Alma Mater Studiorum · Università di Bologna

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

CONFRONTO TRA MECCANISMI DI ANONIMIZZAZIONE PER IL CLOUD PEER-TO-PEER

Tesi di Laurea in Sicurezza delle Reti

Relatore:
GABRIELE D'ANGELO

Presentata da:
NICHOLAS BRASINI

Sessione II
Anno Accademico 2018 - 2019

Sommario

Lo scopo principale di questa tesi è quello di implementare la rete di anonimizzazione Invisible Internet Project (I2P) all'interno di un prototipo esistente basato su un'architettura cloud peer-to-peer (P2P). Tale prototipo implementa al proprio interno la rete di anonimizzazione The Onion Router (Tor) e il secondo obiettivo di questa tesi è quello di analizzare e valutare le performance offerte da queste due tipologie di reti. Il prototipo, basandosi su un sistema decentralizzato P2P che, rispetto a quello centralizzato non ha il problema di rappresentare un *single point of failure*, permette la scoperta di peer all'interno della rete mediante algoritmi di gossip, abili a riconoscere i *churn*, ovvero quei nodi appartenenti alla rete che si sono disconnessi non facendone più parte. In questo modo ogni peer sarà aggiornato sullo stato dei peer conosciuti e riuscirà ad eliminare dal proprio elenco i peer disconnessi dalla rete evitando di contattarli inutilmente. Oltre alla funzionalità di ricerca di altri peer all'interno della rete, il prototipo mette a disposizione algoritmi e script utili per poter monitorare la rete stessa ed il suo comportamento e per poter costruire *overlay* tra alcuni dei peer appartenenti alla rete.

Introduzione

Uno dei più grandi punti interrogativi del mondo dell'informatica è sempre stato quello relativo alla *cybersecurity*. Da un lato si trova infatti la fazione di chi la ritiene fondamentale, mentre dall'altro quella di coloro che non se ne curano minimamente ma, anzi, la considerano quasi esclusivamente un passatempo per appassionati del settore. Nel corso degli ultimi anni però il mondo del lavoro nelle grandi e piccole aziende ha trovato un'inaspettata sinergia con quello dell'informatica, dimostrando come uno dei settori più floridi e in espansione sia proprio quello della *cybersecurity*. Analizzando più nello specifico la questione, uno dei macrosettori relativi a questo ambito è quello del cloud computing che, volenti o nolenti, influenza quotidianamente le nostre vite. Da quando abbiamo iniziato ad utilizzare uno smartphone connesso ad Internet, oppure un tablet o un computer, siamo consapevolmente o meno entrati in questo vasto mondo dai contorni non del tutto definiti. Le chat di WhatsApp e Telegram vengono salvate nel cloud, così come i backup dei dati personali e tutta la serie di informazioni presente nei nostri device. La maggior parte dei Cloud Service Provider promette di garantire l'accesso esclusivamente al proprietario dei dati, ma sarà davvero così? I mondi della *cybersecurity* e del cloud computing dunque sono in continua evoluzione, con la lotta per poter garantire l'anonimato che sta entrando nel vivo proprio nel corso di questi anni. Dal momento in cui il cloud continua a crescere e a diffondersi, qualunque utilizzatore di tale sistema è alla ricerca di anonimato per poter proteggere i propri dati, mentre l'altra faccia della medaglia prevede che ci siano sempre più malintenzionati alla ricerca di scavalcare tali barriere. Essendo stato attratto quasi magneticamente da questo settore, ho deciso di scrivere la mia tesi di Laurea magistrale concentrandomi proprio sulle reti di anonimizzazione all'interno di un'infrastruttura cloud peer-to-peer (P2P). Sono partito da un software prototipale già in grado di poter creare l'infrastruttura P2P [1] e di rendere anonimi gli utenti e i peer attraverso la famosa rete di anonimizzazione Tor [2] [3]. Successivamente ho aggiunto le funzionalità necessarie al prototipo per poter utilizzare un'ulteriore rete di anonimizzazione: la Invisible Internet Project (I2P) [4]. La struttura della tesi sarà organizzata come segue. Inizialmente mostrerò lo stato dell'arte del cloud computing, ponendo l'attenzione sui servizi offerti dal mondo cloud, su quelli messi a disposizione dal prototipo in mio possesso e sull'importanza dell'anonimizzazione

in tale ambito. Successivamente analizzerò il background informatico necessario per poter padroneggiare tutti gli strumenti utili per comprendere a fondo questo lavoro di tesi. A quel punto mi focalizzerò sul mondo dell'I2P, descrivendo nel dettaglio le sue specifiche e come sono riuscito ad aggiungere il supporto per tale rete all'interno del prototipo. Terminati questi passaggi non rimarrà che considerare la valutazione delle performance, analizzando prima quelle della rete e successivamente quelle relative al prototipo finale. In particolare, per quanto riguarda questo ultimo argomento, sarà interessante prendere in considerazione le differenze che possono sorgere tra la versione non anonima e quella anonima, analizzando nel dettaglio le diversità che intercorrono tra Tor e I2P.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Il mondo del cloud computing e l'anonimizzazione delle reti

In questo capitolo l'obiettivo è quello di focalizzare l'attenzione sul cloud computing e sull'anonimizzazione delle reti, i due argomenti che rappresentano le fondamenta di questa tesi di Laurea. Dapprima passerò in rassegna le caratteristiche del cloud computing, ponendo l'attenzione sulle diverse tipologie di modelli e servizi, per poi arrivare al confronto tra cloud classico e cloud P2P. Facendo riferimento alla struttura del prototipo che mi è stato messo a disposizione, introdurrò successivamente le sue principali funzionalità e servizi, soffermandomi poi sulle tematiche legate all'anonimizzazione delle reti. Infine valuterò lo stato dell'arte per quanto riguarda progetti simili a quello di questa tesi, prima di aprire il capitolo successivo all'interno del quale evidenzierò tutte le caratteristiche tecniche di questo progetto.

1.1 Il cloud computing

Come accennato nell'introduzione, l'evoluzione che il cloud computing sta progressivamente avendo nel corso degli ultimi anni è esponenziale. In virtù di questa crescita anche gli utenti si stanno abituando ad utilizzare tutte le risorse e i servizi forniti da questo particolare ramo dell'informatica. Con l'avvento dell'Industria 4.0 [5] ma più in generale con l'utilizzo smodato di device e quindi di dati da elaborare e dover mantenere in memoria, il cloud computing ha assunto un ruolo totalmente centralizzante per quanto riguarda l'informatica. La definizione che propone il National Institute of Standard and Technology (NIST) [6] riguardo a tale argomento è piuttosto esplicativa: *"Il cloud computing è un modello per abilitare, tramite la rete, l'accesso diffuso, agevole e a richiesta, ad un insieme condiviso e configurabile di risorse di elaborazione (ad esempio reti, server, memoria, applicazioni e servizi) che possono essere acquisite e rilasciate rapidamente e con minimo sforzo di gestione o di interazione con il fornitore di servizi"*. Nonostante la sua continua evoluzione, il cloud computing continua a mantenere una struttura abbastanza lineare per quanto riguarda caratteristiche, servizi e modelli che vengono offerti agli utenti, che verrà presentata nelle sezioni seguenti.

1.1.1 Caratteristiche del cloud computing

Il modello cloud proposto dal NIST è basato su cinque caratteristiche essenziali che rappresentano i veri punti di forza di tale tecnologia. Sto parlando di *on-demand self-service, broad network access, resource pooling, rapid elasticity* e *measured service*.

