giudizio sul nodo che l'ha fornito. Le ricevute sono sigillate in modo che il fornitore non può associare un *ticket* ad un nodo che ha servito. L'utente che

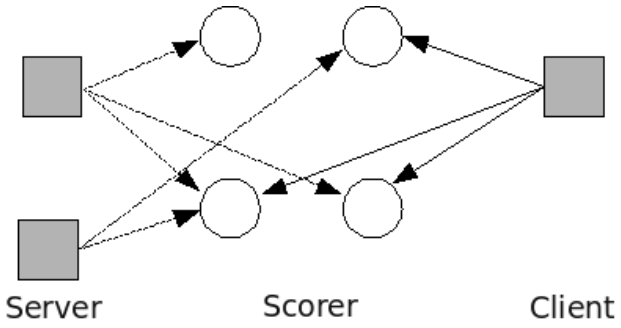

Figura 6.5: Sistema di reputazione a ticket.

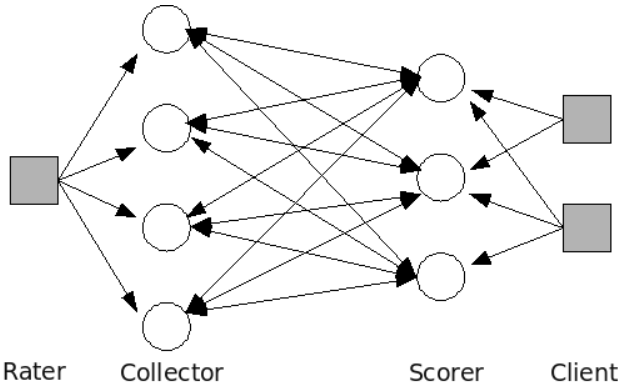

Figura 6.6: Sistema di reputazione decentralizzato.

richiede il servizio sceglie uno o più di questi *scorer* ed invia loro il suo giudizio. Questo meccanismo fa sì che i punteggi siano emessi attraverso un certo numero di nodi di cui sia il fornitore che il richiedente si fidano (figura 6.5).

Un'estensione di questo metodo si può realizzare allargando ad ogni peer della transazione la facoltà di emettere ticket per l'altro in modo da aumentare la simmetria del procedimento di valutazione.

Seguendo un diverso approccio, totalmente decentralizzato, si può realizzare un sistema di reputazione per mezzo di nodi che rivestono tre ruoli diversi: *rater*, *collector* e *scorer* (figura 6.6).

I *rater* sono i due nodi della transazione, esprimono un giudizio reciproco per mezzo di *ticket* e ricevute come spiegato precedentemente. A transazione conclusa, ogni *rater* divide il proprio giudizio in più parti, applicando ad esempio l'algoritmo di *secret sharing* di Shamir [65], e ne invia una parte ad ognuno dei *collector*: i *collector* possono estrarre il giudizio solo combinando le varie parti, ma nessuno di loro può singolarmente conoscerlo. Quando un nodo vuole conoscere la reputazione di un altro peer con cui desidera effettuare una transazione, interroga gli *scorer* ed essi hanno il compito di interrogare a loro volta i *collector*. In questo schema i *collector* realizzano una sorta di database cifrato e disperso [66] contenente i punteggi di reputazione dei nodi della rete.

L'implementazione di questo schema pone sfide tecniche non banali da risolvere sia dal punto di vista delle risorse di banda e calcolo richieste dal protocollo di comunicazione tra gli *scorer* ed i *collector*, sia per quanto riguarda il meccanismo di cui i *collector* devono disporre per autenticare i *rater* e validare i giudizi senza poterne leggere il contenuto.

6.3 Attacchi Sybil ed Eclipse

Gli attacchi *Sybil* ed *Eclipse* sono concettualmente simili allo pseudospoofing perché si appoggiano alla capacità dell'attaccante di impersonare e controllare identità multiple, ma agiscono a livello del protocollo di routing della rete P2P anziché a livello di applicazione. Questo significa che costituiscono una minaccia non solo per i sistemi P2P che permettono l'uso di pseudonimi, ma per qualsiasi *overlay network*. Mentre l'obiettivo dello pseudospoofing è generalmente massimizzare lo sfruttamento delle risorse del data store a vantaggio dell'attaccante, gli attacchi *Eclipse* e *Sybil* si prestano per realizzare azioni di censura o DoS a danno di specifici nodi vittima.

Nelle reti P2P ogni nodo comunica direttamente con un sottoinsieme di nodi *adiacenti* (*neighbor*), relativamente piccolo rispetto alle dimensioni della rete.

In un *Eclipse attack* l'attaccante controlla un certo numero di nodi malevoli e cerca ingannare uno o più nodi vittima nel tentativo di occupare coi suoi nodi ostili le entry della tabella di routing fino a sostituire completamente il gruppo di nodi adiacenti della vittima.

L'impatto di un *Eclipse attack* è notevole: l'attaccante può partizionare la rete P2P e separare i nodi vittima da tutti gli altri peer. L'attaccante è così in grado di controllare completamente e falsificare il risultato delle operazioni (ad esempio di richiesta, ricerca di file, ecc.) della vittima sul data store distribuito o di ingannare il sistema P2P compiendo operazioni a nome della vittima (ad esempio di inserimento, modifica, cancellazione di file, ecc.).

Se il sistema P2P per il data store distribuito offre proprietà di anonimato, allora con un *Eclipse attack* è possibile mettere in crisi l'anonimato delle transazioni, dal momento che l'attaccante è in grado di osservare tutte le comunicazioni in transito attraverso la vittima e questo facilita molto un'analisi del traffico.

Il successo di un attacco di tipo *Eclipse* può essere agevolato servendosi di un *Sybil attack*. In un *Sybil attack* l'attaccante gestisce un supernodo che finge di essere un gran numero di nodi distinti dell'overlay.

Un attacco *Eclipse* è possibile anche in presenza di una misura di protezione efficace contro i *Sybil attack*, ad esempio l'impiego di identità dei nodi certificate da una autorità fidata [67], in sistemi quali Freenet dove i nodi scoprono nuovi nodi consultando l'insieme di nodi adiacenti dei nodi che già conoscono. In questo scenario i nodi ostili possono diffondere informazioni sul proprio insieme di nodi adiacenti che includono solo altri nodi a loro volta malevoli: con questa tecnica anche un attaccante che dispone di un numero esiguo di nodi ostili può portare a compimento un *Eclipse attack*.

Le soluzioni proposte per mitigare la minaccia di un attacco *Eclipse* sono molte, si citano qui di seguito le più significative:

- Si possono usare delle *Constrained Routing Table* (CRT) [67], che impongono vincoli strutturali forti nella scelta dell'insieme dei nodi adiacenti. I nodi posseggono identificativi casuali e certificati e l'insieme di nodi adiacenti contiene solo i nodi dagli identificativi più vicini ad alcuni punti precisi all'interno dello spazio degli identificativi. La certificazione degli identificativi previene attacchi *Sybil* e le CRT ostacolano attacchi *Eclipse*.

  Il problema di questa soluzione è la scelta tassativa dei *neighbour* in base al loro identificativo che non permette di ottimizzare l'efficienza della rete P2P scegliendo come nodi adiacenti quelli che nella topologia dell'*underlay network* possono essere raggiunti con minor ritardo e spreco di banda, applicando ad esempio l'algoritmo di *Proximity Neighbour Selection* (PNS) [68] [69].

  Inoltre questa tecnica non è naturalmente appropriata per reti anonime poiché richiede di identificare univocamente e certificare tutti i peer.

- Un'alternativa è quella già accennata al punto precedente: imporre vincoli sulla prossimità dei *neighbour* [70]. Ogni nodo sceglie come peer adiacenti i nodi con ritardo di rete minimo. Un piccolo insieme di nodi ostili non può facilmente sferrare un attacco *Eclipse* perché non possono avere un ritardo contenuto rispetto a tutti i nodi legittimi.

  I limiti di questa soluzione consistono nell'assumere che l'attaccante non possa manipolare le misurazioni del ritardo e nell'essere efficace solo se le coppie di nodi sono sufficientemente distinte nello spazio dei valori di ritardo, ma simulazioni suggeriscono che questo non vale al crescere delle dimensioni della rete P2P.

- Una soluzione più efficace [71] [72] prende spunto da una semplice osservazione: durante un *Eclipse attack*, i nodi dell'attaccante presentano un numero di archi entranti nel grafo orientato della rete (*indegree*) più elevato rispetto alla media degli *indegree* degli altri nodi della rete P2P. Quindi i nodi legittimi possono limitare l'*indegree* dei nodi ostili scegliendo come loro *neighbour* quei nodi che nella rete P2P presentano un *indegree* inferiore ad un certo valore limite.

  Questa difesa ha l'inconveniente di introdurre un nuovo attacco: un attaccante può operare in modo da inserire gli indirizzi dei nodi vittima nelle tabelle di routing dei nodi ostili in modo da far crescere l'*indegree* delle vittime sopra la soglia e impedire ad altri nodi legittimi di usare le vittime come *neighbour*. Un effetto osservabile in questo caso è l'aumento dell'*outdegree* (numero di archi uscenti dal nodo nel grafo orientato della rete) dei nodi malevoli. Per questo motivo è necessario porre un limite sia sull'*indegree* che sull'*outdegree* dei nodi dell'attaccante e consentire ai nodi legittimi di inserire come *neighbour* i soli nodi che hanno un *indegree* ed un *outdegree* inferiori ad una certa soglia.

  La tecnica impiegata per forzare i nodi a controllare i *degree* degli altri peer assume una rete P2P dove ogni nodo possiede un certificato che associa l'identità del nodo ad una chiave pubblica; inoltre l'*overlay network* deve supportare una primitiva di routing sicura per mezzo di CRT. L'imposizione dei limiti sui *degree* dei nodi è realizzata per mezzo di una verifica distribuita e anonima. Ogni nodo del sistema periodicamente verifica che i nodi adiacenti abbiano un *degree* entro i limiti.

  > A questo scopo ogni nodo x della rete P2P è tenuto a mantenere una lista di tutti i nodi che hanno x nel loro gruppo di *neighbour*, cioè del *backpointer set* di x. Periodicamente x richiede in modo anonimo ad ogni suo *neighbour* di inviargli il suo *backpointer set*. Se il numero di elementi che fanno parte di quel *backpointer* è maggiore del limite di *indegree*, allora il nodo sotto verifica ha fallito il test e x rimuove quel nodo dal suo gruppo di nodi adiacenti.
  >
  > Per assicurarsi che il limite sull'*outdegree* sia rispettato, è verificato in modo analogo a quanto appena descritto che i membri del *backpointer* di un nodo mantengano un insieme di *neighbour* di dimensione appropriata; se il nodo che esegue la verifica scopre che uno dei suoi *neighbour* supera la soglia del *degree*, allora rilascia immediatamente la connessione con quel nodo.
  >
  > Questo meccanismo fa sì che il *degree* dei nodi non conformi al limite rientri in modo naturale al di sotto della soglia consentita. Per assicurarsi che le risposte alle verifiche siano ogni volta nuove ed autentiche, il nodo x include nell'interrogazione un *nonce* casuale e il nodo interrogato replica includendo il *nonce* e firmando la risposta. Perché il sistema sia sicuro è inoltre necessario che l'identità del nodo che effettua la verifica rimanga nascosta dal nodo verificato; se così non fosse un nodo verificato malevolo

potrebbe produrre una risposta fasulla in cui dichiara un *neighbour set* dalla dimensione consentita e che include il nodo che effettua la verifica. Questa caratteristica è implementata tramite un canale di comunicazione che offra la *sender anonimity*: i messaggi di verifica sono inoltrati attraverso un nodo intermedio (*anonymizer node*) scelto dal nodo che avvia la verifica. Per ogni nodo da verificare vengono effettuate varie interrogazioni attraverso differenti *anonymizer* ad intervalli di tempo successivi casuali, in modo da evitare sia attacchi di correlazione tra sfide e risposte, sia la possibilità che l'*anonymizer* scelto di volta in volta sia malevolo.
Secondo le simulazioni effettuate, questa difesa dagli *Eclipse attack* è più efficace delle precedenti basate su vincoli strutturali o su accurate misure di ritardo ed è compatibile con ottimizzazioni quali la PNS.

6.4 Replicazione dei dati

Un'ulteriore strada per gestire il problema dell'*accountability* è ignorare la questione e progettare il sistema affinché sopravviva al malfunzionamento o al comportamento ostile di una parte dei nodi.

Verificare che un nodo stia eseguendo correttamente i suoi compiti in accordo al protocollo comune è un'operazione molto complessa; allora sistemi quali Freenet scelgono di replicare i file e le funzionalità attraverso la rete (*mirroring*), cercando di assicurare che con alta probabilità il sistema funzioni bene, nonostante vi possano essere entità malfunzionanti. Inoltre architetture come quella di Freenet arginano l'effetto di inondazioni di dati volte a saturare il data store memorizzando i dati nei pressi di chi li richiede (*active caching*) in modo da evitare che richieste successive attraversino lunghi tratti di rete; la politica di scarto dei documento dalle cache deve proteggere i file più richiesti dagli utenti, penalizzando i dati fasulli inseriti dall'attaccante.

La ridondanza introdotta da tecniche quali *mirroring* e *active caching* si dimostra di grande aiuto in reti P2P inaffidabili e non fidate garantendo in caso di malfunzionamento di una risorsa la disponibilità di una sua copia. Una replicazione eccessiva tuttavia può significare una spesa aggiuntiva in termini di banda e data store troppo elevata per la rete P2P.

Per questo motivo sistemi quali OceanStore e Free Haven fanno uso di una forma di ridondanza particolarmente efficiente detta *codifica a cancellazione* (*erasure coding*). Ogni *chunk* di un dato viene trasformato in molti frammenti, in modo tale che per ricostruire il dato è necessario possedere solo una frazione del totale dei frammenti.

6.5 Autenticità, integrità e segretezza dei dati

Nella sezione 6.2.2 si è studiato il funzionamento dei sistemi di reputazione e si è visto come un meccanismo centrale sia l'espressione di un giudizio riguardo l'operato dei nodi coinvolti in transazioni. Le tecniche di verifica del buon funzionamento dei nodi in un contesto P2P per il data store distribuito si affidano largamente alla crittografia.

L'autenticità e l'integrità dei file forniti dai nodi sono accertate attraverso una combinazione di firme digitali e funzioni di hash sicure non invertibili. Una funzione di hash produce un sommario univoco di lunghezza fissa a partire da un file di dimensione arbitraria (*message digest*). Poiché il sommario di un file calcolato con una funzione di hash è univoco, è possibile usarlo per assegnare ai documenti un nome che li identifichi e non sia artificiosamente costruibile. In un sistema P2P per il data store distribuito che utilizzi il *message digest* per identificare e localizzare un file nella rete, i nodi possono verificare l'integrità dei dati restituiti generando il valore dell'hash e confrontandolo con l'identificativo richiesto (*redundancy check*).

I nodi possono costruire degli oggetti ricorsivi e in grado di verificare da soli la propria consistenza per mezzo di strutture ad albero in cui i blocchi interni contengono identificativi che puntano ad altri blocchi. Questi oggetti prendono il nome dall'identificativo dei loro blocchi di livello superiore e la loro integrità è assicurata dato che una manipolazione all'interno dell'albero modificherebbe l'identificativo del blocco di livello superiore. È questo il caso di OceanStore (sezione 2.4.1), in cui le strutture ad albero sono facilmente gestite tramite l'utilizzo di puntatori a blocchi, costituiti dal loro identificativo. Il blocco padre può contenere quindi sia dati propri che identificativi di blocchi figli; l'identificativo del blocco padre consente di verificare sia la propria intergrità che quella dei blocchi figli.

OceanStore, Freenet e Free Haven utilizzano SHA-1 come funzione di hash; dal momento che la sicurezza di un algoritmo di hash potrebbe essere messa a repentaglio da evoluzioni future nel campo della crittoanalisi, è bene che i sistemi P2P per il data store distribuito dispongano di una implementazione modulare che preveda la possibilità di aggiornare la funzione di hash impiegata.

Le firme digitali utilizzano la crittografia a chiave pubblica per provare che un utente è l'autore di un certo documento. Quando inserisce un documento nel data store, un utente firma l'identificativo del file con la sua chiave privata. Questa firma è un insieme di bit di dimensione fissa che può essere verificato da chiunque possegga la chiave pubblica.

La cifratura simmetrica entra invece in gioco per tutelare la segretezza dei documenti: i file sono vengono crittografati prima di essere inseriti e memorizzati nel data store.

Queste tecniche consentono ai nodi che hanno richiesto dei file in sola lettura di scartare i documenti manipolati o corrotti e replicare o diffondere quelli integri ed autentici; i nodi sono così in grado di esprimere localmente e in modo passivo un giudizio sul comportamento dei peer da cui hanno ricevuto quelle informazioni.

Esistono però alcune operazioni quali la modifica, la sostituzione o la cancellazione di documenti che hanno invece conseguenze globali sulla rete P2P e pertanto devono essere valutate attivamente da un insieme di nodi. Tipicamente queste operazioni richiedono di verificare le credenziali del nodo che le richiede rispetto ad una lista di controllo degli accessi e la decisione non può essere affidata ad un singolo nodo che potrebbe rivelarsi malevolo.

Sistemi quali OceanStore in questi casi usano il *Byzantine Agreement Protocol* [73] per consentire ad un insieme di nodi di giungere a concordare una decisione unica, anche se alcuni di essi (meno di un terzo del totale) si comportano in modo malevolo tentando di compromettere la procedura.

Una volta raggiunto l'accordo, i nodi possono anche firmare collettivamente la decisione presa per mezzo di uno schema di firma a soglia (*threshold signature*) [74] per consentire ad altri peer di verificarla in un istante futuro.

Meccanismi quali il *Byzantine Agreement* consentono ai sistemi P2P per il data store distribuito di fare un balzo da sistemi di memorizzazione a sola lettura delle informazioni a sistemi capaci di gestire in modo consistente la serializzazione delle modifiche e scritture dei documenti, grazie alla facoltà di individuare qual è la copia più recente di un file e di riconoscere se un file è obsoleto.

6.6 Tolleranza ai malfunzionamenti

Grazie alle tecniche che garantiscono la disponibilità dei documenti (*mirroring*, *active caching*, ecc.) e la loro autenticità ed integrità (hash, firma digitale, *Byzantine Agreement*, ecc.), un sistema P2P per il data store distribuito può comportarsi molto bene in caso di malfunzionamenti di nodi che accadano in modo indipendente, offrendo un servizio che perde un numero estremamente esiguo di blocchi di dato per anno [75].

Nel caso di reti P2P di larga scala non è tuttavia possibile assumere che i nodi falliscano sempre in modo indipendente. In caso di fallimenti correlati la tolleranza ai guasti del sistema rischia di venir

meno. Si pensi al fallimento simultaneo dei nodi OceanStore che ospitano le *replica* di un file: in questo caso lo schema di distribuzione delle *replica* potrebbe non garantire la sopravvivenza del documento. In modo analogo anche gli algoritmi di *Byzantine Agreement* smettono di funzionare correttamente quando si verifica una rottura simultanea di molti nodi.

Purtroppo in sistemi P2P reali la correlazione tra nodi è una proprietà piuttosto comune: si consideri il caso di nodi che condividono lo stesso segmento di rete, operatore, versione dell'applicazione P2P, sistema operativo, architettura hardware o locazione geografica. Un malfunzionamento od una vulnerabilità legati ad uno di questi elementi condivisi possono provocare un fallimento multiplo all'interno della rete P2P. Sistemi quali Freenet arginano il problema dei fallimenti correlati introducendo casualità nella scelta dei nodi sui quali fare *caching* di un documento ed attraverso i quali instradare i messaggi, ma questa soluzione euristica non è conciliabile con requisiti stretti di alte prestazioni ed efficienza.

Analisi e soluzioni per studiare e risolvere il problema dei fallimenti correlati tramite valutazione del grado di indipendenza tra nodi e uso parsimonioso delle risorse sono tuttora oggetto di ricerca.

6.7 Anonimato

Nei sistemi P2P per il data store distribuito i nodi possono essere programmati per funzionare in modo paritetico e quindi costruire un ambiente ideale per creare l'*anonymity set* necessario ad offrire proprietà di anonimato.

Nei capitoli 4.2, 5.1.3 e 5.1.4, sono già stati analizzati i livelli a cui è possibile garantire anonimato in un servizio di data store distribuito: può essere ignota l'identità di chi pubblica, di chi estrae o del nodo contenente un file ed inoltre un peer può non essere in condizione di conoscere il contenuto (o addirittura la presenza stessa) degli oggetti che ospita. Per una trattazione estesa delle tecniche con le quali Free Haven, Freenet e Mnemosyne implementano funzioni di anonimato si faccia riferimento ai rispettivi capitoli. Vengono qui di seguito analizzate le proprietà di anonimato in base ai criteri proposti da Roger Dingledine, Michael J. Freedman e David Molnar in [76].

Nei sistemi P2P per il data store distribuito i differenti tipi di anonimato possono essere ottenuti fondamentalmente grazie a due meccanismi: l'anonimato *computazionale* e anonimato *perfect-forwarding*.

Per anonimato *computazionale* si intende una forma di anonimato che potrebbe essere possibile infrangere grazie a risorse computazionali immense o conoscenze tecniche adeguate. Si presuppone quindi che l'avversario da cui ci si vuole proteggere sia dotato di tecnologie ragionevolmente limitate. Si noti che l'identità protetta con l'anonimato computazionale potrebbe essere svelata a posteriori, in quanto possono restare tracce delle comunicazioni su cui lavorare in un secondo tempo a transazione conclusa. Un esempio di questo tipo di anonimato è lo pseudonimo di un nodo Free Haven ottenuto calcolando il valore di hash di un suo identificativo.

L'anonimato *perfect-forwarding* si ottiene quando il sistema non lascia tracce della transazione effettuata che possano essere sfruttate per identificare i due *endpoint*, anche in caso uno dei due nodi sia compromesso dall'attaccante. Questa forma di anonimato ad esempio si ottiene in Freenet con le componenti stocastiche introdotte nel routing e in Free Haven grazie al canale di comunicazione anonimo Tor. L'anonimato *perfect-forwarding* si dimostra più difficile da assicurare rispetto a quello computazionale perché deve far fronte alla minaccia dell'analisi del traffico e ad un avversario che possa osservare a lungo il sistema per inferire informazioni sulla transazione vittima.

La tabella 6.1 illustra le forme di anonimato impiegate a diversi livelli del sistema nelle reti P2P per il data store distribuito analizzate nei capitoli precedenti.

rete	divulgatore		lettore		nodo		contenuto	
	comp	p-f	comp	p-f	comp	p-f	comp	p-f
Mnemosyne							X	
Free Haven	X	X	X	X	X		X	
Freenet	X	X	X	X			X	

Tabella 6.1: Anonimato computazionale e *perfect-forwarding* nei sistemi P2P esaminati.

Mnemosyne offre l'anonimato dei contenuti, in quanto i documenti memorizzati su un nodo sono protetti sia dalla steganografia che dalla cifratura. Il proprietario di un nodo, non conoscendo la chiave steganografica di un documento, non può sapere dove sono collocate le parti del documento sullo storage. Inoltre un attaccante, anche se riuscisse a ottenere tutte le parti necessarie a ricostruire il documento completo, non conoscerebbe in ogni caso il contenuto del file senza la chiave utilizzata per la cifratura. Entrambe queste forme di anonimato sono di tipo computazionale.

Nella rete Free Haven l'identità di chi pubblica un documento è celata dal canale di comunicazione sottostante (Tor o Mixminion) che utilizza entrambi i tipi di anonimato. Inoltre il nodo che ha inizialmente diffuso il file è ignoto in quanto non è possibile discernere fra le parti nuove appena immesse nel sistema e quelle regolarmente scambiate con gli altri nodi per mantenere la reputazione: questo è un tipo di anonimato *perfect-forwarding*.

L'identità di chi ricerca ed estrae un documento è protetta dal *reply block* differente per ogni richiesta in caso si usi Mixminion o dalla comunicazione broadcast attraverso Tor: l'anonimato è *perfect-forwarding*.

I nodi effettuano le transazioni fra di loro utilizzando degli pseudonimi, identificativi della reputazione che possono essere violati in un secondo tempo: questa forma di anonimato è soltanto computazionale.

Infine l'anonimato del documento è assicurato dall'utilizzo di un IDA, che impedisce di ricostruire il file da pochi frammenti eventualmente in possesso dell'avversario: anche questo anonimato è computazionale.

In Freenet, l'anonimato del contenuto è assicurato dal fatto che un nodo non conosce il nome di un file, ma solo il valore dell'hash calcolato a partire da esso o da una stringa descrittiva, per cui non può individuarlo: si tratta di un anonimato computazionale.

Se si richiede un file attraverso un nodo, il documento viene reperito e memorizzato su di esso e pertanto l'anonimato di un nodo che ospita un file non è garantito. Si noti tuttavia che il nodo su cui risiedeva il file prima della richiesta rimane sconosciuto.

L'anonimato di chi pubblica e di chi richiede un contenuto è garantito dal fatto che il comando di inserimento o di estrazione di una chiave avviene senza conoscere il nodo destinatario, ma viene effettuato un instradamento senza che un nodo conosca il percorso completo. Questo anonimato è sia di tipo computazionale, che di tipo *perfect-forwarding*.

6.8 Differenze ed equilibri: tutti i nodi sono uguali, ma alcuni sono più uguali degli altri

Nella sezione precedente si è evidenziato come in una rete P2P sia possibile disporre di un insieme di nodi che sono trattati in modo uguale e mostrano comportamenti equivalenti. Questa assunzione è ragionevole da un punto di vista teorico e semplifica largamente la modellazione e lo studio degli

Figura 6.7: Proprietà concorrenti in un sistema P2P per il data store distribuito.

algoritmi che regolano il sistema, ma in realtà alcuni peer sono *più uguali* degli altri per via di caratteristiche che possono accomunarli, già accennate in sezione 6.2:

- Risorse: capacità di calcolo, memoria, storage, connettività.
- Gestione: professionale o amatoriale.
- Disponibilità: *high-availability* del calcolatore, connettività *always-on* oppure *on-demand*.
- Posizione: nella rete (*core* o *edge*), locazione fisica (luogo ad accesso pubblico o privato, server-farm o ufficio/casa) e posizione geografica.

Ignorando le differenze tra i nodi, il sistema P2P per il data store distribuito è progettato per adeguarsi al minimo comune denominatore dei peer. Sfruttando le differenze si può invece aspirare ad offrire un servizio orientato verso prestazioni ed affidabilità ed è la strada intrapresa da OceanStore. Questa scelta sembra la più promettente per realizzare servizi che possano attirare una grande base di utenti e costituire anche un prodotto economicamente di successo.

Scegliere quali differenze tra i nodi favorire e come servirsene significa modificare gli equilibri che sussistono tra le principali proprietà di un sistema P2P per il data store distribuito, rappresentate in figura 6.7, propendendo per alcune a discapito di altre.

Per aumentare l'affidabilità del sistema ci si può servire di nodi attivamente e professionalmente gestiti, ad alta disponibilità, ospitati in server-farm e che eseguono un protocollo di Byzantine Agreement: è sufficiente assicurarsi che un terzo di questi nodi non venga compromesso perché il sistema possa funzionare correttamente. In questo modo la responsabilità del buon funzionamento della rete non viene dispersa in modo uniforme tra i peer, ma addensata su alcuni di essi. Come effetto collaterale tali nodi diventano anche i più appetibili per un attaccante che voglia compromettere il sistema. Una debolezza presente in questi nodi ha un impatto maggiore sulla sicurezza del sistema ed il rischio non è più distribuito in egual misura su tutti i partecipanti.

Affinché il data store sia permanente è necessario incentivare il buon comportamento degli utenti e scoraggiare chi persegue il solo vantaggio personale: si può impiegare un sistema di reputazione o di micropagamenti, ma la loro implementazione in un contesto P2P pone ancora numerose difficoltà e va progettata attentamente. Usando i micropagamenti si rischia di diminuire le prestazioni percepite dagli utenti o escludere dal sistema chi non dispone di risorse sufficienti. Con la reputazione bisogna far fronte alla minaccia dello *pseudospoofing*.

Per incrementare le prestazioni e la scalabilità del sistema si possono sfruttare i nodi che dispongono di migliori risorse di connettività e spazio di memorizzazione, in modo da favorire la disponibilità dei file e il numero di hop necessari per raggiungerli. Se la rete è sbilanciata e favorisce la comunicazione con i peer che mostrano prestazioni migliori diventa però più semplice sferrare attacchi *Sybil* ed *Eclipse*. Le prestazioni aumentano con l'aumentare dei nodi della rete ed allo stesso tempo in modo speculare gli utenti sono attratti da un servizio che garantisce elevate prestazioni. Gli utenti vogliono sia funzionalità personalizzabili che facilità d'uso, ma soddisfarli è un compito complesso perché questi due requisiti sono in genere contrastanti.

È evidente quindi come la progettazione di un sistema P2P per il data store distribuito richieda di operare delle scelte attente, accettare compromessi tra requisiti diversi e decidere verso quali obiettivi orientarsi.

Quando entra in gioco l'anonimato tutto si complica ulteriormente, le scelte progettuali hanno ricadute maggiori e i vincoli sono più stringenti. Per un sistema anonimo P2P per il data store distribuito l'eguaglianza dei nodi costituisce la condizione necessaria alla creazione dell'*anonymity set*. La fiducia verso i nodi è distribuita uniformemente su tutta la rete anonima perché non è possibile fare affidamento su peer di cui l'identità non può essere conosciuta con certezza, ad esempio per mezzo di una certificazione via PKI. Senza disporre di nodi fidati e controllabili il *Byzantine Agreement* è inefficace e non si può usare per serializzare le operazioni sullo storage.

Se tutti i peer sono trattati in modo paritetico è possibile che verso i nodi con scarse risorse di banda sia instradato più traffico di quanto essi possano smaltire. Per evitare il conseguente calo di prestazioni si può istruire il sistema P2P affinché privilegi per l'instradamento i nodi con connettività migliore, ma le considerazioni fatte riguardo il sistema di *entry guard* in Tor e il *Next Generation Routing* in Freenet Light insegnano che discriminare tra utenti "buoni" e "cattivi" in base a metriche che tengono conto di prestazioni e risorse può essere pericoloso per la privacy. Il rischio è che avversari che dispongono o dichiarano di offrire le risorse di un *supernodo* possano attrarre traffico e file da memorizzare così da minare la replicazione e la disponibilità dei dati e poter rompere l'anonimato degli utenti attraverso l'analisi del traffico.

La necessità di oscurare l'associazione tra nodi e file ospitati porta a realizzare reti P2P *unstructured*: la ricerca di un documento non è deterministica e quindi un file presente nel data store potrebbe non essere trovato, influenzandone negativamente la disponibilità.

I problemi riscontrati dal progetto Free Haven pongono seri interrogativi sulla possibilità di assicurare la persistenza dello storage per mezzo di un sistema di reputazione distribuito.