- **On-demand self-service**. Un cliente può acquistare unilateralmente e automaticamente le necessarie capacità di calcolo, come tempo macchina e memoria, senza richiedere alcun tipo di interazione umana con i fornitori dei servizi scelti.
- **Broad network access**. Il modello cloud deve garantire un ampio accesso alla rete. Le capacità sono disponibili in rete e accessibili mediante meccanismi standard che promuovono l'uso attraverso piattaforme eterogenee come client *thin* (ovvero coloro che si limitano alla semplice visualizzazione) o *thick* (ovvero dotati di intelligenza locale che opera direttamente sul sistema operativo). Tali piattaforme, al giorno d'oggi, potrebbero essere ad esempio telefoni mobili, tablet, laptop e workstation.
- **Resource pooling**. Le risorse di calcolo del fornitore vengono messe a disposizione in comune per poter servire molteplici consumatori utilizzando un modello condiviso, conosciuto come multi-tenant, con le diverse risorse fisiche e virtuali assegnate e riassegnate dinamicamente in base alla richiesta. Dato il senso di indipendenza dalla locazione fisica, l'utente generalmente non ha controllo o conoscenza dell'esatta ubicazione delle risorse fornite, ma può essere in grado di

specificare la posizione ad un livello superiore di astrazione (come ad esempio paese, Stato o data center). Esempi di risorse includono memoria, elaborazione e larghezza di banda della rete.

- **Rapid elasticity**. Le risorse possono essere acquisite e rilasciate elasticamente, in alcuni casi anche automaticamente, per scalare rapidamente verso l'esterno e l'interno in relazione alla domanda. Al consumatore le risorse disponibili spesso appaiono illimitate e fruibili in qualsiasi quantità ed in qualsiasi momento.

- **Measured service**. I sistemi cloud controllano automaticamente e ottimizzano l'uso delle risorse, facendo leva sulla capacità di misurazione ad un livello di astrazione appropriato per il tipo di servizio (ad esempio memoria, elaborazione, larghezza di banda e utenti attivi). L'utilizzo delle risorse può essere monitorato, controllato e segnalato, fornendo trasparenza sia per il fornitore che per l'utilizzatore del servizio stesso.

1.1.2 Servizi del cloud computing

Fin dagli albori del cloud computing, le tipologie di servizio messe a disposizione dei consumatori sono state tre, ovvero il *Software as a Service* (SaaS), il *Platform as a Service* (PaaS) ed infine l'*Infrastructure as a Service* (IaaS). Da questi tre servizi preponderanti ne sono poi nate altre tipologie, come ad esempio il *Data as a Service* (DaaS) e l'*Anything as a Service* (XaaS), che però non vengono menzionate ufficialmente nella definizione di cloud computing proposta dal NIST.

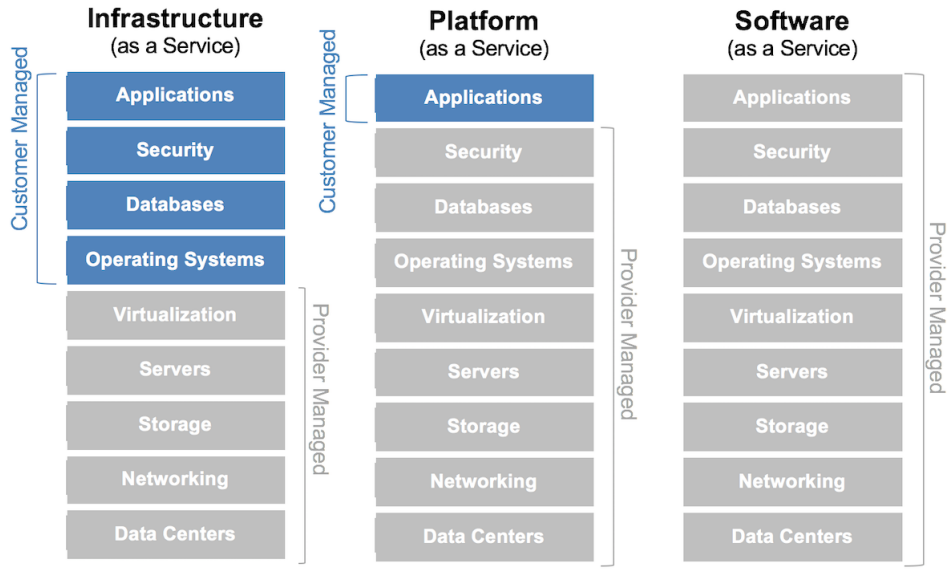

Figura 1.1: *Struttura dei principali servizi cloud [7]*

- **SaaS**. Si fornisce al consumatore la possibilità di utilizzare le applicazioni del fornitore all'interno di un'infrastruttura cloud. Le applicazioni sono accessibili da diversi dispositivi attraverso un'interfaccia *thin*, come ad esempio un'applicazione email su browser, oppure da programmi dotati di apposita interfaccia. Il consumatore non gestisce o controlla l'infrastruttura cloud sottostante, compresi rete, server, sistemi operativi, memoria e nemmeno le capacità delle singole applicazioni, con la possibile eccezione di limitate configurazioni a lui destinate.

- **PaaS**. Si fornisce al consumatore la possibilità di distribuire sull'infrastruttura cloud applicazioni create in proprio oppure acquisite da terzi, utilizzando linguaggi di programmazione, librerie, servizi e strumenti supportati dal fornitore. Il consumatore non gestisce né controlla l'infrastruttura cloud sottostante, compresi rete, server, sistemi operativi e memoria, ma ha il controllo sulle applicazioni ed eventualmente sulle configurazioni dell'ambiente che le ospita.

- **IaaS**. Si fornisce al consumatore la possibilità di acquisire elaborazione, memoria, rete ed altre risorse fondamentali di calcolo, inclusi sistemi operativi e applicazioni. Il consumatore non gestisce né controlla l'infrastruttura cloud sottostante, ma controlla sistemi operativi, memoria, applicazioni ed eventualmente, in modo limitato, alcuni componenti di rete, come ad esempio il firewall.

Come mostrato nella Figura 1.1 appaiono evidenti le possibilità di personalizzazione e controllo che il consumatore ha nei confronti dei diversi servizi messi a sua disposizione. Il SaaS è il servizio più semplice e non richiede pertanto alcun tipo di controllo da parte dell'utente. Esso potrà infatti utilizzare una serie di software resi disponibili online, evitando dunque tutta la trafila relativa all'installazione sulla propria macchina e le configurazioni che ne seguirebbero. Il PaaS lascia invece libertà al consumatore per quanto riguarda il mondo della applicazioni, mentre l'IaaS è il servizio maggiormente configurabile e pensato per un bacino d'utenza con particolari necessità di controllo sul sistema.

1.1.3 Modelli di sviluppo del cloud computing

Anche in questo caso il modello proposto dal NIST nel corso degli anni non ha subito particolari variazioni. Mentre nella sezione precedente ho focalizzato l'attenzione sul punto di vista del consumatore, con i servizi a lui offerti e pronti per essere utilizzati, ora sposto il focus sul versante opposto, ovvero quello dei fornitori e dei modelli di sviluppo per una rete cloud. Esistono principalmente quattro tipologie di modelli, che sono cloud privato, cloud pubblico, cloud ibrido e cloud comunitario.

- **Cloud privato**. L'infrastruttura cloud è fornita per uso esclusivo da parte di una singola organizzazione comprendente molteplici consumatori (ad esempio filiali).

Può essere posseduta, diretta e gestita dall'organizzazione stessa, da un società terza o da una combinazione delle due e può esistere dentro o fuori le proprie sedi. Lo stesso ente che ne richiede la costruzione usufruisce anche del sistema.

- **Cloud pubblico**. L'infrastruttura cloud è fornita per un uso aperto a qualsiasi consumatore. Può essere posseduta, diretta e gestita da un'azienda, da un'organizzazione accademica o governativa oppure da una combinazione delle precedenti. Esiste dentro le sedi del fornitore cloud. In questo caso dunque il fornitore eroga al pubblico un servizio di cloud computing ed è responsabile della sua manutenzione e del corretto funzionamento delle macchine.

- **Cloud ibrido**. L'infrastruttura è una composizione di due o più infrastrutture cloud (privata, comunitaria o pubblica) che rimangono entità distinte, ma unite attraverso tecnologie standard o proprietarie, che abilitano la portabilità di dati e applicazioni (ad esempio per bilanciare il carico di lavoro tra cloud).