Nelle attuali reti P2P garantire l'anonimato implica quindi introdurre inefficienze a livello computazionale, di banda e di memorizzazione. Se questo determina basse prestazioni del sistema P2P per il data store distribuito, allora gli utenti non sono incentivati ad unirsi alla rete. Purtroppo le garanzie di anonimato e le prestazioni di una rete P2P anonima migliorano proprio all'aumentare degli utenti e del traffico perché si espande l'*anonymity set* e aumenta la disponibilità dei documenti. Il problema è circolare e pertanto difficile da risolvere.

Nella fase di lancio di un nuovo sistema anonimo P2P il fattore umano costituisce un aspetto critico. Inizialmente, quando i nodi e gli utenti della rete P2P sono pochi, l'*anonymity set* è di dimensioni contenute. Quando i potenziali nuovi utenti devono scegliere tra un nuovo sistema per il data store

distribuito dotato di proprietà di anonimato migliori e un vecchio sistema meno avanzato (si pensi all'attuale coesistenza di Freenet Light e Freenet Dark), la scelta ricade probabilmente sulla rete che conta più utenti e promette quindi un *anonymity set* più grande. A rompere questo stallo (la privacy attira gli utenti, ma gli utenti creano le condizioni perché ci sia privacy) possono contribuire gli utenti che hanno bisogno solo di un basso livello di anonimato e accettano di partecipare alla rete anche se l'*anonimity set* è modesto.

Anche la *reputazione sociale* riguardo la tipologia degli utenti del sistema è importante: una rete P2P che si ritiene usata per lo più da pedofili e terroristi difficilmente attrarrà una grande utenza ed anzi rischierà di attirare avversari potenti che non avranno alcuno scrupolo a rompere l'anonimato degli utenti e svelarne pubblicamente l'identità.

Un altro elemento che può influire sul successo della fase di *bootstrap* è la *percezione* della *facilità d'uso* (*usability*) che gli utenti hanno del sistema: se gli utenti *pensano* che la rete sia popolare allora saranno disposti ad unirsi ad essa. In una rete P2P per il data store distribuito l'usabilità diventa un vero e proprio requisito per la sicurezza [77]. Un software P2P comodo da usare è eseguito da molti utenti e più frequentemente, si pensi banalmente al numero di utenti disposti ad operare su uno storage distribuito a bassa latenza rispetto ad uno ad alta latenza.

Il software non dovrebbe presentare troppe opzioni configurabili dall'utente finale, anche se queste possono essere viste in modo positivo da un fetta dell'utenza. È rischioso inserire opzioni che possono influire sui meccanismi volti a garantire la segretezza o l'anonimato delle comunicazioni, perché non si può assumere che l'operatore sappia intervenire su questi parametri con cognizione di causa. Introdurre opzioni rende più difficile eseguire un *auditing* accurato del codice perché aumentano le linee di codice, cresce esponenzialmente il numero di configurazioni possibili e le combinazioni di opzioni più rare subiscono un testing sul campo insufficiente.

Secondo queste osservazioni il software P2P per il data store distribuito dovrebbe essere fornito *out-of-the-box* e preconfigurato correttamente per costruire una grande rete di nodi molto interconnessi ed il più possibile uguali ed autonomi. Un software di questo genere è un bersaglio appetibile per essere infettato da un *worm*, per cui nel suo sviluppo è preferibile scegliere linguaggi di programmazione altamente tipizzati e che non permettano accessi diretti alla memoria, seguire delle *best practice* di programmazione sicura ed condurre un attento *auditing* del codice.

Questa analisi ha illustrato le difficoltà che si riscontrano nel conciliare requisiti diversi in una rete P2P per il data store distribuito. I sistemi considerati più che risolvere i conflitti operano delle scelte che li orientano verso obiettivi ed impieghi specifici. Mentre le tecniche per far coesistere caratteristiche contrastanti, ad esempio alte prestazioni ed anonimato, sono tuttora argomento di ricerca, una soluzione attualmente poco esplorata è la creazione di *gateway* che rendano interoperabili le diverse reti. In questo caso la sfida più difficile è rendere compatibili i diversi protocolli di routing P2P, nonché i metodi di ricerca e memorizzazione dei file.

6.9 Chaos, ordine e sicurezza

Dall'analisi condotta si comprende come la sicurezza di una rete P2P per il data store distribuito sia un processo che investe il sistema in tutto il suo ciclo di vita (la progettazione, lo sviluppo del software, il *testing*, il *deployment*, la promozione e la gestione del servizio, l'educazione degli utenti) ed ogni sua proprietà.

L'architettura P2P rende il compito di controllare l'accesso ai dati dello storage, garantirne la riservatezza, l'integrità e la disponibilità molto più difficile che nel tradizionale paradigma di servizio centralizzato. Le problematiche di sicurezza aumentano per via dell'incremento di complessità, della

distribuzione del *trust* su tutti i peer, della partecipazione dinamica e delle potenziali enormi dimensioni della rete.

In reti P2P strutturate è possibile rafforzare le difese del sistema riponendo un alto livello di fiducia sul gruppo di nodi gestiti professionalmente dal fornitore del servizio e sui quali si possono attuare i ben noti meccanismi di prevenzione, rilevamento ed investigazione.

Con l'aumentare delle dimensioni del sistema tuttavia questo non è più sufficiente ad assicurare la sicurezza. John Kubiatowicz, leader di OceanStore, suggerisce una tecnica per estrarre garanzie in grandi reti P2P, indicata come "Stabilità attraverso la Statistica" [78]. La rete P2P è vista come un sistema termodinamico. In termodinamica una proprietà di un aggregato (la temperatura) è stabile nonostante i singoli elementi (le molecole) agiscano in modo altamente variabile. Analogamente il *comportamento aggregato* di molti peer può raggiungere uno stato stabile nonostante i singoli elementi che interagiscono nel sistema possano dimostrarsi ostili oppure malfunzionanti. Esiste una competizione tra ordine e chaos.

L'ordine latente nel sistema P2P è alimentato ad esempio dalle funzionalità del DOLR che stabilisce associazioni precise tra i peer e i documenti che essi ospitano e permette di localizzare in modo efficiente gli oggetti. Un altro esempio è l'IDA che mantiene un legame matematico tra i frammenti di un file sparsi casualmente su nodi diversi. L'ordine latente interviene attraverso questi meccanismi per stabilizzare il comportamento della rete in caso di malfunzionamento di alcuni peer permettendo comunque di recuperare integralmente e attraverso un percorso ottimizzato il file richiesto.

Il chaos aumenta nella rete col trascorrere del tempo, con l'accumularsi dei fallimenti e con lo spostamento della distribuzione dei tempi di servizio verso valori peggiori. Questo fenomeno rispecchia l'aumento dell'entropia in un sistema chiuso sancito dal secondo principio della termodinamica.

Il sistema P2P può sfruttare l'ordine latente se riesce a mantenere il livello di entropia entro una certa soglia, si pensi al limite di un terzo di nodi ostili perché il Byzantine Agreement Protocol abbia successo. Questo è possibile implementando nelle reti sia funzionalità di organizzazione e configurazione automatiche, sia di *introspezione*, ovvero di impiegare le risorse inutilizzate per osservare e valutare il comportamento del sistema ed attivare di conseguenza opportuni meccanismi correttivi ed adattativi.

I tre principi cardine da seguire sono quindi:

- Ridondanza (*redundancy*):
  utilizzare più risorse di quelle strettamente necessarie così da aumentare la resilienza a malfunzionamenti ed attacchi e servirsi di quelle inutilizzate per analizzare il comportamento del sistema.

- Sostituzione (*replacement*):
  individuare tramite meccanismi opportuni le componenti che falliscono per spostarsi su altre funzionanti.

- Ripristino (*restoration*):
  ripristinare l'ordine latente per ridurre l'entropia, ad esempio con la periodica aggiunta di nuovi nodi potenti e affidabili ad opera del fornitore del servizio.

Anche la *casualità* può essere funzionale alla stabilità e sicurezza del sistema, evitando che esso propenda sistematicamente verso una certa configurazione e proteggendolo da alcune tipologie di attacco (si pensi all'arbitrarietà della distribuzione dei frammenti di un file secondo un IDA che impedisce di sapere a priori su quali nodi venga memorizzato un documento). Le tecniche probabilistiche divengono centrali nei sistemi anonimi P2P per il data store, dove il livello di introspezione raggiungibile è necessariamente basso; in questo caso lo studio del comportamento del sistema può essere condotto attraverso simulazioni.

Appendice A

Mixminion

Mixminion [79] [80] è una diffusa implementazione di un *remailer anonimo* (*anonymous remailer* [81]), ovvero un meccanismo per la trasmissione di messaggi di posta elettronica in modalità anonima.

Durante la trasmissione di un'e-mail, pur applicando una qualche forma di cifratura del contenuto, utilizzando tecnologie quali PGP e S/MIME, sono ancora presenti troppe informazioni note ad un osservatore ostile sulla rete: soprattutto identità del mittente e destinatario, ma anche informazioni sul programma di posta, lunghezza del messaggio e così via.

L'identità del mittente di un messaggio inviato tramite Mixminion, assieme alle altre informazioni sensibili, non vengono svelate al destinatario, né ad un osservatore malevolo; si dice che il contenuto è scorrelato dall'autore.

L'anonimato viene raggiunto facendo passare il messaggio in forma cifrata attraverso diversi nodi, che conoscono solamente i nodi adiacenti.

L'implementazione di Mixminion permette la trasmissione anonima di generici pacchetti di dati, non solamente lo scambio di messaggi di posta elettronica. Esso può quindi essere utilizzato come livello di rete di base per applicazioni elaborate con forti requisiti di privacy, come ad esempio Free Haven (si veda il capitolo 4).

Mixminion è sviluppato da Roger Dingledine, Nick Mathewson e George Danezis; il primo di questi è lo sviluppatore principale di Free Haven e con il secondo ha sviluppato Tor.

A.1 Evoluzione dei remailer

I protocolli di remailer anonimi sono evoluti nel tempo. Si distinguono tre generazioni, precedute da un tipo di remailer non propriamente anonimo; Mixminion è l'implementazione standard di riferimento della terza e ultima generazione.

A.1.1 Pseudonym remailer

La prima forma di remailer, indicato successivamente come remailer anonimo di tipo 0, è in verità un *pseudonym remailer*, in quanto si tratta di un singolo host che smista i messaggi secondo degli pseudonimi ivi memorizzati.

Il server di questo tipo più importante è stato `anon.penet.fi`; esso offriva a metà degli anni '90 un servizio di SMTP e POP che prima di inoltrare i messaggi di posta toglieva le informazioni sul mittente contenuti nell'header e utilizzava come identificativo del mittente uno pseudonimo. Per le risposte, il destinatario inoltrava il messaggio all'indirizzo pseudonimo, ed il remailer gestiva la corrispondenza fra pseudonimo e mittente.

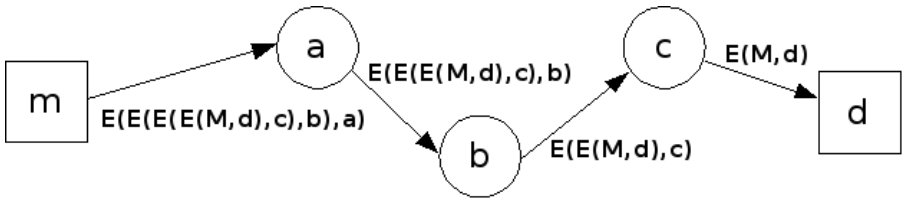

Figura A.1: Sequenza di cifrature di un messaggio M diretto al destinatario d. Il mittente m provvede a cifrare in cascata tramite l'algoritmo a chiave asimmetrica $E(t, k_p)$ utilizzando per ogni passo la chiave pubblica del nodo successivo.

Il problema fondamentale di questo approccio, che è stata anche la principale causa della chiusura di `anon.penet.fi`, è la necessaria presenza sul server di una lista di corrispondenze fra mittente effettivo e pseudonimo utilizzato. Oltre alla scarsa anonimità del sistema, un attaccante che monitori la rete può conoscere il destinatario da un messaggio in transito, in quanto la consegna del messaggio è affidata al classico SMTP.

A.1.2 Cypherpunk remailer - Tipo 1

La prima forma di inoltro dei messaggi di posta elettronica propriamente anonima si basa su una rete di *cypherpunk remailer*; dei nodi che, accettando messaggi di posta, possibilmente cifrati con algoritmi a chiavi asimmetriche (utilizzando programmi quali PGP o GPG), li inoltrano rimuovendo le informazioni riguardanti il mittente ed il destinatario. La destinazione è specificata in appositi campi tra l'header e il corpo dell'email.

Il messaggio deve essere preventivamente costruito ad-hoc dal mittente, che decide un percorso casuale attraverso i remailer per raggiungere il destinatario. Infatti cifrando varie volte in modo opportuno il messaggio, è possibile utilizzare più remailer in cascata, ognuno dei quali può decifrare una parte di header del messaggio e scoprire l'indirizzo soltanto del prossimo remailer; così nessun remailer conosce il contenuto del messaggio, soltanto il primo può conoscere l'identità del mittente e soltanto l'ultimo conoscere il destinatario. La figura A.1 illustra un esempio di percorso utilizzando 3 remailer, e la sequenza di cifrature necessarie.

Il limite principale di questo meccanismo rimane l'impossibilità di gestire le risposte; il destinatario di un messaggio anonimo, non avendo riferimenti sul mittente, non può rispondergli. Inoltre i messaggi sono tracciabili, sono troppe le informazioni sulla dimensione e sul flusso dei messaggi note ad un eventuale malintenzionato.

A.1.3 Nymserver

Per ottenere un riferimento anonimo del mittente, ci si può appoggiare ad un server che gestisce un riferimento al mittente; tali server vengono detti *pseudonym server*. L'informazione sulla corrispondenza tra identità del mittente e pseudonimo fornito non è memorizzata direttamente; il server infatti conosce per ogni pseudonimo un blocco precalcolato dal mittente (*reply block*) per inoltrare la risposta tramite una catena di remailer anonimi.

Il mittente che vuole quindi permettere risposte al suo messaggio deve prima dell'invio del messaggio accordarsi con il nymserver per predisporre il reply block, e quindi nei campi header riservati dal meccanismo Cypherpunk inserire il suo pseudonimo.

Si noti che il destinatario può anche comunicare al nymserver tramite una catena di remailer;

sia identità del mittente che del destinatario sono quindi nascoste agli occhi del nymserver, che può scaricare ogni responsabilità (*plausible deniability*).

Il server che storicamente offre questo servizio è `nym.alias.net`, collocato presso il MIT.

Per necessità è nata una classe di server che offre un ulteriore servizio: l'elenco dei remailer conosciuti. Generalmente oltre al nome del nymserver presentano statistiche (effettuate con l'invio di messaggi di prova) sulla loro disponibiltà e sui servizi offerti; sono quindi noti con il nome di *statistics server* o *directory server*.

A.1.4 Mixmaster remailer - Tipo 2

I nodi di tipo cypherpunk sono evoluti in mixmaster remailer, la seconda generazione di remailer anonimi; il sistema nuovo è quindi retrocompatibile.

Questi nodi cercano di limitare i problemi di tracciabilità effettuando diverse operazioni:

- Spezzano prima dell'inoltro i messaggi in pacchetti di dimensione fissa, con eventuale riempimento con dati casuali dei blocchi troppo corti.
- Effettuano l'invio dei pacchetti in tempi differenti, tenendo sempre un numero minimo di pacchetti da inviare (*pooling*).
- Inviano dei pacchetti finti (*dummy message*) per evitare la situazione in cui solo un utente sull'intera rete comunica, rendendo evidente il flusso di dati.
- Il messaggio è ricomposto a destinazione, tramite un'apposita applicazione.

Anche con i remailer di tipo mixmaster rimane il problema della risposta ai messaggi, essendo la comunicazione unidirezionale; anche qui ci si può appoggiare ad un nymserver per tenere un riferimento al mittente.

A.1.5 Debolezze principali

La rete Mixmaster con i Nymserver finora descritta è soggetta ad alcuni attacchi noti [82]:

- *Passive subpoena attack.*
  Un malevolo osservatore esterno può registrare uno storico dei pacchetti passati da un certo nodo e tentare di interpretare il contesto dei blocchi dei messaggi (attacco a citazione). Inoltre il compito può essere reso più facile se è possibile riconoscere i propri messaggi generati ad-hoc in un flooding attack.
- *Active subpoena attack.*
  Un attaccante del sistema che dispone di un nodo remailer può comunque tentare l'attacco a citazione sopra descritto, dato che dispone dei pacchetti transitati. Una soluzione ai due attacchi a citazione potrebbe essere l'introduzione di un meccanismo di cifratura dei collegamenti fra i nodi utilizzando una rotazione di chiavi temporanee cambiate in un breve arco di tempo.
- *Passive partition attack.*
  Un attaccante può sapere quali nodi client aggiornano spesso la loro lista di server e quali invece tengono la loro lista statica, osservando le connessioni ai directory server, che transitano in chiaro. Dopo un certo lasso di tempo sapere quali nodi hanno una lista aggiornata e quali no rende più facile il tracciamento. Per risolvere la debolezza tutti i nodi dovrebbero utilizzare la stessa politica di aggiornamento della lista.

- *Active partition attack.*
  Il directory server stesso può essere ostile e presentare solo una parte dei nodi remailer, secondo le necessità per il tracciamento. Per risolvere questa eventualità si potrebbe utilizzare una struttura di firme a chiave pubblica, ma è necessario avere alcuni directory server sempre fidati.

- *Tagging attack.*
  È presente un controllo sugli header dei remailer tramite un hash, ma non per i campi non sensibili nell'header dell'email; è possibile per un attaccante che possiede più nodi marcare i messaggi e riconoscerli per tracciare il loro percorso. Una soluzione potrebbe essere quella di calcolare l'hash su tutto l'header, ma il tagging potrebbe comunque essere effettuato inserendo una marcatura prima o dopo il messaggio.

A.2 Mixminion - Tipo 3

Mixmaster rappresenta una serie di accorgimenti per "rattoppare" Cypherpunk, aumentando sia la complessità del sistema che quindi l'inaffidabilità; oltre ai problemi noti sopra elencati, la gestione delle risposte rimane comunque macchinosa e scomoda. Per questi motivi nel 2003 nasce Mixminion, che parte dall'esperienza dei due tipi di remailer precedenti e raccogliendo i loro meccanismi di funzionamento li migliora, rendendo inoltre più semplice la gestione delle risposte [83].

La rete Mixminion segue il modello a tratte già illustrato: ogni nodo conosce soltanto l'identità del nodo precedente e di quello successivo nel tragitto di una comunicazione, a causa della cifratura a chiavi asimmetriche. Anche qui i nodi effettuano le varie operazioni di frammentazione e dispersione dei pacchetti, con le migliorie necessarie a risolvere i problemi presentati.

Una delle novità introdotte riguarda il protocollo di trasmissione fra i nodi remailer, che allarga notevolmente il suo ambiente di utilizzo: non avviene più a livello di SMTP, ma direttamente su TCP/IP, permettendo lo scambio agevole di qualsiasi tipo di dati.

Uno dei più grandi miglioramenti riguarda il diverso approccio al percorso dei pacchetti: i messaggi trasmessi tra i nodi sono composti da due header e un payload. I due header specificano due percorsi distinti attraverso i nodi, uno dei quali deve essere segnato: in corrispondenza del nodo segnato (*crossover point*), i due header vengono scambiati. Questo meccanismo permette tre tipi di comunicazione:

- *Forward message*, in cui il mittente rimane anonimo e decide il percorso dei due header, calcolati opportunamente per poter raggiungere il destinatario noto. Solo questa modalità trasmette messaggi in forma binaria e permette l'invio di dati non solo testuali.

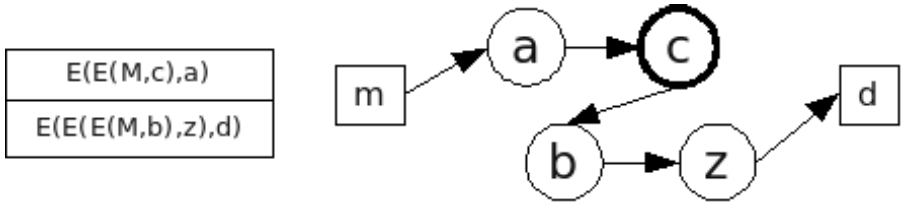

Figura A.2: Forward message.

- *Direct reply*, in cui il destinatario utilizza un reply block simile a quello di Nymserver per rispondere al mittente; questo gli deve già essere noto, per esempio se è stato comunicato durante un messaggio precedente.

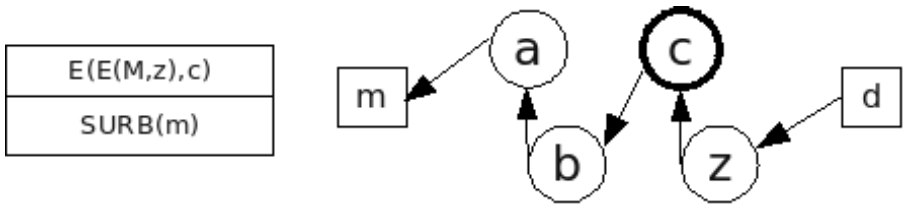

Figura A.3: Direct reply.

- *Anonymous reply*, in cui sia il mittente che il destinatario sono sconosciuti. Il mittente sceglie il percorso del primo header e utilizza come secondo header il reply block del destinatario.

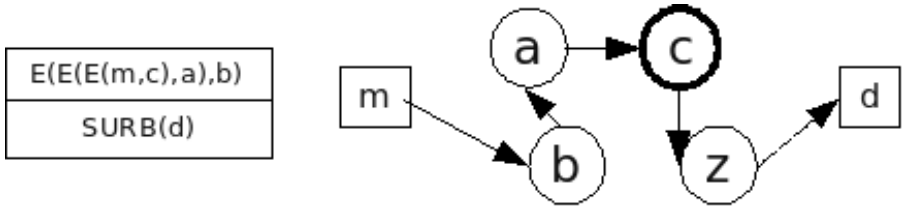

Figura A.4: Anonymous reply.

I reply block, qui chiamati SURB (*Single Use Reply Block*) poiché devono essere utilizzati soltanto una volta, sono l'equivalente del blocco precalcolato memorizzato dal Nymserver. Esso in pratica contiene il percorso cifrato a cascata da seguire per ritornare al mittente. Questi blocchi sono inviati anonimamente a vari nodi della Mixminion scelti a caso per permettere ad eventuali messaggi di risposta in attesa ivi memorizzati di giungere a destinazione. Nel caso quindi di *anonymous reply*, se il SURB è già noto lo si imposta come secondo header, altrimenti si lascia al destinatario il compito di recuperare il messaggio.

Per contrastare la marcatura dei pacchetti da parte di alcuni nodi malevoli fra i quali è presente il nodo di crossover, è possibile utilizzare più di un crossover point; è statisticamente difficile scegliere casualmente più nodi malevoli per il crossover su una rete in gran parte affidabile.

In Mixminion alcuni nodi offrono funzionalità aggiuntive: statistiche di rete e *directory server*. Quest'ultimo servizio, che raccoglie e offre una lista di nodi presenti, è delicato in quanto può presentare una lista di nodi ad hoc per tentare un attacco a partizionamento (*partition attack*): infatti fornendo ad un soggetto da attaccare una lista di nodi controllati, è possibile tracciare la trasmissione. Per evitare ciò è necessario richiedere le liste di nodi da diversi directory server.

Un problema risolto in parte riguarda la possibilità per un nodo ostile di modificare i tempi di invio di alcuni messaggi aspettando per l'inoltro periodi in cui la rete è maggiormente scarica; questo problema è fortemente limitato dal pooling ma pur sempre possibile anche se con molta imprecisione. Inoltre il fatto stesso che la rete non sia uniformemente carica nel tempo può a lungo termine fornire informazioni sempre più dettagliate riguardo l'identità dei mittenti e dei destinatari; questo problema è comune e generalmente non risolto nei sistemi di comunicazione anonima.

Appendice B

TOR: The Second-Generation Onion Router

Tor è un overlay network distribuito pensato per anonimizzare le applicazioni a bassa latenza basate su TCP come *web browsing*, *secure shell* e *instant messaging*.

I client scelgono un percorso attraverso la rete e costruiscono un *circuito*. Il traffico fluisce sul circuito in *celle* di dimensione fissa che ad ogni nodo vengono private di un imbustamento crittografico usando una chiave simmetrica (questo procedimento ricorda il modo in cui una cipolla, in inglese *onion*, viene progressivamente sbucciata ed ha suggerito il nome *onion routing*) e quindi inoltrate verso il prossimo hop.

Il progetto Tor è gestito da The Free Haven Project e costituisce un elemento importante per la realizzazione di sistemi di comunicazione e pubblicazione anonimi e resistenti alla censura. Tor è sviluppato da Roger Dingledine, Nick Mathewson, Paul Syverson ed altri volontari e diretto da Shava Nerad; è basato sul concetto di Onion Routing [84] di prima generazione elaborato nello U. S. Naval Research Laboratory.

B.1 Modello di minaccia

Tor è progettato per realizzare una difesa contro un attaccante non-globale in grado di controllare e monitorare un sottoinsieme di nodi della rete. L'attaccante può iniettare, ritardare, alterare o scartare il traffico su alcuni collegamenti tra i nodi. Questo modello di minaccia è simile a quello assunto da altri sistemi anonimi a bassa latenza, ad esempio Freenet.

Tor tuttavia non tenta di garantire protezione rispetto a *traffic confirmation attack* in cui l'avversario osserva due utenti che sospetta stiano comunicando tra loro per confermare o respingere questa ipotesi. Tor intende piuttosto ostacolare la raccolta di ulteriori informazioni da parte di un attaccante che nutra un sospetto a priori molto debole riguardo l'identità degli attori della comunicazione.

Obiettivo primario dell'attaccante è associare chi inizia una connessione con la sua destinazione e viceversa; obiettivo secondario è mettere in relazione le transazioni, ovvero le connessioni di rete, per stabilire se esse provengano dalla stessa origine così da delineare un profilo dell'utente osservando i *pattern* delle sue abituali comunicazioni.

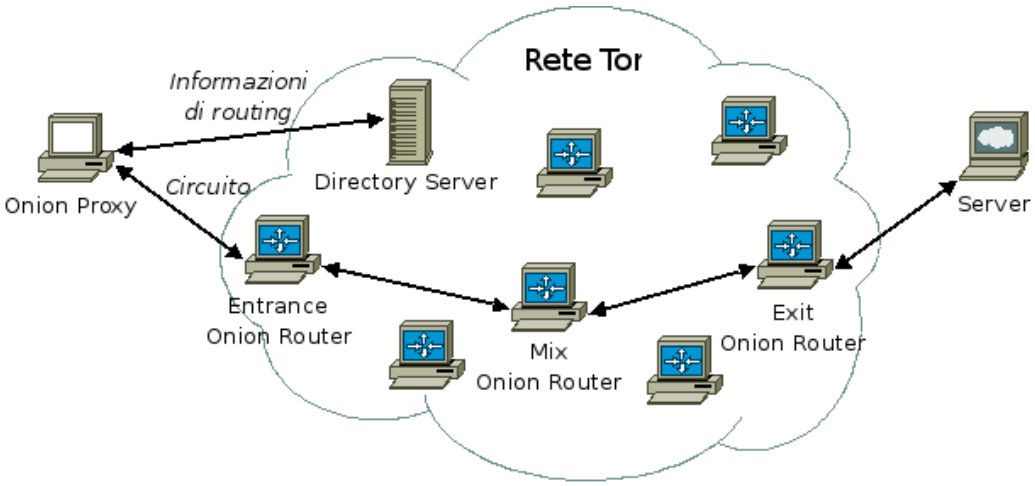

Figura B.1: La rete Tor.

B.2 Architettura

La rete Tor permette il trasporto di stream TCP in modo anonimo; non integra il tunneling per protocolli non basati sugli stream, come UDP, che è demandato ad eventuali servizi esterni.

Un insieme di *Onion Router* (OR) rappresenta il componente server della rete e agisce come relay per un certo numero di stream di comunicazione appartenenti a utenti diversi inoltrando il traffico nel core della mix network.

Un *Onion Proxy* (OP) è la parte client della rete che inietta il traffico dell'utente nella rete degli OR. L'OP ritira informazioni dalle *directory*, stabilisce circuiti attraverso la rete, gestisce le connessioni richieste dalle applicazioni dell'utente; spesso l'OP è un servizio in esecuzione sul calcolatore dell'utente.

L'OP non effettua nessuna normalizzazione del protocollo applicativo. Se l'utente desidera restare anonimo rispetto al server destinazione che risponde alle sue richieste mentre sta usando un protocollo complesso e variabile come l'HTTP, allora Tor deve essere affiancato ad un filtering proxy, ad esempio Privoxy, per nascondere le differenze tra client, disattivare le caratteristiche del protocollo che possono lasciar trapelare informazioni sull'identità dell'utente e anche per delegare ad esso la risoluzione dei nomi DNS perché venga effettuata tramite il *lookup service* della rete Tor così da non rivelare la destinazione ad un ascoltatore. Grazie a questa separazione dal componente che normalizza il protocollo applicativo, Tor può essere usato anche per accedere a servizi anonimi per la rete, ma autenticati nei confronti di chi offre il servizio.

Gli OP accettano tramite un'interfaccia SOCKS 4 gli stream TCP avviati dall'utente e ne fanno il *multiplexing* attraverso i circuiti. Un circuito è un percorso di tre OR (per default, ma questo numero è configurabile) sulla rete Tor dall'OP al server destinazione desiderato. Il primo OR su un circuito è chiamato *entrance router*, il secondo OR è detto *mix router* e l'ultimo hop è l'*exit router*.

I flussi TCP in transito attraverso la rete Tor sono divisi e impacchettati in celle. Ogni cella ha dimensione fissa pari a 512 byte, ma per ridurre la latenza può contenere un payload utile più corto e padding per i byte rimanenti; questo è importante specialmente per supportare protocolli interattivi, quali SSH, che inviano sulla rete messaggi contenenti pochi caratteri digitati alla tastiera.

Le celle sono memorizzate in buffer separati per ogni stream ed escono dal nodo in modalità round-robin, secondo una scansione circolare dei buffer delle connessioni. Questo assicura che tutte

le connessioni vengano ritardate in modo equo e rappresenta una strategia comune per garantire il miglior servizio best effort. Quando il buffer di una connessione è vuoto viene saltato ed è spedita una cella dal successivo buffer di un'altra connessione che non risulta vuoto. Dal momento che uno degli obiettivi di Tor è offrire comunicazioni a bassa latenza, le celle non subiscono esplicitamente un ritardo, un riordinamento, un raggruppamento od uno scarto, al di là della semplice strategia appena descritta: Tor non esegue alcun esplicito riordinamento (*mixing*) delle celle sui nodi.