- **Cloud comunitario**. L'infrastruttura cloud è fornita per uso esclusivo da parte di una comunità di consumatori di organizzazioni con interessi comuni (ad esempio requisiti di sicurezza, vincoli di condotta e di conformità). Può essere posseduta, diretta e gestita da una o più delle organizzazioni della comunità, da una società terza o da una combinazione delle due e può esistere dentro o fuori le proprie sedi.

1.1.4 Cloud classico e cloud P2P a confronto

Siamo tutti abituati a pensare al cloud sulla base delle esperienze di vita quotidiana: chi non ha mai sentito parlare di Dropbox o di Google Drive? Le multinazionali di questi prodotti hanno costruito in giro per il mondo grandi data center che contengono numerosi server all'interno dei quali vengono salvati i dati dei propri clienti. Solitamente i data center vengono costruiti in luoghi freddi poiché bisogna adottare meccanismi per la dissipazione del calore generato al loro interno. Uno dei problemi principali legato al cloud tradizionale è il *single point of failure* [8]. Essendo in questo caso il cloud interpretato come un'unità centralizzata, nel momento in cui uno specifico data center dovesse subire seri problemi (come ad esempio la distruzione in seguito ad una calamità naturale), tutto ciò in esso contenuto andrebbe irrimediabilmente perduto. Un'altra delle problematiche legate al cloud classico è quella relativa allo Stato in cui è sito il data center [9]. Se i governi dovessero porre restrizioni sui dati sensibili, si potrebbero avere delle problematiche nella loro gestione. Per questi motivi, nel corso degli ultimi anni, la tendenza è quella di creare architetture federate [10], che prevedono la costruzione di più data center interconnessi da una linea veloce. Da questa soluzione poi si è iniziato anche a valutare la possibilità di creare una rete cloud

P2P, dove i singoli calcolatori potrebbero rappresentare allo stesso tempo sia i client della rete ma anche i server. In questo modo si andrebbero a neutralizzare i problemi legati alla dissipazione del calore, ai data center e allo Stato di appartenenza. Nessuno potrebbe decidere di spegnere questo cloud poiché si tratterebbe di un'architettura in cui tutti i peer della rete avrebbero pari diritti rispetto a quelli degli altri. Anche in questo caso però potrebbero sorgere delle complicazioni: il cloud, per definizione, dovrebbe essere affidabile, ma essendo composto da una serie di dispositivi non affidabili (potrebbero infatti essere accesi e spenti a piacimento dagli utenti che li controllano) bisognerebbe implementare dei meccanismi per capire quali siano i dispositivi attualmente disponibili e quali invece siano spenti. Nel momento in cui ci si trovasse di fronte alla scelta relativa a quale tipologia di cloud utilizzare, bisognerebbe valutare ciò di cui si ha bisogno. Qualora si ricercasse qualità nel servizio la scelta migliore ricadrebbe sul cloud centralizzato, che permetterebbe di avere prestazioni migliori. Qualora invece si scegliesse il cloud P2P si eviterebbero tutti i problemi sopra elencati ma si andrebbe incontro, con ogni probabilità, ad una qualità di servizio minore. Nel caso di questa seconda soluzione, ogni utente metterebbe a disposizione della rete le proprie risorse (composte da RAM, processore e spazio di archiviazione) in attesa che altri peer riescano ad individuarlo e a contattarlo mediante algoritmi di gossip, così da poter creare e mantenere aggiornata una rete di utenti dislocati in tutto il mondo. Dal momento in cui il prototipo che mi è stato messo a disposizione è basato su un'architettura cloud P2P, ho deciso di approfondire maggiormente questo aspetto, valutandone prima di tutto la struttura. In generale in un'architettura cloud P2P chiameremo nodo o peer un'istanza del software in esecuzione su un calcolatore [11]. Possono esserci più nodi presenti all'interno della stessa macchina, ma questo in uno scenario reale non avrebbe molto senso. I nodi sono interconnessi da una rete costruita su di una rete standard come quella di Internet. Il coordinamento tra i diversi nodi viene realizzato in maniera totalmente decentralizzata attraverso decisioni locali prese singolarmente dal nodo in questione.

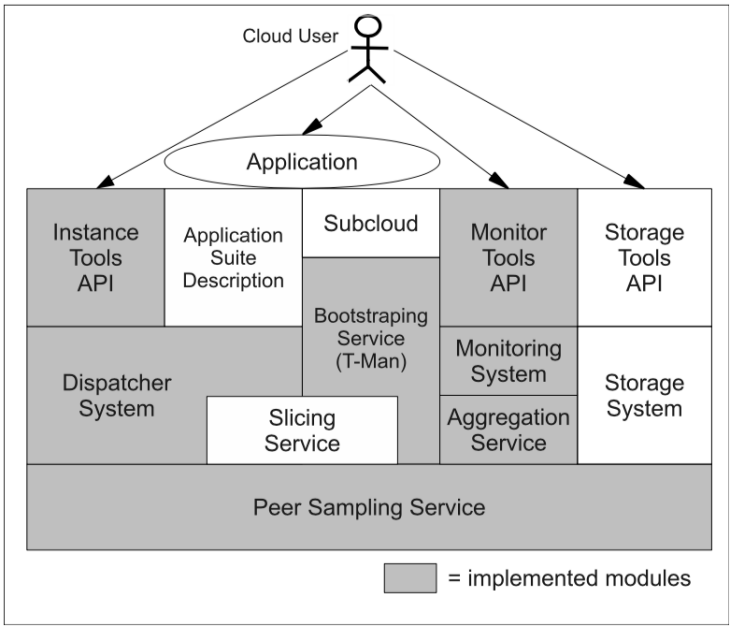

Figura 1.2: *Architettura del sistema cloud P2P [1]. I componenti bianchi non sono ancora stati implementati in questo prototipo.*

La Figura 1.2 rappresenta l'architettura del prototipo del sistema cloud P2P. Di seguito analizzerò i diversi componenti che la caratterizzano.

- **Peer Sampling Service (PSS)**. Questo primo componente rappresenta le fondamenta dell'architettura in questione. Si tratta di un algoritmo che permette di poter costruire una rete condivisa dai nodi che fanno parte del sistema. Si presuppone che i nodi che entrano all'interno della rete per la prima volta non conoscano tutti i nodi facenti parte della stessa rete. L'obiettivo di questo algoritmo è quello di creare dei collegamenti tra i vari nodi del sistema, affinché possano conoscersi e scambiarsi informazioni. I nodi non tengono traccia di tutti i peer presenti all'interno della rete con relativi indirizzi, bensì ne conservano solamente una parte all'interno di una *partial view*. Periodicamente ogni nodo seleziona un peer dalla propria *partial view* e gli invia una parte dei nodi conosciuti. In questo modo i peer della rete continueranno a scambiarsi informazioni sui vicini per mantenere all'interno delle view solamente quelli attivi. Questo algoritmo permette di eliminare i *churn*, ovvero quei peer non più attivi che non portano alcun contributo alla rete.

- **Slicing Service**. Sfruttando i collegamenti offerti dal PSS, lo Slicing Service è in grado di ordinare i nodi rispetto ad uno specifico attributo. Una volta terminato l'algoritmo è possibile richiedere al servizio delle porzioni della rete che corrispondono ad un sottoinsieme di questa.

- **Aggregation Service**. Questo servizio per poter essere messo in atto ha bisogno di collaborare con il PSS. L'obiettivo è quello di distribuire ai peer presenti

all'interno della rete informazioni globali come ad esempio la dimensione della rete stessa. L'algoritmo richiede che ciascun peer mantenga una variabile condivisa, che dovrà essere aggiornata dopo aver scambiato con un altro peer il valore di tale variabile. Avvenuto lo scambio, entrambi i peer applicano una funzione di *update* sui due valori ottenendone un terzo sulla base del tipo di funzione applicata. Attraverso l'aggregazione è possibile ricavare alcune importanti funzioni come il calcolo della media e l'estrazione degli estremi di massimo e minino, che possono rivelarsi utili per la costruzione del sistema di monitoraggio.