Tor implementa alcuni accorgimenti per offrire equità di servizio, limitare velocità e frequenza di trasferimento ed evitare la congestione del traffico sui nodi. Innanzitutto Tor impiega una strategia *token bucket* per assicurarsi che i volumi di traffico sul lungo termine vengano mantenuti al di sotto di un certo limite impostato da ogni operatore di nodo Tor; tuttavia questo sistema non impedisce a picchi di dati di propagarsi attraverso una connessione. Questi picchi sarebbero naturalmente limitati dalla massima ampiezza di banda di ciascun hop e potrebbero saturare la connessione di rete di alcuni nodi Tor.

Per evitare una situazione di congestione di questo tipo, è realizzato un doppio meccanismo a finestra a livello di circuito. Ogni stream è associato a due finestre, la prima (*packaging window*) descrive quante celle possono essere ricevute dall'OP dall'utente, mentre la seconda (*delivery window*) indica quante celle è permesso inviare al di fuori della rete. Se sulla rete sono in transito troppe celle non ancora accettate dalla destinazione finale, il nodo Tor smette di accettare altre celle fino a che la condizione di congestione non viene risolta. In questo modo il mittente non invia più dati di quanti la destinazione sia pronta ad accettare, evitando così di saturare i buffer dei nodi Tor intermedi; inoltre le connessioni possono avere solo un numero limitato di celle in transito sulla rete senza che abbiano ancora ricevuto conferma (acknowledge) e questo impedisce agli utenti di inondare la rete col loro traffico.

Il fulcro dell'architettura di rete di Tor è la rete a commutazione di circuito (*circuit switched network*). Il meccanismo con cui viene stabilita una connessione è attentamente progettato per preservare l'anonimato, impedendo ad un osservatore di collegare le informazioni crittografiche o ricostruire la rotta completa seguita dalla connessione, che rimane nota solo all'utente che ha originato il circuito.

L'OP di un utente che desidera iniziare a trasmettere un flusso dati, crea preventivamente un circuito in maniera incrementale (*telescopic path-building*), negoziando una chiave simmetrica con ogni OR del circuito, un nodo per volta.

L'OP dell'utente per prima cosa si collega ad un OR scelto in modo casuale, negozia la chiave segreta per mezzo del *self-signed ephemeral Diffie-Hellman key exchange* [85] e viene impiegato lo standard *Transport Layer Security* (TLS [86]) per stabilire un canale sicuro e proteggere le connessioni tra i nodi e garantire forward secrecy, impedendo a un attaccante di modificare i dati in transito o di impersonare un OR.

Da questo momento l'OP invia tutte le comunicazioni nel tunnel creato per mezzo di questo circuito e può connettersi ad altri due o più OR, scambiare chiavi con essi e proteggere così la comunicazione attraverso livelli multipli di cifratura. Ogni livello è decifrato da uno specifico nodo Tor usando la chiave simmetrica concordata per quel circuito e i dati in esso contenuti sono inoltrati al prossimo OR usando tecniche standard di etichettatura delle rotte.

In questo documento non si tratteranno attacchi alle componenti crittografiche di Tor, per cui non si approfondiranno gli elementi di questo argomento, ma è possibile consultare le specifiche di Tor [87] per maggiori informazioni a rigurado.

Infine, quando un certo numero di OR (di default tre) sono stati configurati e sono pronti a trasportare il circuito, l'utente può chiedere all'ultimo OR del circuito di collegarsi a una specifica porta TCP di un indirizzo IP remoto o di un nome di dominio. I dati a livello applicazione, come le

richieste HTTP o le sessioni SSH, possono quindi essere trasmessi sul circuito in modo trasparente all'applicazione e con le normali modalità.

Ogni OR può specificare le sue *exit policy*, ovvero a quali porte TCP dei server consente agli utenti di raggiungere quando è usato come exit router in un circuito.

Ogni circuito Tor può essere sfruttato per trasportare molti flussi TCP, tutti originati dallo stesso OP; questa caratteristica è utile per supportare protocolli come l'HTTP nei quali per ogni singola transazione possono servire molte connessioni, anche verso nodi diversi nella rete. I circuiti Tor inutilizzati hanno vita breve e sono rimpiazzati dopo pochi minuti. Quest'azione comporta la scelta di una nuova rotta attraverso la rete Tor, lo scambio di nuove chiavi e la configurazione dei tunnel cifrati.

Un problema nel design originale dell'Onion Router era l'adozione di uno *stream cipher* privo di controllo di integrità che esponeva il traffico ad un *malleability attack*: anche se l'attaccante non poteva decifrare le celle, qualsiasi modifica dei dati cifrati avrebbe creato un corrispondente cambiamento nei dati in uscita dalla rete. Tor risolve questa vulnerabilità per mezzo di un controllo di integrità ad ogni hop del circuito. Quando un utente negozia una chiave con un nuovo hop, essi inizializzano un digest SHA-1 derivato dalla chiave e quindi condividono inizialmente una quantità casuale nota solo a loro. I due estremi della connessione sommano progressivamente a questo digest SHA-1 il contenuto di ogni cella che creano e includono in ogni cella i primi quattro byte del digest corrente. Ogni coppia di nodi inoltre mantiene un digest SHA-1 dei dati ricevuti per verificare la correttezza degli hash arrivati.

Un altro difetto dell'Onion Router di prima generazione era l'utilizzo di aggiornamenti *in-band* dello stato della rete: ogni router inviava a tutti i suoi vicini un annuncio firmato, i quali lo propagavano oltre seguendo un meccanismo di flooding. Questa caratteristica permetteva ad un attaccante di diffondere false informazioni riguardo i gruppi di appartenenza dei router, la topologia e lo stato corrente della rete, realizzando un *partitioning attack* in cui la conoscenza della rete dei client vittima veniva alterata.

Tor utilizza un piccolo gruppo ridondato di OR ben conosciuti e detti Directory Server (DS) per rilevare i cambiamenti di topologia della rete e dello stato dei nodi. Gli OR periodicamente pubblicano sui DS un sommario firmato che riporta il proprio stato. I DS combinano queste informazioni con le proprie viste della topologia della rete e compilano una directory, cioè una descrizione firmata dello stato dell'intera rete. I client inizialmente sono precaricati con una lista dei DS e delle releative chiavi, per poter eseguire una fase di bootstrap in cui inizializzano la propria conoscenza della rete; in seguito periodicamente scaricano via HTTP la directory dai DS.

Tor consente inoltre agli operatori dei nodi di offrire hidden service, ovvero servizi la cui locazione è nascosta che garantiscono l'anonimato del server destinazione di uno stream TCP iniziato dall'utente (*responder/backward anonymity*) e resilienza agli attacchi DDoS: chi ad esempio ospita un webserver come hidden service permette agli utenti di contattarlo attraverso la rete Tor, senza rivelare l'indirizzo IP del server.

La figura B.2 indica le azioni che un client, Alice, ed un webservice offerto come hidden service, Bob, devono compiere coi loro rispettivi OP locali per stabilire una comunicazione (ogni freccia in figura indica un canale anonimo stabilito attraverso due o più OR intermedi):

- Bob genera un paio di chiavi pubbliche a lunga durata per identificare il suo servizio.
- Bob sceglie alcuni OR come suoi *introduction point*, ovvero i punti di contatto per il client, e se questi acconsentono crea un circuito verso ognuno di essi e li istruisce affinché attendano richieste da servire (1).
- Bob annuncia i suoi introduction point sul *lookup service* dei *Directory Server* (DS), firmando l'annuncio con la sua chiave pubblica (2).

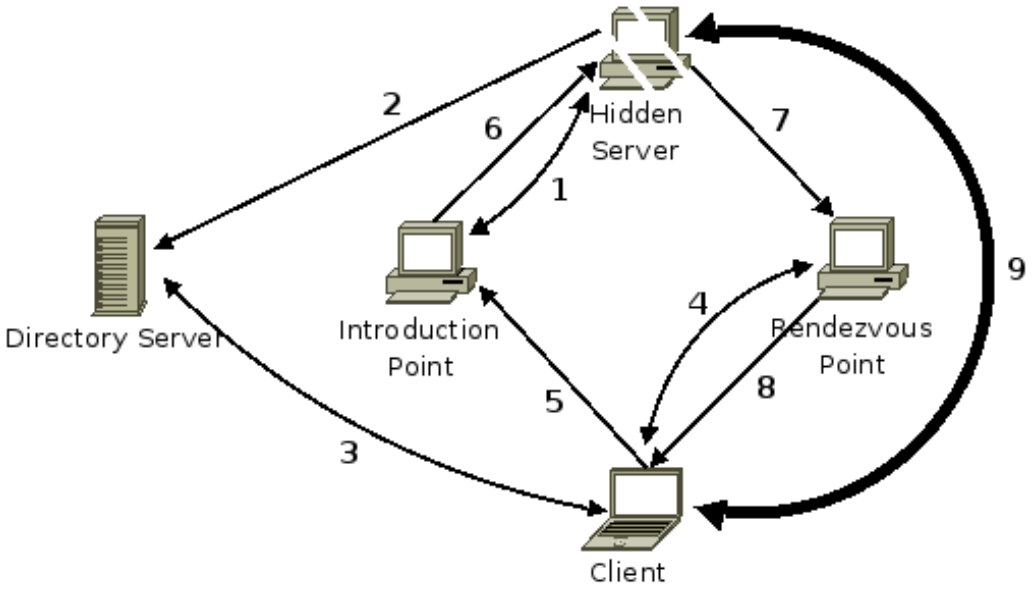

Figura B.2: Raggiungere l'hidden server.

- Alice viene a conoscenza con un meccanismo *out-of-band* del servizio di Bob, ritira dal lookup service i dettagli del servizio, inclusi gli indirizzi degli introduction point (3).
- Alice sceglie un OR come suo rendezvous point per le connessioni al servizio di Bob, costruisce un circuito verso il rendezvous point e gli fornisce un *rendezvous cookie* scelto in modalità casuale per riconoscere Bob (4).
- Alice si connette attraverso uno stream anonimo ad uno degli introduction point di Bob e gli invia un messaggio cifrato con la chiave pubblica di Bob in cui indica la propria identità, il rendezvous point da lei scelto, il rendevouz cookie e la parte iniziale di un handshake Diffie-Hellman (DH) (5). L'introduction point invia il messaggio a Bob (6).
- Se Bob acconsente a comunicare con Alice, costruisce un circuito col rendezvous point di Alice (7) e invia il rendezvous cookie, la seconda parte dell'handshake DH ed un hash della chiave di sessione che ora i due condividono (8).
- Il rendezvous point collega il circuito di Alice a quello di Bob, ma non può riconoscere Alice, Bob o i dati che essi trasmettono.
- Alice istruisce il circuito affinché l'OP di Bob si connetta al webserver di Bob.
- A questo punto lo stream anonimo è stabilito e Alice e Bob possono comunicare normalmente (9).

L'architettura fin qui descritta è relativa alla prima versione di Tor rilasciata pubblicamente nel Marzo 2004. Nelle sezioni seguenti si farà una panoramica degli attacchi a Tor documentati in letteratura ed ordinati cronologicamente e si indicheranno i provvedimenti presi dal team di sviluppo ed implementati nelle release più recenti di questo software.

B.3 Analisi a basso costo di traffico Tor

I metodi tradizionali di analisi del traffico possono essere applicati contro Tor a diversi livelli di granularità.

La prima classe di attacchi considera la rete anonima come una *black box* e prende in esame solo le tempistiche secondo le quali le connessioni vengono iniziate degli utenti e servite fino alla destinazione esterna alla rete Tor. Questo genere di attacchi è molto potente e capace di rivelare pattern di comunicazione ripetitivi all'interno di Tor. Ad esempio i *disclosure attack* e gli *statistical disclosure attack* sono in grado di indicare sul lungo periodo se un particolare utente si colleghi ogni giorno ad un insieme di siti web attraverso Tor. Tuttavia per portare a termine un attacco di questo tipo, l'attaccante deve essere in grado di osservare un'ampia frazione della rete per poter registrare chi sta accedendo a quali risorse e quali servizi esterni vengono contattati. Un avversario di questo genere ricade al di fuori del modello di minaccia contro cui Tor ambisce a proteggere.

Una seconda famiglia di attacchi agisce ad una granularità più fine, ispeziona il traffico dall'interno della rete di comunicazione anonima e registra la forma corrente del traffico (ovvero l'andamento del carico) su ogni collegamento della rete. Dal momento che i pattern complessivi di traffico non vengono distorti da ogni OR che li veicola, un osservatore globale è in grado di correlare tempi e volumi degli stream in ingresso e uscita per riuscire a tracciare il percorso di una connessione stabilita attraverso gli OR. Anche in questo caso le caratteristiche dell'attaccante stanno al di fuori del modello di minaccia considerato nel design di Tor e le risorse richiesta all'avversario sono ancora maggiori di quelle necessarie a portare a termine un *disclosure attack*.

Ciononostante gli attacchi considerati sono robusti: quando l'attaccante non è globale e dispone di una quantità inferiore di dati, magari parziali e ad una risoluzione di dettaglio più bassa, l'attacco richiede più tempo ed un numero di prove maggiore perché possa portare allo stesso grado di certezza sull'identità delle parti che comunicano, ma sul lungo periodo avrà successo. Quindi un attaccante che controlla solo una parte della rete Tor, così da essere incluso nel modello di minaccia assunto, può essere comunque in grado di tracciare alcune comunicazioni a caso, pur considerando che lo sforzo per l'intercettazione e l'analisi è decisamente demotivante.

Steven Murdoch e George Danezis in [88] spiegano che questi attacchi possono essere condotti senza necessariamente disporre di grandi quantità di dati da analizzare. L'osservazione alla base di questo attacco a basso costo di risorse è che tutti gli stream processati e trasportati su un particolare nodo Tor interferiscono l'uno con l'altro, dal momento che consumano risorse condivise di uno stesso calcolatore quali tempo del processore e banda di rete. Un carico maggiore su un nodo Tor, dovuto ad esempio anche solo ad una connessione extra, risulta in un aumento della latenza di tutte le altre connessioni instradate attraverso quel nodo. Un attaccante può sfruttare questo covert channel inoltrando una connessione attraverso uno specifico nodo Tor e misurare la latenza dei messaggi per ricavare una stima del carico di traffico sul nodo, proporzionale ai carichi di tutte le connessioni servite; quindi può confrontare con tecniche convenzionali di analisi del traffico [89] questo dato con un pattern di traffico a lui noto per verificare se esso sia presente nel profilo del carico del nodo e quindi se il nodo lo stia veicolando.

Nella sua versione più potente, che comunque rientra nel modello di minaccia considerato da Tor, l'attacco prevede che l'avversario controlli sia un OR ostile che il server destinazione a cui si connette l'utente che si vuole rintracciare; la figura B.3 mostra questa configurazione.