- **Bootstraping Service**. Si tratta di un servizio che permette di creare un *overlay* tra i nodi che partecipano al sistema. Si potrebbe infatti voler creare una subcloud tra specifici nodi della rete. In questo modo ad ogni subcloud corrisponde un BS dedicato, che opera in un certo senso come un controllore: fornisce l'accesso diretto all'applicazione costruita dall'utente, dialoga con il sistema di monitoraggio o i servizi sottostanti, quali aggregazione o PSS, per eseguire attività di verifica. In questo prototipo è stato utilizzato l'algoritmo TMan che avrà come compito quello di costruire gli *overlay*. Questo algoritmo genera un grafo la cui topologia è definita a priori attraverso una funzione *rank*, che ordina i vicini di un nodo sulla base di un valore di similitudine calcolato dalla stessa. Dopo aver fatto ciò, sarà in grado di collegare i nodi in anelli oppure alberi.

- **Moduli di sistema**. Il sistema di monitoraggio (Monitoring System) ha come obiettivo quello di monitorare costantemente le risorse, sapendo quante sono libere e quante invece sono occupate. Il sistema di archiviazione (Storage System) si occupa della memorizzazione delle risorse, mentre il dispatcher (Dispatcher System) si occupa di gestire l'ambito delle comunicazioni.

- **Moduli di interfaccia**. Questo componente include tutte le Application Programming Interface (API) che potranno essere utilizzate dall'utente per dialogare con il sistema. Un esempio di API potrebbe essere quello relativo alla creazione di un *overlay*, dove l'utente specifica il nome della subcloud ed il numero di nodi che la costituirà.

1.2 L'anonimizzazione delle reti

Per quanto diversi utenti della rete Internet non abbiano nulla da nascondere o segreti da celare, è sempre più frequente, nel corso degli ultimi tempi, la ricerca dell'anonimato anche per le più semplici azioni come quella di navigare all'interno di una pagina web. Le notizie che circolano in rete nei confronti di questo argomento creano spesso il panico nell'utente, soprattutto con l'avvento dei social network e con la gestione dei dati ad essi collegati. In un articolo di qualche anno fa [12] il comportamento delle più grandi multinazionali venne messo in discussione, con milioni di utenti che hanno iniziato a temere per la salvaguardia del proprio anonimato. L'obiettivo di questa sezione è quello di introdurre l'anonimizzazione delle reti, argomento che verrà ripreso in maniera più tecnica nei capitoli successivi quando descriverò nel dettaglio l'implementazione di tali reti all'interno del prototipo. Dapprima tenterò dunque di inquadrare l'anonimato considerando ciò che significava centinaia di anni fa e ciò che invece significa ora, per poi introdurre le reti di anonimizzazione, prima di un'analisi più dettagliata nel capitolo successivo.

1.2.1 L'importanza dell'anonimato dalle fondamenta

L'anonimato ha sempre avuto un ruolo fondamentale all'interno della civiltà. Fin dal '700 infatti [13] era pratica comune per gli scrittori utilizzare pseudonimi o rimanere nell'anonimato. Questa situazione era data dal fatto che molto spesso si temevano censure e critiche, cosicché gli scrittori preferivano anteporre la propria libertà ad un eventuale successo dovuto al proprio lavoro. Oggigiorno le cose sono un po' cambiate: non si parla più di opere scritte a mano ma molto spesso di post scritti sui social network, oppure di ricerche effettuate tramite i motori di ricerca di Internet. La sostanza però non cambia, trattandosi sempre di anonimato. Ma perché le persone avrebbero bisogno di nascondersi dietro ad un'identità *fake*? Nel caso di utenti malintenzionati è abbastanza ovvio il motivo, ma per tutti coloro che non hanno cattive intenzioni, quali potrebbero essere le motivazioni? Non ci sono risposte giuste o risposte sbagliate, ma alcuni casi che potrebbero indurre alla ricerca dell'anonimato sono qui di seguito trattati.

- **Privacy**. Sebbene tale termine si discosti da quello relativo all'anonimato, molto spesso vengono messi sullo stesso piatto della bilancia. Ad esempio potrei scrivere un post su un blog con un nickname fittizio, preservando pertanto la privacy. Non è però detto che il mio anonimato venga preservato, poiché per garantirlo servirebbero ben altre tecniche rispetto ad un banale nickname. Ad ogni modo, si può dire che la privacy sia una delle componenti relative all'anonimato, e che sia pertanto lecito ricercarla anche nel mondo di Internet. Molto spesso si sente controbattere con frasi come: *"Ma se non hai nulla da nascondere, perché ricerchi*

la privacy?". Una possibile risposta può essere quella data da William Stallings, scrittore americano: *"Garantire che le persone mantengano il diritto di controllare quali informazioni vengono raccolte su di loro, come vengono utilizzate, chi le ha utilizzate, chi le mantiene e per quale scopo vengono utilizzate"* [14].

- **Reclami anonimi**. Potrebbe capitare di voler denunciare un fatto temendo di avere ripercussioni sulla propria salute e sulla propria vita, pertanto si potrebbe desistere e magari lasciare impunito un certo crimine. Con l'anonimato invece avremmo la possibilità di denunciare una specifica azione senza paura di subire ritorsioni.

- **Libertà di pensiero e di parola**. Quante volte sentiamo nei notiziari o leggiamo articoli che riguardano la libertà di pensiero e parola vietata in tantissimi stati del mondo? L'anonimato potrebbe servire anche per proteggere questi diritti che non sempre vengono rispettati, o, nei casi peggiori, per permettere a tutti di poter manifestare la propria idea ed il proprio pensiero senza paura di essere scoperti.

- **Posizione geografica**. Rispetto alle precedenti motivazioni, questa è strettamente legata all'utilizzo di Internet. Mediante l'Internet Protocol address (indirizzo IP) è infatti possibile localizzare una specifica macchina in una circoscritta area geografica. Un malintenzionato, per vendicarsi di un torto subito, potrebbe rintracciare l'indirizzo IP della persona che vuole punire e agire di conseguenza. Ovviamente ho estremizzato la questione, ma l'idea è voler preservare l'anonimato anche per queste possibili situazioni.

- **Informazioni tracciabili**. Senza anonimato un utente malintenzionato potrebbe scoprire quali siti web l'utente di un determinato computer visita. Anche le informazioni che l'utente ricerca potrebbero essere intercettate e visualizzate qualora non venissero adottati meccanismi di crittografia. Questo implica il fatto di mettere a rischio i propri dati sensibili e l'anonimato potrebbe lenire questa possibilità.

Le motivazioni sopra citate sono solamente alcune di quelle immaginabili e che potrebbero comunque bastare ad invogliare un utente a ricercare l'anonimato su Internet. In generale però è molto importante capire come la ricerca dell'anonimato non sia automaticamente sinonimo di truffa o cattive intenzioni, tutt'altro. L'anonimato è una risorsa, è una possibilità che deve essere data ma soprattutto accettata anche dagli scettici. Attualmente le comunicazioni possono già essere rese anonime così come la navigazione su Internet può essere anonimizzata. Per effettuare però calcoli anonimi tutto ciò non basta. È necessario infatti qualcosa come un sistema cloud anonimo e le architetture P2P sono il miglior punto di partenza, poiché garantiscono di base il