L'OR ostile crea una connessione che passa attraverso un altro OR, di cui si desidera misurare il carico di traffico e che fa parte del circuito tra utente e server destinazione. Quando l'utente contatta il server destinazione controllato dall'attaccante, il server risponde inviando all'utente attraverso la connessione Tor dati modulati secondo un pattern molto particolare (*probe traffic*), che ad esempio

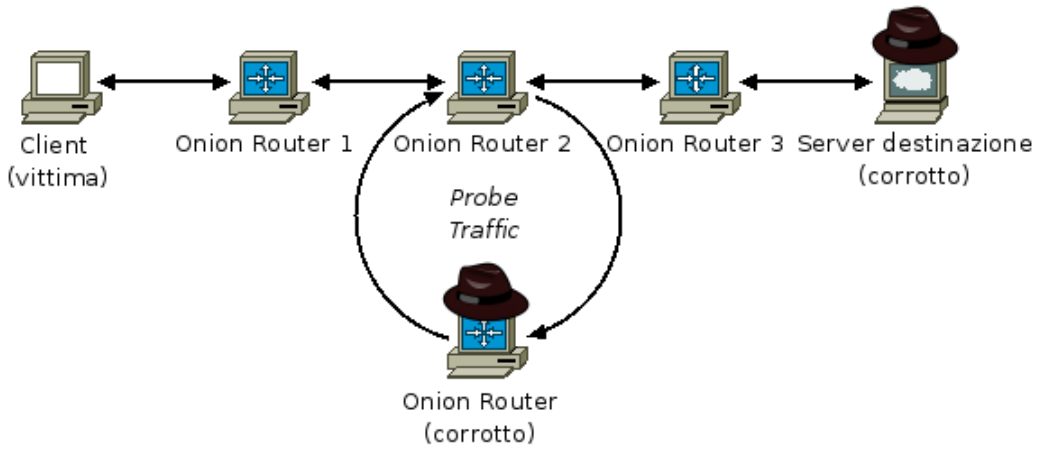

Figura B.3: Attacco con onion router ostile.

consiste in sequenze di brevi burst di dati. Poiché l'attaccante conosce il pattern in input alla rete Tor, può ricostruire un template e usarlo per rilevare se il volume del traffico nel nodo Tor analizzato è correlato con essa.

A partire da questo attacco principale, Murdoch e Danezis propongono una serie di varianti:

- È possibile determinare se due stream in uscita dallo stesso OR appartengono allo stesso utente (*linkability attack*). Questa è una conseguenza del fatto che Tor usa la stessa connesione per instradare più stream provenienti dal medesimo utente: è possibile verificare se due stream che escono dalla rete Tor e accedono a due server ostili appartengono ad un unico utente.

  Per scoprire quali OR trasportano i due stream, si può usare l'attacco principale: si risponde alle richieste dei client con un *probe traffic* diverso per ognuno dei due server ostili e si analizzano gli OR alla ricerca di template compatibili con entrambi i profili di traffico inviati. Mentre la probabilità che due utenti diversi usino lo stesso exit node in una rete di N OR è $1/N$, la probabilità che il percorso completo di tre nodi sia il medesimo, se ogni nodo è scelto casualmente, scende a circa $1/N^3$ ed offre un'indicazione attendibile che i due stream appartengono allo stesso utente.

  Sebbene sia opinione diffusa che un percorso composto da più nodi risulti maggiormente sicuro per l'anonimato, è interessante notare come in questo caso aumentando il numero di OR usati per ogni percorso diminuisce il numero possibile di catene di OR diverse e questo favorisce l'identificazione precisa dei circuiti. Verificare che un secondo stream appartiene allo stesso utente che ha avviato un altro stream già tracciato è più semplice dal momento che l'attaccante conosce già i due OR sul percorso del primo stream e quindi deve solo testarli per confermare che il secondo stream appartiene alla stessa connessione ed al medesimo utente.

- Se il traffico della comunicazione anonima da analizzare proviene soprattutto dall'utente vittima verso il server, il server dell'attaccante non ha molte possibilità di far variare il carico di traffico a suo piacimento.

  Una soluzione può essere quella di modulare il probe traffic inviato al nodo Tor vittima ripetendolo in un ciclo e cercare di rilevare gli effetti nelle richieste inviate dall'utente.

Un problema di questo approccio è che il normale metodo di testare contemporaneamente tutti gli OR nella rete diventa problematico perché la modulazione dello stream vittima è la combinazione del carico indotto su tutti e tre gli OR sul percorso. Un'alternativa è testare ogni OR in sequenza, ma considerato il tempo di vita limitato di uno stream significa ridurre la durata del test e quindi la sua accuratezza. Oppure l'attaccante può testare tutti i nodi, ma usando un pattern differente e "ortogonale" per ogni OR, cosicché la combinazione risultante osservata può essere scomposta nelle componenti originali.

Si può realizzare un attacco adattativo verificando brevemente tutti gli OR della rete, eliminando alcuni dei nodi che non risultano far parte del percorso e ripetendo il test sui nodi rimanenti, possibilmente aumentando la sua durata; il procedimento è ripetuto finché non rimangono solo i tre OR del percorso.

Infine, un'opzione è quella di testare prima una frazione dei nodi; se lo stream risultante risulta affetto allora quella frazione contiene almeno uno dei nodi nel percorso, altrimenti tutti gli OR in quel gruppo possono essere scartati.

- Nel caso l'attaccante non disponga di controllo totale sul server destinazione e possa solo monitorare il suo collegamento senza modificarne il carico, può servirsi delle tecniche esposte al punto precedente oppure avvantaggiarsi della conoscenza di un pattern di traffico osservato sul server, anche se in questo caso l'attacco richiederebbe probabilmente più tempo per raggiungere un risultato, ma il traffico potrebbe comunque essere adatto ad indurre degli effetti osservabili sugli OR intermedi.

  Anche se l'attaccante non è in grado di monitorare il link, può comunque portare a termine l'attacco se il traffico rilevato possiede caratteristiche note visibili sugli OR attraverso i quali transita.

- Un'attaccante che non possa né osservare, né cambiare il traffico sul server di destinazione, può comunque dedurne il carico usando il server e osservando il tempo di risposta. Inoltre egli può modifcare il carico del server destinazione variando l'aggressività di un attacco DoS.

- Gli attacchi precedenti sono diretti a identificare gli OR adoperati per veicolare un particolare stream, ma è possibile anche agire per identificare l'utente che ha avviato lo stream. Un attaccante per raggiungere questo obiettivo deve osservare le connessioni di tutti e tre gli OR sul percorso dello stream. Per semplificare l'attacco si può incentrare il monitoraggio delle connessioni solo sull'entry router, che corrisponde a quello che presenta maggiori distorsioni del *probe traffic* dal momento che è l'OR più distante dal server ostile.

B.4 Localizzazione degli Hidden Server tramite elezione ad ultimo OR

In [90] Lasse Øverlier e Paul Syverson presentano quattro attacchi rapidi ed economici per rivelare l'indirizzo di un Hidden Server (HS). Tutti gli attacchi si basano sull'architettura degli hidden service di Tor discussa nella sezione B.2 e sfruttano la stessa configurazione, illustrata in figura B.4.

L'attaccante controlla un client (OP) e un OR e il suo scopo è posizionarsi come primo nodo del circuito.

In questi attacchi il client e l'OR dell'attaccante sono modifcati rispetto alle versioni ufficiali, nei modi seguenti:

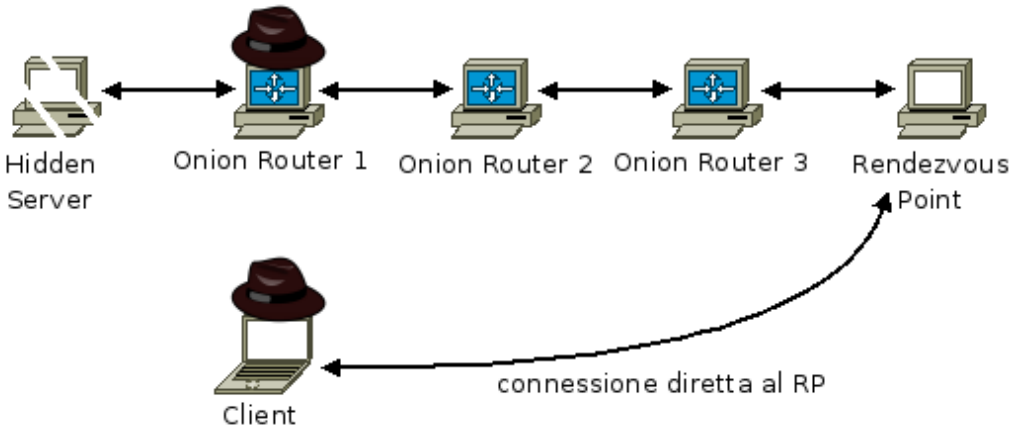

Figura B.4: Configurazione per attacchi all'Hidden Server.

- Il client si connette direttamente, con un solo hop, al RP per ridurre il tragitto e la latenza del traffico tra il client e l'HS, permettendo di rilevare e correlare più facilmente i pattern di traffico.
- Il client chiude il circuito verso l'HS per ogni pattern trasmesso con successo, così da evitare il riutilizzo dei circuiti e forzare la costruzione di un nuovo circuito alla prossima richiesta.
- L'attaccante oltre che impersonare il client esegue anche un OR che partecipa alla rete Tor e trasporta traffico per altri nodi. Egli mantiene una lista dei circuiti attivi e cerca di correlare i dati in transito sul circuito generato dal suo client con tutti gli altri circuiti per capire se sta veicolando gli stessi dati sia come client che come OR.
- L'OR dell'attaccante annuncia ai DS un uptime ed una banda massimi per indurre gli altri nodi a sceglierlo per i loro circuiti (in sezione B.6 si dà giustificazione di questo accorgimento).

Alcuni circuiti della rete Tor mostrano una corrispondenza con un pattern di traffico atteso quando il client invia e riceve dati dall'HS. L'attaccante esamina tutti i circuiti attivi che passano attraverso il suo nodo alla ricerca di quel pattern; se trova una corrispondenza, allora il suo nodo è entrato a far parte del circuito tra HS e *Rendezvous Point* (RP) come primo, secondo o terzo nodo. L'attaccante si accorge quando il suo OR è usato come terzo nodo, il più vicino al RP, dal momento che conosce l'indirizzo IP del RP: in questo caso abbandona il circuito e ripete l'attacco.

Nel caso in cui il suo OR non conosca nessuno dei due indirizzi IP agli estremi del circuito su cui ha rilevato traccia del pattern, l'attaccante sa di essere il primo, direttamente connesso con l'HS, o il secondo nodo nel circuito. Per accertare la propria posizione sfrutta alcuni metodi statistici basati sul conteggio e sui tempi di ricezione e trasmissione delle celle, nonché sulla direzione seguita da ogni singola cella in transito. In questo modo l'attaccante continua a raccogliere campioni di dati fino a che non ne possiede a sufficienza per determinare quando si è connesso all'HS come primo OR nel circuito verso il RP: a quel punto viene a conoscenza dell'indirizzo IP dell'HS.

I quattro attacchi descritti sono i seguenti:

- *Service Location Attack.*
  Quando l'hidden service è ospitato su un client che utilizza la rete Tor, ma non partecipa ad essa come OR, l'attaccante è in grado di localizzare l'HS in pochi minuti. In questo scenario infatti

l'attaccante viene a sapere l'IP dell'HS appena l'analisi statistica segnala una corrispondenza perché un'interrogazione ai DS per quell'IP mostrerà che non è elencato come OR nella directory e quindi è sicuro si tratti dell'HS, altrimenti non avrebbe avuto modo di comunicare con il client dell'attaccante.

- *Predecessor Attack.*
  Nella configurazione di rete considerata, l'attaccante può condurre facilmente un predecessor attack [91] contro Tor.

  L'attaccante mantiene un statistica degli indirizzi IP che contattano il server nel caso sia rilevata una corrispondenza di pattern di traffico. Selezionando solo i circuiti sui quali c'è stata una corrispondenza e usando un percorso di m nodi verso il RP, un singolo indirizzo IP compare in circa $1/m$ di queste connessioni quando HS seleziona il primo nodo. L'indirizzo IP dell'HS è quello che compare più frequentemente ed è semplice individuarlo se m è significativamente più piccolo del numero di nodi in rete. Il tempo richiesto perché questo attacco abbia successo è pari a circa due ore nei test condotti dai suoi autori.

- *Distance Attack.*
  Se non è possibile collezionare informazioni sugli indirizzi IP come richiesto dal punto precedente, l'attaccante può usare un altro metodo per calcolare la sua distanza dall'HS e dedurre se stia agendo come primo OR.

  Se l'attaccante dispone di una descrizione del profilo del traffico generato tra il suo client e l'HS, può analizzare i tempi di risposta di tale comunicazione. L'attaccante misura i periodi dove i dati campionati variano tra fasi di traffico in uscita ed in entrata, ottenendo un'approssimazione del round trip time, ed è così in grado di stimare la distanza dell'HS dal suo client. Raggruppando i nodi in base ai round trip time rilevati, l'attaccante individua alcuni gruppi di nodi statisticamente più vicini all'HS degli altri: se il suo OR è incluso tra questi allora occupa il primo posto nella catena degli OR.

- *Controllo del RP.*
  Se si considera un attaccante con maggiori risorse e che gestisce due OR, è possibile attuare attacchi in cui anche il RP è sotto controllo dell'attaccante.

  Ora l'attaccante conosce l'indirizzo IP del RP e pertanto può capire quando è l'ultimo nodo del circuito verso l'HS (nodo 3 in figura B.4) dal momento che si vede connesso al RP.

  La scelta del RP è effettuata dal client e questo permette all'attaccante di eleggere uno dei suoi due OR come RP, lasciando ancora la possibilità all'HS di usare l'altro suo OR per costruire il circuito verso il RP; grazie a questo l'attaccante capisce quando si trova come secondo nodo nel circuito (nodo 2 in figura B.4) perché sia il client che il RP sono connessi allo stesso nodo (nodo 1 in figura B.4).

  Quando l'attaccante non occupa nessuna delle posizioni precedenti allora è certo di essere collegato direttamente all'HS (nodo 1 in figura B.4) e ne ha scoperto quindi l'indirizzo IP.

  Questo attacco è più rapido ed accurato del Service Location Attack e inoltre funziona sia per identificare hidden service ospitati su OR che su client esterni alla rete Tor.

Tutti gli attacchi visti sfruttano la capacità dell'attaccante di indurre l'HS a creare nuovi circuiti fino a trovarsi connesso direttamente al nodo ostile. In seguito alla pubblicazione del paper di Øverlier e Syverson, in Tor è stato implementato nel Maggio 2006 un sistema di entry guard come contromisura: consultando i DS viene scelto come primo nodo di ogni circuito un router stabile e con molta banda per

agire da entrance router. Questa nuova caratteristica pur risolvendo alcune vulnerabilità ne introduce altre che sono discusse in sezione B.6.

B.5 Localizzazione degli Hidden Service attraverso il loro Clock Skew

Nella sezione B.3 si è visto come su un OR i pattern di traffico in transito su un circuito influiscono in modo osservabile sugli altri circuiti veicolati da quel nodo Tor. Una soluzione proposta per questa vulnerabilità è che Tor offra una QoS fissa per ogni connessione indipendentemente dal suo traffico (strategia *non-interference*), diminuendo potenzialmente le prestazioni, ma rendendo il sistema resistente agli *interference attack* considerati. Murdoch descrive un attacco in grado di aggirare anche questa contromisura, di scoprire l'indirizzo IP del calcolatore che ospita un hidden service e addirittura di localizzarlo geograficamente [92].

Nonostante si adotti una QoS che eviti ai flussi in transito su un OR di interferire l'un l'altro, si osserva comunque un fenomeno inevitabile: quando un circuito sul nodo sta trasportando meno dati della capacità ad esso riservata, il carico sulla CPU del calcolatore che ospita il nodo si riduce. La riduzione del carico sul processore induce una diminuzione della temperatura che a sua volta provoca un *clock skew*, ovvero uno spostamento della frequenza attuale dell'oscillatore a cristallo che guida il clock del sistema rispetto a quella nominale. Il clock skew presenta variazioni nel tempo minime per uno stesso calcolatore, ma per calcolatori diversi si registrano differenze significative di clock skew.