fatto di non aver alcun tipo di controllo centralizzato, dal momento che tutti i nodi che compongono tale rete sono peer. Ma a cosa potrebbe servire nel concreto realizzare un sistema cloud P2P anonimo? Perché potrebbe essere interessante sfruttare tale tecnologia? Una prima applicazione che potrebbe trarre vantaggio dall'utilizzo di un sistema cloud P2P anonimo è quella relativa al salvataggio dei dati nel cloud. Ogni utente infatti potrebbe mettere a disposizione degli altri peer della rete una porzione di memoria del proprio calcolatore o dei propri dispositivi di archiviazione esterni. Se tutti gli utenti appartenenti a questo sistema cloud mettessero a disposizione degli altri utenti spazi di memoria, si riuscirebbe a creare un'infrastruttura molto vasta. In questo caso la prerogativa del sistema sarebbe quella di mantenere l'anonimato dei dati ma soprattutto quella di garantire l'accesso esclusivamente al proprietario. L'idea è dunque quella di permettere agli utenti di poter salvare i propri file (che siano immagini, video, documenti di testo e quant'altro) negli spazi di memoria offerti dagli utenti della rete e nello stesso tempo mettere a disposizione una parte dei propri spazi. In questo modo i peer, dopo una fase iniziale di conoscenza reciproca, potrebbero sfruttare gli spazi di memoria messi a disposizione dagli altri peer della rete. In questo caso l'anonimato e il controllo degli accessi sarebbero fondamentali: da una parte chi mette a disposizione la propria memoria dovrà avere la garanzia che nessuno riesca a controllare eventuali altri dati di sua proprietà presenti all'interno del dispositivo, dall'altra coloro che salvano i propri dati sui dispositivi della rete dovranno essere sicuri che tali dati potranno essere acceduti esclusivamente dal proprietario. Un sistema di questo tipo potrebbe fare concorrenza al cloud centralizzato, soprattutto perché si eviterebbero problemi come il *single point of failure*. D'altro canto però la qualità del servizio sarebbe inferiore, pertanto tale applicazione potrebbe essere pensata per un'infrastruttura privata o per un'azienda di piccole dimensioni. Una seconda applicazione che sfrutterebbe il sistema cloud P2P anonimo potrebbe essere realizzata nell'ambito dei *Massively multiplayer online games* (MMOG) [15]. Negli ultimi tempi infatti il cloud P2P sta diventando una soluzione accattivante per questo tipo di giochi online, in quanto solleva gli operatori dall'onere di acquistare e mantenere grandi quantità di risorse di calcolo, archiviazione e comunicazione, offrendo al contempo un'infinita scalabilità. A tal riguardo però le risorse cloud non sono disponibili gratuitamente, così si cerca di ridurre al minimo i costi mediante un'attenta valutazione delle risorse stesse. A questo sistema P2P si potrebbe poi aggiungere l'anonimato, che in alcuni casi potrebbe davvero arrivare a salvare la vita dei giocatori. Come analizzato infatti da [16], è sempre più frequente la tentazione di malintenzionati di provare ad adescare dei giovani giocatori all'interno di questo mondo virtuale, per poi convincerli ad agire contro la propria volontà. Con l'anonimato si riuscirebbero a mantenere nascoste tutte quelle informazioni che potrebbero essere sfruttate in maniera negativa da parte di queste persone, nella speranza di riuscire a limitare o ad eliminare totalmente questo genere di attività online illecita.

1.2.2 L'anonimato nelle reti Internet

Al giorno d'oggi, le reti di anonimizzazione più conosciute sono Tor e I2P, ma ne esistono altre come ad esempio Freenet [17] che si pongono come obiettivo quello di salvaguardare l'anonimato degli utenti della rete. Nel prototipo al quale ho lavorato, come indicato precedentemente, era già implementata la rete di anonimizzazione Tor, mentre io ho aggiunto quella relativa alla rete I2P. Pertanto nel prossimo capitolo prenderò in considerazione queste due reti, che al giorno d'oggi rappresentano i due colossi nel mondo dell'anonimizzazione. Prima di fare però il passo verso questo interessante confronto, è bene comprendere come possano avvenire i contatti all'interno di una rete classica. L'elemento fondamentale per far sì che una comunicazione possa essere instaurata è l'indirizzo IP [18]. Basandosi il mondo di Internet sul semplice principio relativo al trasferimento di informazioni da un calcolatore all'altro, è necessario che ciascun calcolatore possa essere identificato mediante un indirizzo IP. Genericamente, l'indirizzo IP è un'informazione univoca che viene consegnata a tutti i componenti della rete nel momento in cui viene effettuato un collegamento all'interno della rete da parte del calcolatore in questione. Visitare pagine web, inviare email, scaricare un software oppure aggiornarlo: ognuna di queste singole azioni distribuisce all'interno della rete il nostro indirizzo IP, ovvero la chiave per identificarci. Ci sono diverse tipologie di indirizzo IP, ma non è questa la sezione adatta per spiegarne il funzionamento. Una volta discusso ciò, prendiamo come esempio due entità su Internet, Bob e Alice. Bob conosce l'indirizzo di Alice, pertanto riesce a contattarla. A questo punto anche Alice entra in possesso dell'indirizzo di Bob, che può utilizzare per rispondere al messaggio precedentemente ricevuto. Questa potrebbe essere la classica situazione di due utenti che non hanno nulla in contrario nel condividere il proprio indirizzo. Potrebbe però capitare di andare incontro ad una situazione in cui, per disparati motivi, Bob vorrebbe comunicare con Alice senza rivelare la propria identità. Bob allora potrebbe contattare una terza entità su Internet utilizzandola come *relay*, affinché consegni il suo messaggio ad Alice. In questo modo Alice penserà di aver ricevuto il messaggio da questa terza entità invece che da Bob. Ma questa soluzione non basta: anche Alice, in questo caso la destinataria, vorrebbe agire nell'anonimato nascondendo la propria identità. Potrebbe dunque servirsi di una quarta entità su Internet, anch'essa con funzione di *relay*, che manterrà l'indirizzo di Alice spiegando che potrà essere contattata solamente mediante tale *relay*. Così facendo colui che volesse contattare Alice dovrebbe passare dalla quarta entità senza conoscere direttamente l'indirizzo di Alice. Anche questa soluzione presenta però una falla, ed è quella relativa ai *relay*. Chi ci assicura che queste entità non siano corrotte e che non rivelino tali informazioni custodite gelosamente? Ovviamente le reti di anonimizzazione sono molto più complesse della situazione appena presentata, che può però essere utile per iniziare a comprendere il tipo di ragionamento all'interno di tale mondo. Per riassumere questa breve introduzione relativa all'anoni-

mizzazione, è importante domandarsi come si potrebbe effettivamente raggiungere tale scopo all'interno delle reti odierne. La prima idea è quella di aggiungere step intermedi tra il mittente ed il destinatario, ovvero aumentare il numero di passaggi all'interno di un percorso cosicché il pacchetto debba seguire strade più tortuose all'interno della rete prima di raggiungere l'effettiva destinazione. In questo modo sarebbe piuttosto complesso riuscire a identificare la vera fonte del messaggio (questo principio è la base sia di Tor che di I2P, come esplicherò nel dettaglio nel capitolo successivo). La seconda idea riguarda la crittografia. Considerando solamente la prima soluzione, riusciremmo ad inviare i nostri dati e le nostre informazioni attraverso la rete, senza (probabilmente) essere identificati come i mittenti. Però tali informazioni viaggerebbero attraverso la rete in chiaro, non potendo dunque mantenere integrità (assicura che le informazioni vengano modificate solo in maniera autorizzata) e riservatezza (informazioni private o riservate non sono rese disponibili o divulgate a persone non autorizzate) [14]. Per risolvere questi problemi la crittografia [19], ovvero la *"tecnica di rappresentazione di un messaggio in una forma tale che l'informazione in esso contenuta possa essere recepita solo dal destinatario"* [20], rappresenta una delle componenti fondamentali del nostro sistema di anonimato.

1.2.3 Lavori correlati

Analizzando la situazione attuale nei confronti dell'argomento di questa tesi, non sono molti i progetti che ho trovato sul web con la struttura basata su sistemi cloud con architettura P2P. Quello che più mi ha colpito è il lavoro di tesi di Michael Ammann, uno studente svizzero che ha progettato e realizzato l'implementazione di un sistema P2P per abilitare le funzionalità di condivisione su una piattaforma cloud [21]. Secondo lo studente, uno dei principali vantaggi dell'utilizzare un'architettura P2P è legato al problema della centralizzazione. Sono molti infatti i servizi cloud che si basano su un'architettura centralizzata, risultando dunque essere soggetti ad inconvenienti come ad esempio le calamità naturali. I sistemi P2P sono una buona alternativa per affrontare le limitazioni dei sistemi centralizzati nell'ambito della condivisione dei file, grazie all'utilizzo condiviso delle risorse distribuite e alla maggiore tolleranza agli errori. Lo studente all'interno di questo lavoro ha progettato, implementato e valutato un sistema di condivisione di file basato sul P2P da integrare in PiCsMu [22], una nuova applicazione di cloud storage indipendente dalla piattaforma. Il sistema proposto presenta un design modulare che si basa su interfacce ben definite ed un concetto di archiviazione P2P con buone prestazioni. Funzionalità aggiuntive che tentano di distinguere questo approccio dagli altri sistemi di condivisione file P2P includono una ricerca file integrata e la possibilità di condividere file privatamente. Contrariamente al prototipo sul quale ho lavorato, non vengono sfruttate le reti di anonimizzazione come Tor e I2P, sebbene all'interno della tesi vengano introdotti i sistemi anonimi P2P. Un altro lavoro molto

interessante dal punto di vista teorico è quello di Risto Laurikainen, sviluppatore finlandese che ha scritto un trattato sulle comunicazioni sicure ed anonime all'interno del cloud [23]. L'obiettivo di questo paper è quello di stabilire un elenco di requisiti per un sistema di comunicazione sicuro e anonimo e quindi esamina come i sistemi esistenti possano essere utilizzati per soddisfare tali requisiti. In generale comunque, al giorno d'oggi, non ho trovato alcun tipo di lavoro o ricerca che prendesse in considerazione nello stesso momento un sistema cloud P2P e l'anonimizzazione delle reti attraverso Tor e I2P.

Capitolo 2

Background informatico del sistema cloud P2P anonimo

Dopo aver introdotto gli argomenti relativi a questa tesi, è giunto il momento di analizzare i principali strumenti informatici che sono stati utilizzati all'interno di questo lavoro. L'obiettivo di questo capitolo è dunque quello di prendere in considerazione tutti gli aspetti tecnici che permetteranno di poter comprendere a fondo questo progetto. Prima di tutto introdurrò le due reti di anonimizzazione presenti all'interno del prototipo, ovvero Tor e I2P. Dopo averne analizzato le caratteristiche, metterò a confronto le due reti per valutarne analogie e differenze. Una volta fatto ciò prenderò in considerazione le specifiche del progetto, descrivendo i principali algoritmi e gli script utilizzati.

2.1 The Onion Router - Tor

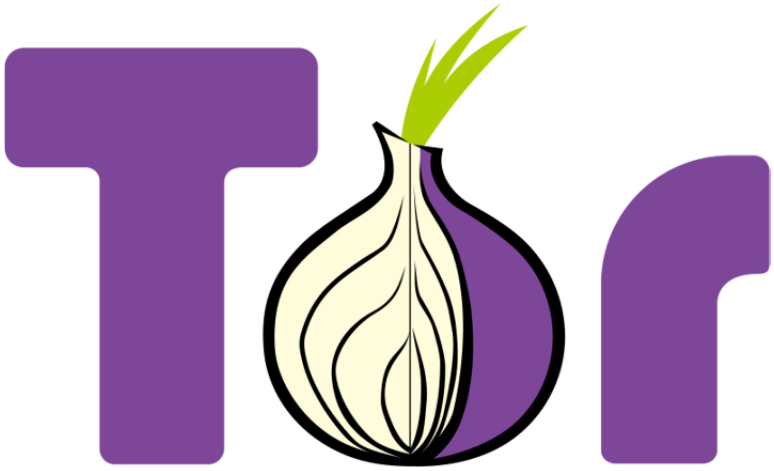

Figura 2.1: *Logo del progetto Tor* [24]

Una delle più famose reti di anonimizzazione è senza dubbio Tor. Il progetto [25] nacque a partire dalla seconda metà degli anni '90, quando la mancanza di sicurezza della rete Internet e la sua vulnerabilità all'analisi del traffico, che permetteva ad utenti malintenzionati di intercettare le comunicazioni trasmesse attraverso la rete stessa, diedero l'input a David Goldschlag, Mike Reed e Paul Syverson, presso il laboratorio di ricerca navale degli Stati Uniti, di iniziare a lavorare ad un sistema che garantisse l'anonimato delle comunicazioni online [26]. Nel corso di questi anni tale progetto è riuscito a coinvolgere tantissimi utenti: ad oggi sono infatti quasi due milioni [27] coloro che utilizzano questa tipologia di rete per navigare nel web. La prima versione della rete Tor è stata distribuita nel 2002, mentre nel 2004 è stata pubblicata la seconda generazione del protocollo Onion Router [28]. Da quel momento in poi la crescita della rete Tor è rimasta costante fino ai giorni nostri.

2.1.1 Caratteristiche di Tor

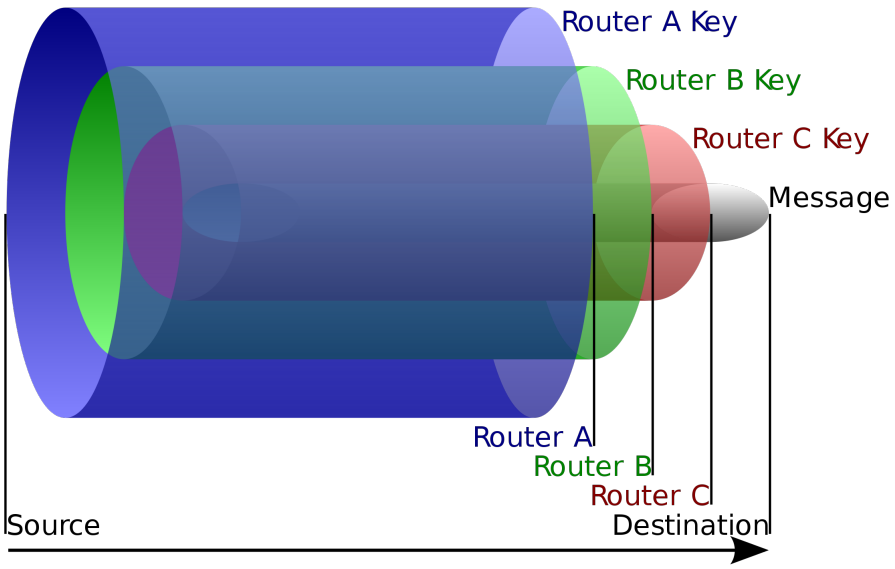

Figura 2.2: *Struttura a cipolla di un pacchetto Tor* [29]

L'obiettivo principale di Tor è quello di permettere agli utenti di poter navigare online in maniera del tutto anonima, scoraggiando così eventuali attacchi informatici perpetrati da utenti malintenzionati. Tor è un servizio di comunicazione anonima basato su circuito, che opera su una rete distribuita costruita su Internet e progettata per anonimizzare le applicazioni basate sul Transmission Control Protocol (TCP). La particolarità principale di questa rete di anonimizzazione si basa sul percorso che i pacchetti in uscita da uno specifico terminale seguono prima di poter raggiungere il destinatario (per una spiegazione più dettagliata si veda la sezione 2.1.4). Contrariamente a quanto succede per la rete Internet, i pacchetti vengono spediti verso router intermedi prima di poter raggiungere l'effettiva destinazione. I pacchetti che giungono a questi router però non sono trasmessi in chiaro, bensì crittografati con più livelli di protezione: da qui deriva il nome di questo progetto, che fa riferimento agli strati di una cipolla [30]. La Figura 2.2 mostra come il messaggio venga crittato con più chiavi, ognuna delle quali è in possesso dei router intermedi. Il primo router del circuito dunque decritterà con la propria chiave il primo strato del messaggio, ma non potrà fare altro che inviarlo al router successivo perché non è in possesso della chiave per decrittare il secondo strato del messaggio. Si procederà in questo modo fino a quando il messaggio non verrà consegnato dall'ultimo router del circuito alla destinazione effettiva (si noti che in quest'ultimo tratto il messaggio non è coperto da alcuno strato di crittografia [31]). Questo tipo di progetto si basa prevalentemente sulla partecipazione degli utenti, che possono decidere di sostenerlo diventando essi stessi router intermedi

della rete. Prima di entrare nel dettaglio delle sue specifiche, analizzo di seguito le principali caratteristiche relative al protocollo utilizzato nella rete Tor [28].