Il clock skew di un calcolatore in rete si può stimare a partire da alcune sorgenti di timestamp, ad esempio i messaggi ICMP Timestamp Request, le TCP Timestamp Option e i numeri di sequenza TCP: l'attacco proposto sfrutta le TCP Timestamp Option perché sono abilitate di default sulla maggioranza dei moderni sistemi operativi e, a differenza dei messaggi ICMP, non vengono di solito bloccate dai firewall perché sono utili a raffinare le stime dei round trip time e per proteggere dal riuso dei sequence number su reti veloci.

L'attacco si basa sull'osservazione che un hidden server esibisce una differenza misurabile di clock skew quando una particolare connessione è attiva rispetto a quando è inattivo. Quando la connessione non è attiva il server ha un carico computazionale minore e quindi il processore si raffredda. La variazione di clock skew indotta dal cambio di temperatura viene rilevata dall'attaccante remoto attraverso richieste di timestamp.

L'attaccante non ha bisogno di controllare un OR: usa un client per accedere all'hidden service e varia nel tempo il traffico inviato così da provocare un riscaldamento o un raffreddamento del server su cui l'hidden service è in esecuzione. Simultaneamente interroga tutti i server che sospetta possano ospitare l'hidden service ricevendo in risposta i loro timestamp: dall'analisi di questi timestamp ottiene una stima dei clock skew e quando individua una correlazione tra lo skew di un calcolatore e il pattern di traffico indotto allora ha scoperto l'identità dell'hidden server.

È interessante notare come l'atto di provocare un clock skew e misurarlo da remoto si possa vedere come un *thermal covert channel*: attaccare un hidden server può essere modellato come una violazione delle politiche di controllo di un flusso di informazione in un sistema distribuito.

Il client dell'attaccante che accede all'hidden service attraverso la rete anonima sfrutta l'associazione tra lo pseudonimo del server e il suo indirizzo IP pubblico, che rappresenta un'informazione ad un livello di confidenzialità "alto". Tuttavia al client è impedito diffondere questa informazione dalla *trusted computing base* della rete anonima.

L'utente che accede all'hidden server direttamente dispone solo di una informazione "bassa", cioè il solo indirizzo IP reale, ma se il processo "alto" fa trapelare l'informazione al processo "basso", allora l'anonimato del server è violato.

La particolarità dell'attacco non risiede tuttavia nella sola caratteristica appena evidenziata. L'attacco consente di andare oltre la mera scoperta dell'indirizzo IP dell'hidden server: è possibile ottenere un'informazione sulla posizione geografica del server. L'attaccante anziché misurare il clock skew contattando il server al di fuori della rete anonima può suscitare le risposte contenenti i timestamp passando direttamente dalla rete Tor e valutando le macrovariazioni dello skew nel tempo indotte dall'ambiente può dedurre il luogo geografico dove il server risiede fisicamente.

La conoscenza del clock skew non fornisce informazioni sulla temperatura assoluta, ma solo sulle sue fluttuazioni. La longitudine può essere individuata trovando il picco di temperatura giornaliero per stabilire quale sia l'ora locale. Per scoprire la latitudine si può studiare il cambiamento della durata del giorno su un periodo di tempo sufficientemente lungo.

Queste tecniche si rendono particolarmente utili quando non sia possibile dedurre la località geografica dal solo indirizzo IP, come nel caso di IP anycast, connessioni satellitari o connessioni dial-up a lunga distanza.

B.6 Attacchi con limitate risorse al meccanismo di routing preferenziale

Al termine della sezione B.4 è stato citato il sistema di *entry guard* introdotto per proteggere il nodo iniziale dei circuiti e che funziona eleggendo come *entrance router* uno tra i nodi più affidabili e fidati. L'algoritmo (*entrance router selection algorithm*) opera scegliendo automaticamente un insieme di OR segnalati dai directory server come "veloci" e "stabili".

Per i directory server un router veloce è un router che dichiara una banda al di sopra della media di tutte le bande annunciate. Un router stabile è definito come uno che annuncia un uptime maggiore della media di quelli di tutti gli altri router.

Un client Tor per selezionare gli OR da utilizzare nel circuito applica l'algoritmo di *entry guard* per scegliere l'*entrance node*, quindi impiega un altro algoritmo (*non-entrance router selection algorithm*) per eleggere il secondo e terzo OR cercando di usare router con buona banda e elevato uptime, ma senza privilegiare sempre i router migliori secondo questi parametri. Per ottimizzare le prestazioni questo secondo algoritmo fa sì che tutti i nodi della rete Tor vengano usati per trasportare il traffico, ma i nodi con maggiori risorse di banda e molto stabili vengano adoperati più spesso.

Kevin Bauer, Damon McCoy ed altri sfruttano la tendenza del meccanismo di routing di Tor a preferire nodi stabili e con molta banda per implementare un attacco alla rete anonima [93]. L'attaccante deve controllare un sottoinsieme di $m > 1$ nodi tra gli OR attivi, gestendo propri OR malevoli o compromettendo alcuni OR esistenti ed inizialmente onesti.

In una prima fase i nodi dell'attaccante annunciano ai directory server di non imporre restrizioni sulle proprie exit policy, cioè di essere pronti a veicolare traffico diretto a qualsiasi indirizzo IP e porta TCP, e di disporre di molta banda ed un elevato uptime, anche se questo non è vero ed essi in realtà sono nodi di limitate risorse. L'attaccante in questo modo dà l'impressione che i suoi OR costituiscano un'ottima scelta e aumenta le probabilità che i client Tor scelgano loro come entrance router ed exit router nella costruzione di nuovi circuiti, condizione necessaria per procedere alla seconda parte dell'attacco.

Nella seconda fase, l'attaccante vuole violare la privacy dell'utente correlando le richieste del client connesso all'entrance router ostile alle risposte del server connesso all'exit router ostile. Ogni OR dell'attaccante memorizza le seguenti informazioni per ogni cella ricevuta:

- Posizione occupata dal router nel circuito corrente (*entrance*, *mix* o *exit router*).
- Timestamp attuale.

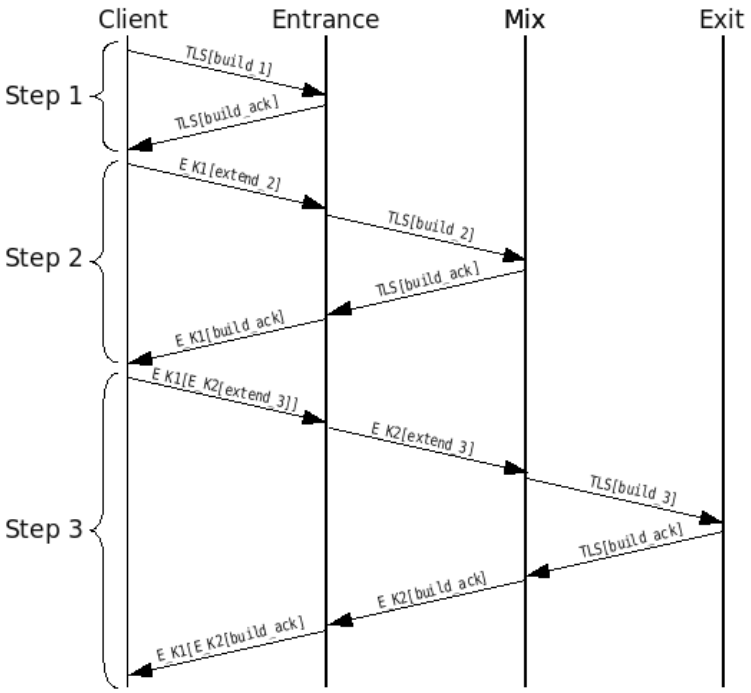

Figura B.5: Costruzione di un circuito Tor.

- ID del circuito dell'hop precedente.
- Indirizzo IP dell'hop precedente.
- Porta di connessione dell'hop precedente.
- Indirizzo IP del prossimo hop.
- Porta di connessione del prossimo hop.
- ID del circuito del prossimo hop.

Gli OR ostili si coordinano attraverso un'autorità centralizzata che raccoglie le informazioni sulle celle e le utilizza per eseguire l'algoritmo di *path linking*. Questo algoritmo determina quali percorsi che includono un OR ostile in posizione di ingresso ed uscita della rete veicolano le richieste di costruzione di un particolare circuito provenienti dall'OP dell'utente: in questo modo le identità del mittente e del destinatario sono associate e l'anonimato del sistema è compromesso. L'algoritmo di *path linking* sfrutta la peculiarità del motodo di costruzione dei circuiti Tor: la sequenza di pacchetti scambiati in questa fase, riportata in B.5, costituisce un pattern preciso e facilmente riconoscibile.

Il diagramma mostra le tre fasi della sequenza: nella prima il client sceglie l'*entrance router* del circuito, nella seconda l'entrance router inoltra la richiesta del client al *mix router* selezionato e nella terza l'entrance e il mix router spediscono la richiesta finale di costruzione del circuito all'*exit node* desiderato.

Per sfruttare questo algoritmo di costruzione del circuito gli OR dell'attaccante associano l'informazione temporale di ogni passo all'analisi del numero e della direzione delle celle memorizzate.

L'entrance OR ostile verifica che la richiesta di creazione del circuito sia originata da un OP e non da un OR: per farlo controlla semplicemente che il nodo non sia elencato come OR sui directory server.

In seguito l'algoritmo controlla che i tre passi della sequenza di pacchetti avvengano in corretta successione cronologica e che il prossimo hop visto dall'entrance router ostile sia lo stesso visto come hop precedente dall'exit router ostile.

Infine, durante il terzo ed ultimo passo, verifica che la cella indirizzata verso l'exit node dall'entrance node sia ricevuta prima della risposta dall'exit node.

Se tutti i passi dell'algoritmo sono soddisfatti, allora il circuito è stato compromesso.

Nei test di laboratorio condotti dagli autori dell'attacco, quando l'attaccante controlla circa il 9% degli OR di una rete di 60 nodi riesce a compromettere quasi la metà di tutto il traffico Tor e l'attacco si suppone scali $O(N)$ per una rete di dimensione N.

La particolarità di questo attacco consiste nella capacità del *path linking algorithm* di violare l'anonimato dei flussi prima della trasmissione di qualsiasi payload utile sulla connessione. Questo offre un vantaggio notevole per l'attaccante: egli può accorgersi di non essere riuscito a violare l'anonimato della comunicazione e dal momento che controlla almeno un OR sul percorso tra mittente e destinazione può terminare il circuito prima che sia completamente costruito. Poiché non è ancora riuscito a trasferire alcun dato, il client cercerà di costruire un nuovo circuito e offrirà quindi all'attaccante una nuova opportunità di scoprirne l'identità.

A partire dall'attacco base, si possono considerare alcune sue varianti:

- *Compromissione di client preesistenti.*
  I client che partecipano alla rete prima che gli OR ostili ne entrino a far parte hanno già scelto il loro insieme di nodi entry guard. L'attaccante può agire almeno in due modi per compromettere l'anonimato dei client preesistenti.

  Per prima cosa, se è in grado di osservare il traffico del client, può dedurre quali siano le entry guard usate dal client, lanciare contro di esse un DoS e renderle così inutilizzabili. In questo modo il client deve forzatamente scegliere una nuova lista di entry guard, nella quale potrebbero esserci gli OR ostili.

  Un altro metodo è quello di effettuare un DoS contro alcuni nodi chiave stabili che pertanto fungono da entry guard per molti client per sostituire i propri OR ad essi.

- *Miglioramento delle performance in caso di risorse limitate.*
  Nello schema base gli OR dell'attaccante attirano molto traffico e se dispongono di risorse limitate il carico eccessivo potrebbe degradare le prestazioni dell'intera rete Tor e quindi far insospettire gli utenti.

  Per migliorare le performance avvertite dagli utenti l'attaccante, se vuole colpire un particolare client, può evitare di servire le richieste di creazione del circuito di tutti gli utenti tranne quelle provenienti dall'obiettivo. Quest'azione fa sì che i client non serviti escludano gli OR ostili dai propri circuiti, considerandoli malfunzionanti: poiché il fallimento di un nodo è un evento comune nella rete Tor, è probabile che l'anomalia non venga rilevata come malevola.

- *Distruzione selettiva dei circuiti.*
  Se un OR ostile non viene scelto né come entrance né come exit router, può rompere il circuito bloccando tutto il traffico in transito su di esso. Il circuito verrà ricreato dando una nuova possibilità agli OR dell'attaccante di occupare entrambe le posizioni di interesse.

- *Sostituzione di tutte le entry guard oneste.*
  In una variante del *Sybil attack*, l'attaccante inonda la rete Tor con un numero di OR ostili che annunciano un alto uptime e ampia banda tale da incrementare in modo consistente la media di questi parametri.

  Siccome le entry guard sono router che possiedono caratteristiche al di sopra della media, se questa sale sufficientemente si può impedire a tutti i router onesti di essere scelti come entry guard, a favore degli OR ostili che andranno a prenderne il posto.

- *Compromissione solo dell'entry node.*
  In uno scenario dove l'attaccante vuole de-anonimizzare le richieste di un client Tor verso uno specifico server ed è in grado di monitorare il collegamento di rete del server contattato, è sufficiente che egli comprometta il solo entry router. L'attacco è portato a termine utilizzando una tecnica di *watermarking* [94] dei pacchetti del client inviati attraverso l'*entrance router* ostile.

Gli attacchi descritti in questa sezione possono essere mitigati grazie all'adozione di un sistema di reputazione distribuita che verifichi tutte le informazioni collezionate che possono influenzare le decisioni sul routing attraverso osservazioni locali effettuate dai client e tenute in considerazione nel processo di selezione dei nodi per i circuiti. Inoltre è ragionevole sostenere che un vincolo sulla diversità spaziale costituisca un ostacolo contro la realizzazione di una rete di OR ostili da parte dell'attaccante.

L'attuale versione stabile di Tor (0.1.1.26) non corregge queste vulnerabilità, tuttavia il team del progetto sostiene che questa tipologia di attacchi provocherebbe un vistoso "inquinamento" della base dati dei directory server e sarebbe pertanto facile da rivelare ed arginare. Finora non si ha segnalazione di attacchi "in the wild" che abbiano sfruttato questa tecnica.

Bibliografia

[1] Ragib Hasan, Zahid Anwar, William Yurcik, Larry Brumbaugh, Roy Campbell. A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems. Proceedings of the International Conference on Information Technology: Coding and Computing, ITCC'05(2):44-54, 2005. <http://www.projects.ncassr.org/storage-sec/papers/itcc05.pdf>

[2] Atul Singh, Tsuen-Wan Ngan, Peter Druschel, Dan S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. Proceedings of INFOCOM, April 2006. <http://www.eecs.harvard.edu/~mema/courses/cs264/papers/eclipse-infocom06.pdf>

[3] B.Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition. Wiley Publishing, 1996.

[4] N.Ferguson, B.Schneier. Practical Cryptography. Wiley Publishing, 2003.