- **Segretezza dei nodi successori**. Come detto in precedenza, ogni nodo conosce esclusivamente il proprio predecessore ed il proprio successore, mentre non è a conoscenza di nessun altro nodo della rete. Una delle caratteristiche principali di Tor è proprio quella di evitare che un eventuale nodo ostile riesca a riconoscere tutti i nodi, o gran parte di essi, presenti all'interno della rete per plasmare di seguito i flussi TCP a proprio vantaggio.

- **Topologia dei circuiti a perdite**. Una delle debolezze del protocollo della rete Tor è relativo ai router di uscita della rete. Un eventuale attaccante infatti potrebbe studiare i circuiti della rete riuscendo a capire quali siano i router finali del percorso. Per risolvere questo problema, i router intermedi potrebbero dirottare il traffico verso i nodi a metà del circuito per variare il percorso dei pacchetti e confondere eventuali attaccanti.

- **Controllo della congestione**. I nodi ai bordi della rete, mediante funzioni end-to-end [32] per mantenere l'anonimato, riescono a rilevare eventuali congestioni presenti all'interno di uno specifico percorso e di conseguenza invieranno meno dati al suo interno fino a quando la congestione non si sarà attenuata.

- **Server di directory**. Nella rete Tor esistono nodi più affidabili di altri, detti server di directory. Da questi nodi periodicamente gli utenti scaricano informazioni relative ai router della rete e al loro stato attuale.

- **Controllo integrità end-to-end**. Uno degli obiettivi principali della rete Tor è quello di mantenere l'integrità dei dati. Questo significa che i router intermedi non possono modificare il messaggio che stanno trasportando a proprio piacimento, ma l'unica operazione che possono fare è quella di decrittare il proprio strato della cipolla e capire a quale nodo successivo inviare il messaggio.

- **Punti rendezvous e servizi nascosti**. In Tor è fondamentale la salvaguardia dell'anonimato dei servizi, oltre a quella dei client. Sono dunque i client a negoziare i punti necessari (punti rendezvous) per connettersi ai servizi nascosti, che così potranno mantenere l'anonimato indipendentemente da chi deciderà di visitarli.

Dopo aver analizzato le caratteristiche del protocollo di rete Tor, è importante anche considerare le caratteristiche relative al design che gli sviluppatori hanno deciso di seguire per poter realizzare questa rete di anonimizzazione.

- **Distribuibilità**. Il servizio deve essere distribuito e utilizzato nel mondo reale. Pertanto, non deve essere troppo oneroso dal punto di vista delle risorse (ad esempio, richiedendo più larghezza di banda di quella che i volontari sono disposti a fornire), non deve costituire un grave onere per gli operatori (ad esempio, consentendo agli aggressori di implicare gli Onion Router in attività illegali) e non deve essere difficile o costoso da implementare (ad esempio, richiedendo patch del kernel o proxy separati per ogni protocollo).

- **Flessibilità**. Il protocollo deve essere flessibile e ben documentato, in modo che Tor possa essere utilizzato per eventuali ricerche future. Molti dei problemi presenti nelle reti di anonimato a bassa latenza, come la generazione di traffico fittizio o la prevenzione di attacchi, devono poter essere risolti anche grazie alla semplicità del protocollo.

- **Usabilità**. Un sistema difficile da usare ha un minor numero di utenti e poiché i sistemi di anonimato nascondono gli utenti tra gli utenti, un sistema con un minor numero di utenti offre meno anonimato. L'usabilità non è quindi solo una comodità ma un requisito di sicurezza. Tor non dovrebbe quindi richiedere la modifica di applicazioni, non dovrebbe introdurre ritardi proibitivi e dovrebbe richiedere il minor numero di configurazioni possibile. Infine, Tor dovrebbe essere facilmente utilizzabile su tutte le piattaforme comuni, non potendo richiedere agli utenti di modificare il loro sistema operativo solamente per poter sfruttare questa tecnologia.

- **Semplicità di design**. I parametri di progettazione e di sicurezza del protocollo devono essere ben compresi. Funzionalità aggiuntive impongono costi di implementazione e complessità; l'aggiunta di tecniche non comprovate alla progettazione minaccia la leggibilità e la facilità di analisi della sicurezza. Tor mira a implementare un sistema semplice e stabile che integri gli approcci più chiari per proteggere l'anonimato.

2.1.2 Componenti della rete Tor

Come introdotto nella sezione precedente, la rete Tor si basa sulla presenza di router intermedi attraverso i quali transitano i pacchetti prima di poter raggiungere la propria destinazione. Nel momento in cui un client si connette alla rete Tor, viene scelto casualmente un percorso mediante la costruzione di un circuito, composto da una serie di nodi. Ogni nodo presente all'interno di tale circuito conosce esclusivamente i suoi vicini, ovvero chi lo precede e chi lo segue, senza sapere nulla di tutti gli altri nodi presenti in rete. Esistono due principali tipologie di nodi, gli Onion Router e gli Onion Proxy.

- **Onion Router**. Si tratta del principale tipo di nodo presente all'interno della rete Tor. In una rete deve esserci più di un Onion Router e gli obiettivi di questi nodi sono quello di connettere i client alle destinazioni richieste e quello di inoltrare i dati al nodo successore nel circuito creato appositamente per quel client. Per poter comunicare con gli altri Onion Router, ognuno di essi mantiene attiva una connessione Transport Layer Security (TLS) [33]. Per poter operare nel migliore dei modi, ogni nodo di questo tipo deve mantenere in memoria due chiavi:
  - *Identity key*: è una chiave a lungo termine utilizzata per scopi diversi. I più importanti sono la firma dei certificati TLS e la firma del descrittore del router (un breve sommario delle sue chiavi, contenente informazioni quali indirizzo, larghezza di banda, politica di entrata e uscita).
  - *Onion key*: è una chiave a breve termine e viene utilizzata per decrittografare le richieste provenienti dagli utenti al fine di creare un circuito. Le onion key vengono aggiornate e cambiate periodicamente, per mitigare gli effetti di una possibile compromissione delle chiavi stesse.
- **Onion Proxy**. Si tratta di un software locale che ogni utente deve eseguire per poter partecipare alla rete Tor. Permette di stabilire i circuiti di Tor e gestire le connessioni dalle applicazioni.

2.1.3 Pacchetti della rete Tor

Dopo aver analizzato i due principali componenti della rete Tor, è necessario specificare le diverse tipologie di pacchetti utilizzate da questa rete di anonimizzazione. I pacchetti all'interno della rete Tor vengono chiamati celle. Si tratta di speciali pacchetti di dimensioni fisse (512 byte) [34] che includono un'intestazione ed un *payload*. L'intestazione ingloba un identificatore di circuito (*circID*) che specifica su quale circuito la cella dovrà viaggiare e anche un comando (*cmd*) che definisce come utilizzare il

payload di quella cella. Tor differenzia le celle in due tipologie sulla base del comando ad esse assegnato.

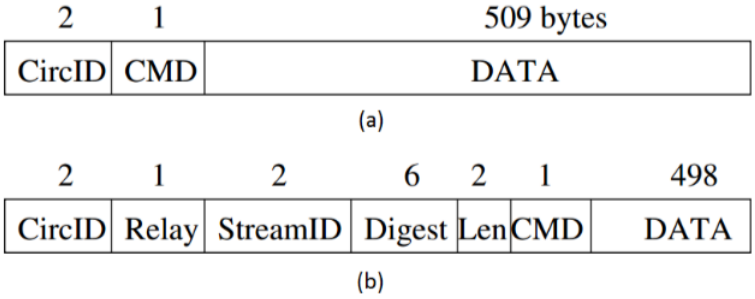

Figura 2.3: *Struttura delle due tipologie di celle nella rete Tor* [28]

- **Celle di controllo**. Queste celle, la cui struttura è riassunta nella Figura 2.3 (a), sono sempre elaborate dal nodo che le riceve e specificano uno dei seguenti comandi:
  - *padding*: utilizzato per mantenere vivo un circuito;
  - *create/created*: il primo viene utilizzato per realizzare un nuovo circuito, mentre il secondo per notificare l'avvenuta creazione di un nuovo circuito;
  - *destroy*: utilizzato per distruggere uno specifico circuito.
- **Celle di relay**. Queste celle, la cui struttura è riassunta nella Figura 2.3 (b), vengono utilizzate per trasportare i dati dal mittente al destinatario. L'intestazione di questo tipo di cella include altri campi come un byte per identificare le celle di inoltro, un identificatore di flusso (*streamID*), una firma digitale (*digest*) per il controllo integrità end-to-end e la lunghezza del *payload* (*len*). I comandi per queste celle sono:
  - *relay data*: utilizzato per inviare i dati lungo il flusso;
  - *relay begin/end*: utilizzato per aprire/chiudere un flusso;
  - *relay teardown*: utilizzato per chiudere un flusso interrotto;
  - *relay connected*: utilizzato per notificare all'Onion Proxy che un relay è stato correttamente avviato;
  - *relay extend/extended*: utilizzato per estendere il circuito di un salto e notificarlo;
  - *relay truncate/truncated*: utilizzato per chiudere una sola parte del circuito e notificarlo;
  - *relay sendme*: utilizzato per il controllo della congestione.