[5] D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, J. Kubiatowicz. OceanStore: An Extremely Wide-Area Storage System. Technical Report UCB/CSD-00-1102. 1999. <http://oceanstore.cs.berkeley.edu/publications/papers/pdf/oceanstore-tr-may99.pdf>

[6] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. University of California, Berkeley, 2000. <http://oceanstore.cs.berkeley.edu/publications/papers/pdf/asplos00.pdf>

[7] PlanetLab, <http://www.planet-lab.org/>

[8] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, J. Kubiatowicz. Pond: the OceanStore prototype. In USENIX FAST, Mar. 2003. <http://srhea.net/papers/fast2003-pond.pdf>

[9] Chimera e Tapestry, <http://p2p.cs.ucsb.edu/chimera/>, <http://current.cs.ucsb.edu/projects/chimera/>

[10] OceanStore. <http://oceanstore.cs.berkeley.edu/>, <http://oceanstore.sf.net/>

[11] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, J. D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. IEEE Journal on selected areas in communications, VOL. 22, NO. 1, January 2004. <http://pdos.csail.mit.edu/~strib/docs/tapestry/tapestry_jsac03.pdf>

[12] Y. Zhao, J. D. Kubiatowicz, A. D. Joseph, I. Stoica, J. Chuan. Decentralized Object Location and Routing: A New Networking Paradigm Dissertation, University of California, Berkeley, 2004. <http://oceanstore.cs.berkeley.edu/publications/papers/pdf/zhao.pdf>

[13] B. Bloom. Space/time trade-offs in hash coding with allowable errors. In Communications of the ACM, volume 13(7). July 1970. <http://gnunet.org/papers/p422-bloom.pdf>

[14] J. Douceur. The Sybil attack. In IPTPS, 2002. <http://www.cs.rice.edu/Conferences/IPTPS02/101.pdf>

[15] OpenDHT, <http://www.opendht.org/>

[16] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, H. Yu. OpenDHT: A Public DHT Service and Its Uses. UC Berkeley and Intel Research, 2004. <http://berkeley.intel-research.net/sylvia/f230-rhea.pdf>

[17] Bamboo, <http://bamboo-dht.org/>

[18] S. Rhea, D. Geels, T. Roscoe, J. Kubiatowicz. Handling churn in a DHT. In USENIX Annual Tech. Conf., June 2004. <http://srhea.net/papers/bamboo-usenix.pdf>

[19] M. Zitterbart, M. Dischinger. A flexible and scalable peer-to-peer multicast application using Bamboo, University of Karlsruhe (TH), 2004. <http://www.cl.cam.ac.uk/research/srg/netos/futuregrid/dischinger-report.pdf>

[20] P. Eaton, H. Weatherspoon, J. Kubiatowicz. Efficiently Binding Data to Owners in Distributed Content-Addressable Storage Systems. University of California, Berkeley, Security in Storage Workshop, December 2005. <http://www.cs.berkeley.edu/~eaton/pubs/talks/syslunch_2005_11_21.pdf>

[21] D. Xiaodong Song, D. Wagner, A. Perrig. Practical Techniques for Searches on Encrypted Data. University of California, Berkeley, 2000. <http://www.ece.cmu.edu/~dawnsong/papers/se.pdf>

[22] Xerox PARC's Bayou Project, <http://www2.parc.com/csl/projects/bayou/>

[23] Mnemosyne, <http://www.cl.cam.ac.uk/~smh22/>

[24] S. Hand, T. Roscoe. Mnemosyne: Peer-to-Peer Steganographic Storage. Sprint Advanced Technology Lab, 1 Adrian Court, Burlingame, CA 94010, USA, 2002. <http://www.cl.cam.ac.uk/Research/SRG/netos/papers/2002-mnemosyne-iptps.pdf>

[25] T. Roscoe and S. Hand. Transaction-based Charging in Mnemosyne: a Peer-to-Peer Steganographic Storage System. Sprint Advanced Technology Laboratory, Burlingame, CA 94010, USA, University of Cambridge Computer Laboratory, Cambridge, CB3 0FD, UK, 2002. <http://www.cl.cam.ac.uk/Research/SRG/netos/papers/2002-mnemosyne-pisa.pdf>

[26] Steganography. <http://en.wikipedia.org/wiki/Steganography>

[27] StegFS, a steganographic filesystem <http://stegfs.sourceforge.net/> <http://www.mcdonald.org.uk/StegFS/>

[28] Il progetto Free Haven, <http://freehaven.net>

[29] M. Janczyk. The Free Haven Project. Distributed Anonymous Storage Service. University of Freiburg, 2007. <http://cone.informatik.uni-freiburg.de/teaching/seminar/p2p-networks-w06/submissions/FreeHaven.pdf>

[30] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. 1989 <http://discovery.csc.ncsu.edu/~aliu3/reading_group/p335-rabin.pdf>

[31] R. Dingledine, N. Mathewson, P. Syverson. Reputation in P2P Anonymity Systems. 2003. <http://www.sims.berkeley.edu/research/conferences/p2pecon/papers/s2-dingledine.pdf>

[32] R. Dingledine, M. J. Freedman, D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. 2000. <http://freehaven.net/doc/berk/freehaven-berk.ps>

[33] P. Resnick, R. Zeckhauser, E. Friedman, K. Kuwabara. Reputation Systems: Facilitating Trust in Internet Interactions. Communications of the ACM, 43(12), December 2000. <http://www.si.umich.edu/~presnick/papers/cacm00/reputations.pdf>

[34] Freenet public repository. <http://freenet.googlecode.com/svn/trunk/>

[35] Algoritmo Hill Climbing. <http://en.wikipedia.org/wiki/Hill_climbing>

[36] M. K. Reiter, A. D. Rubin. Anonymous web transactions with Crowds. Communications of the ACM 42(2), 32-38. 1999. <http://gecko.cs.purdue.edu/gnet/papers/p32-reiter.pdf>

[37] David L. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM 24(2), 84-88. 1981. <http://freehaven.net/anonbib/cache/chaum-mix.pdf>

[38] Rachna Dhamija. A Security Analisys of Freenet. 2000. <http://www.ischool.berkeley.edu/~rachna/courses/cs261/paper.html>

[39] S. Garfinkel. Pretty Good Privacy. O'Reilly & Associates inc. 1995.

[40] Ian Clarke. Freenet's Next Generation Routing Protocol. 2003. <http://freenetproject.org/ngrouting.html>

[41] Lance Cottrell. Mixmaster & Remailer Attacks. <http://web.inf.tu-dresden.de/~hf2/anon/mixmaster/remailer-essay.html>

[42] C. Agosti, S. Zanero. Sabbia: a low-latency design for anonymous networks. January 2005. <http://www.s0ftpj.org/docs/sabbia-pet2005.pdf>

[43] Mixminion: A Type III Anonymous Remailer. <http://www.mixminion.net/>

[44] Premix routing. <http://wiki.freenetproject.org/PremixRouting>

[45] PreTunneling. <http://wiki.freenetproject.org/PreTunneling>

[46] Freedom. <www.zeroknowledge.com>

[47] I2P anonymizing network <http://www.i2p.net/>

[48] Routing table takeover attacks. <http://wiki.freenetproject.org/RoutingTableTakeover>

[49] Peter Biddle, Paul England, Marcus Peinado, Bryan Willman. The Darknet and the Future of Content Distribution. 2001. <http://msl1.mit.edu/ESD10/docs/darknet5.pdf>

[50] Bryan Ford, Pyda Srisuresh. Peer-to-Peer communication across network address translators. Proceedings of the 2005 USENIX Technical Conference, 2005. <http://pdos.csail.mit.edu/papers/p2pnat.pdf>

[51] Station-to-Station protocol. <http://en.wikipedia.org/wiki/Station-to-Station_protocol>

[52] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, O. Reingold. Just fast keying: Key agreement in a hostile internet. ACM Trans. Inf. Syst. Secur., 7(2):242–273, 2004. <http://theory.csail.mit.edu/~canetti/materials/jfk.pdf>

[53] Stanley Milgram. The Small World problem. Psychology Today, 1:61, 1961.

[54] Aaron Clauset, Cristopher Moore. How do networks become navigable?. Preprint, 2003. <http://www.research.ibm.com/nips03workshop/Presentations/Clauset.pdf>

[55] Oskar Sandberg. Distributed Routing in Small-World Networks. 2005. <http://www.math.chalmers.se/∼ossa/swroute.pdf>

[56] Oskar Sandberg. Searching in a Small World. 2005. <http://www.math.chalmers.se/∼ossa/lic.pdf>

[57] Progetto Winston Smith. <http://www.winstonsmith.info/pws/>

[58] Convegno E-Privacy. <http://e-privacy.winstonsmith.info/>

[59] Garrett Hardin. The Tragedy of the Commons. Science 162, pp. 1243-1248. 1968.

[60] Adam Back. Hashcash - A Denial of Service Counter-Measure. technical report, 1 Agosto 2002. <http://www.hashcash.org/papers/hashcash.pdf>

[61] Hashcash. <http://www.hashcash.org/>

[62] A. Juels, J. Brainard. Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks, NDSS '99.

[63] Ronald L. Rivest, Adi Shamir, David A. Wagner. Time-Lock Puzzles and Timed-Release Crypto. 1996. <http://people.csail.mit.edu/rivest/RivestShamirWagner-timelock.ps>

[64] David Hopwood. Recipient-Hiding Blinded Public-Key Encryption. unfinished draft, <http://www.users.zetnet.co.uk/hopwood/crypto/rh/>

[65] Adi Shamir. How to share a secret. Communications of the ACM, 22(1), pp. 612-613, 1979. <http://www.cs.tau.ac.il/∼bchor/Shamir.html>

[66] Tal Malin. Private Information Retrieval and Oblivious Transfer. MIT Ph.D. thesis, 1999.

[67] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. Proceeding of USENIX Operating System Design and Implementation (OSDI), Boston, MA, Dec. 2002. <http://www.cs.rice.edu/∼dwallach/pub/osdi2002.pdf>

[68] M. Castro, P. Druschel, Y. C. Hu, A. Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical Report MSR-TR-2003-52, Microsft Research, June 2003. <http://research.microsoft.com/∼antr/PAST/location-msrtr-2003-52.pdf>

[69] K. P. Gummadi, S. Saroiu, S. D. Gribble, S. Ratnasamy, S. Shenker, I. Stoica. The impact of DHT routing geometry on resilience and proximity. Proceedings of ACM SIGCOMM, Karlsruhe, Germany, Aug. 2003. <http://www.mpi-sws.mpg.de/∼gummadi/papers/p1101-gummadi.pdf>

[70] K. Hildrum, J. Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. Proceeding of 17th International Symposium on Distributed Computing, Sorrento, Italy, Oct. 2003. <http://iris.lcs.mit.edu/irisbib/papers/asymp:isdc/paper.pdf>

[71] Atul Singh, Tsuen-Wan Ngan, Peter Druschel, Dan S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. Proceedings of INFOCOM 2006. 25th IEEE International Conference on Computer Communications, Barcelona, Spain, April 2006. <http://www.cs.rice.edu/∼druschel/publications/Eclipse-Infocom.pdf>

[72] Atul Singh, Miguel Castro, Antony Rowstron, Peter Druschel. Defending against Eclipse attacks on overlay networks. Proceedings of the 11th ACM SIGOPS European Workshop, Leuven, Belgium, September 2004. <http://project-iris.net/irisbib/papers/eclipseattack:sigops04/paper.pdf>

[73] L. Lamport, R. Shostak, M. Pease. The Byzantine generals problem. ACM Trans. Program. Lang. Syst. 4, 3 (July 1982), 382-401. <http://research.microsoft.com/users/lamport/pubs/byz.pdf>

[74] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, J. Kubiatowicz. Maintenance'free global data storage. IERRR Internet Computing 5, 5 (Sept-Oct. 2001). <http://srhea.net/papers/ieeeic.pdf>

[75] H. Weatherspoon, J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. Proceedings of First International Workshop on Peer-to-Peer Systems (IPTPS 2002). <http://www.cs.rice.edu/Conferences/IPTPS02/170.pdf>

[76] Roger Dingledine, Michael J. Freedman, David Molnar. "Free Haven, in Andy Oram (editor), Peer-to-Peer: Harnessing the Power of Disruptive Technologies, O'Reilly, Nov 2000. <http://freehaven.net/doc/oreilly/freehaven-ch12.html>

[77] Roger Dingledine, Nick Mathewson. Anonymity Loves Company: Usability and the Network Effect. Workshop on the Economics of Information Security, June 2006. <http://freehaven.net/doc/wupss04/usability.pdf>

[78] John Kubiatowicz. Extracting Guarantees from Chaos. Communications of the ACM Volume 46, Issue 2 pp. 33-38. February 2003. <http://oceanstore.cs.berkeley.edu/publications/papers/pdf/CACM-kubiatowicz.pdf>

[79] G. Danezis, R. Dingledine, N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol, 2006. <http://mixminion.net/minion-design.pdf>

[80] R. Dingledine. Mixminion: Design of a Type III Anonymous Remailer Protocol. DefCON, 2002. <http://freehaven.net/doc/defcon02/slides-dc02.pdf>

[81] Anonymous Remailer FAQ. <http://www.andrebacard.com/remail.html>

[82] R. Dingledine, L. Sassaman. Attacks on Anonymity Systems: The Teory. 2003. <http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-sassaman-dingledine/bh-us-03-sassaman.pdf>

[83] G. Danezis, R. Dingledine, N. Mathewson. Mixminion specifications, last revision 2007. <http://mixminion.net/minion-spec.txt> <http://mixminion.net/dir-spec.txt>

[84] Paul F. Syverson, David M. Goldschlag, Michael G. Ree. Anonymous Connections and Onion Routing. Proceedings of the 18th Annual Symposium on Security and Privacy, IEEE CS Press, Oakland, CA, pp. 44-54, May 1997. <http://www.itd.nrl.navy.mil/ITD/5540/personnel/goldschlag/publications/oakland97.ps>

[85] W. Diffe, M. E. Hellman. New Directions in cryptography, IEEE Transactions on Information Theory, IT-22(6):644-654, 1976. <http://crypto.csail.mit.edu/classes/6.857/papers/diffie-hellman.pdf>

[86] T. Dierks, C. Allen. RFC 4346: The Transport Layer Security (TLS) Protocolo Version 1.1, Aprile 2006. <http://www.ietf.org/rfc/rfc4346.txt?number=4346>

[87] R. Dingledine, N. Mathewson. Tor Protocol Specification. 20 Luglio 2006, <http://tor.eff.org/cvs/doc/tor-spec.txt>

[88] Steven J. Murdoch, George Danezis. Low-Cost Traffic Analysis of Tor. IEEE Symposium on Security and Privacy 2005: 183-195. <http://www.cl.cam.ac.uk/∼sjm217/papers/oakland05torta.pdf>

[89] G. Danezis. The traffic analysis of continuous-time mixes. Proceedings of Privacy Enhancing Technologies workshop (PET 2005), vol 3434 of LNCS, May 2004. <http://homes.esat.kuleuven.be/∼gdanezis/cmm2.pdf>

[90] Lasse Øverlier, Paul Syverson. Locating Hidden Servers. Proceedings of the 2006 IEEE Symposium on Security and Privacy 00, 100-114, 2006. <http://freehaven.net/anonbib/cache/hs-attack06.pdf>

[91] M. K. Wright, M. Adler, B. N. Levine, C. Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. ACM Trans. Inf. Syst. Secur. 7(4), 489-522, 2004. <http://ranger.uta.edu/∼mwright/papers/wright-tissec.pdf>

[92] Steven J. Murdoch. Hot or Not: Revealing Hidden Services by their Clock Skew. Proceedings of the 13th ACM conference on Computer and communications security, 27-36, 2006. <http://www.cl.cam.ac.uk/∼sjm217/papers/ccs06hotornot.pdf>

[93] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, Doublas Sicker. Low-Resource Routing Attacks Against Anonymous Systems. University of Colorado at Boulder Techinal Report, CU-CS-1025-07, 2007. <http://www.cs.colorado.edu/department/publications/reports/docs/CU-CS-1025-07.pdf>

[94] X. Wang, S. Chien, S. Jajodia. Tracking anonymous peer-to-peer voip calls on the internet. Proceedings of the ACM Conference on Computer and Commnications Security, 81-91, Novembre 2005. <http://ise.gmu.edu/∼xwangc/Publications/CCS05-VoIPTracking.pdf>