2.1.4 Circuiti della rete Tor

A questo punto le basi della rete Tor sono state analizzate e si può procedere con la creazione del circuito, ovvero il percorso che i pacchetti seguiranno prima di raggiungere la destinazione. Originariamente, il protocollo di Tor generava un circuito per ogni diverso flusso TCP, ma dal momento in cui la creazione di un singolo circuito richiedeva diversi decimi di secondo (a causa della crittografia a chiave pubblica [35] e della latenza della rete) si è deciso che ogni circuito possa essere condiviso da più di un flusso TCP nello stesso istante. Inoltre, per evitare ritardi, i circuiti vengono costruiti a priori e per garantire una maggior sicurezza periodicamente vengono costruiti nuovi circuiti e riassegnati ai client Tor.

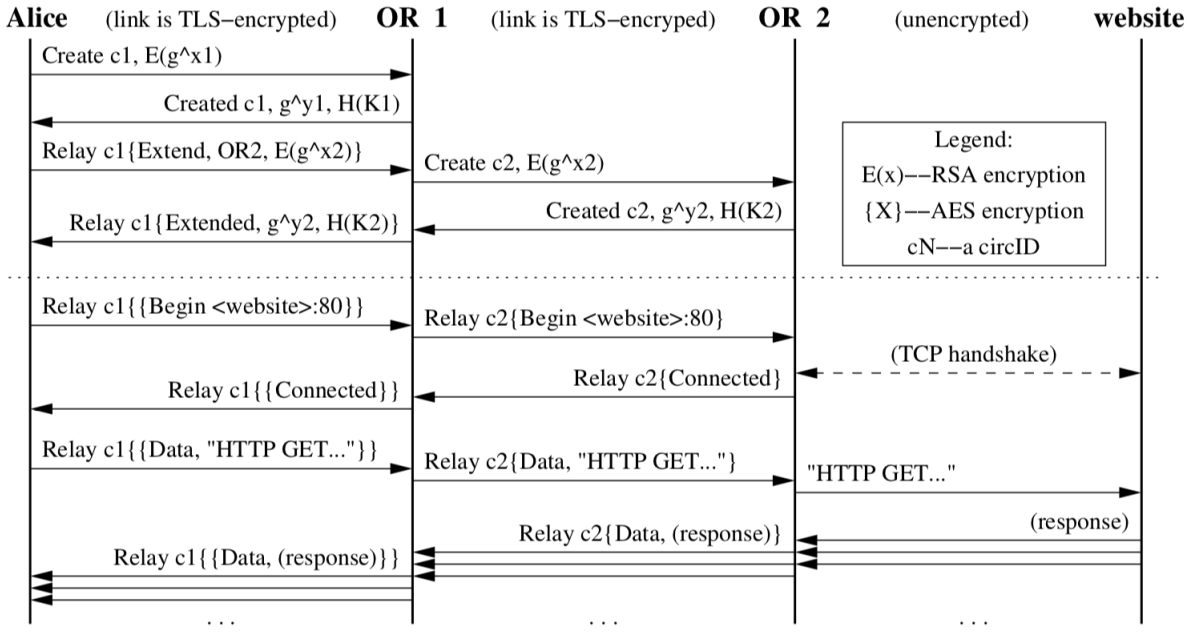

Figura 2.4: *Creazione di un circuito Tor composto da due Onion Router per contattare una pagina web* [28]

La Figura 2.4 mostra i passi che servono per poter realizzare la creazione di un nuovo circuito. In questo caso Alice desidera visitare una pagina web mediante la rete di anonimizzazione Tor, ma per farlo ha prima bisogno di costruire un percorso per i suoi pacchetti. È l'Onion Proxy (OP) che si occupa di contattare gli Onion Router (OR) presenti nella rete (in questo caso si è deciso di creare un circuito composto da due OR, solitamente se ne selezionano almeno tre) e di negoziare con essi una chiave simmetrica. Per poter conoscere gli OR della rete, l'OP contatta uno dei server di directory per ottenere una lista degli OR attivi. L'obiettivo dell'OP di Alice è quello di costruire un circuito composto da due OR e di negoziare due chiavi simmetriche, una col primo OR ed una col secondo, che verranno utilizzate per crittare con due livelli di sicurezza (due strati della cipolla) il suo messaggio. Il primo passo compiuto dall'OP è quello di scegliere l'*exit node* del percorso, in questo caso OR2. Una volta

fatto ciò, può iniziare a costruire il percorso inviando una cella di controllo con il comando *create* ad OR1, che creerà il *circID* del nuovo percorso. Il *payload* di tale cella contiene la prima metà dell'handshake di Diffie-Hellman (g^x) [36] crittato con la chiave di OR1. OR1 invia come risposta una cella di controllo con il comando *created*, il cui *payload* contiene la seconda metà dell'handshake di Diffie-Hellman (g^y) e l'hash della chiave negoziata ($K = g^{xy}$). A questo punto l'OP ha stabilito che il primo OR del percorso è OR1 e adesso deve contattare l'OR successivo. Per farlo invia una cella di relay con il comando *extend* ad OR1, il quale si incaricherà di mandare a OR2 la cella di controllo con il comando *create*, che, seguendo lo stesso procedimento di prima, permetterà all'OP di aggiungere OR2 al percorso. Il percorso è stato creato, pertanto l'OP si incaricherà di stabilire la connessione con la pagina web desiderata passando attraverso i due nodi intermedi (inviando celle di relay con comando *begin*). Nel momento in cui l'OP riceverà la cella di relay con comando *connected* significa che il circuito è pronto per essere utilizzato per inviare i dati mediante il comando *data*.

2.1.5 Hidden service della rete Tor

Così come Tor ha l'obiettivo di garantire l'anonimato dei client connessi alla sua rete, è altresì vero che i client stessi potrebbero voler offrire dei servizi anonimi ad altri utenti della rete. Si pensi ad esempio alla pubblicazione di articoli su un sito web, oppure ad un servizio di messaggistica istantanea [37], realizzati in modo tale che i servizi offerti non espongano un indirizzo IP ma rimangano nell'anonimato. Tor permette di poter creare questi servizi detti *hidden service*, ovvero servizi anonimi di cui non è possibile venire a conoscenza dell'indirizzo IP ma che sono identificati mediante un indirizzo fittizio conosciuto come *onion address* (come ad esempio l'indirizzo ajschweiubcwiecw.onion). Uno degli obiettivi relativi a questa tipologia di servizi riguarda la difesa nei confronti degli attacchi informatici di tipo Distributed Denial of Service (DDoS) [38], che si basano sulla conoscenza dell'indirizzo IP del servizio per inondarlo di richieste causandone disagi. In questo caso, non essendo disponibile l'indirizzo del servizio offerto, eventuali attaccanti dovrebbero per forza attaccare tutti i circuiti della rete Tor e quindi gli OR, rendendo più difficile scalfire il servizio offerto. Di seguito analizzerò le fasi relative alla creazione e alla divulgazione di un *hidden service*, che sarà importante comprendere a fondo dal momento in cui tali servizi sono stati largamente utilizzati all'interno del prototipo.

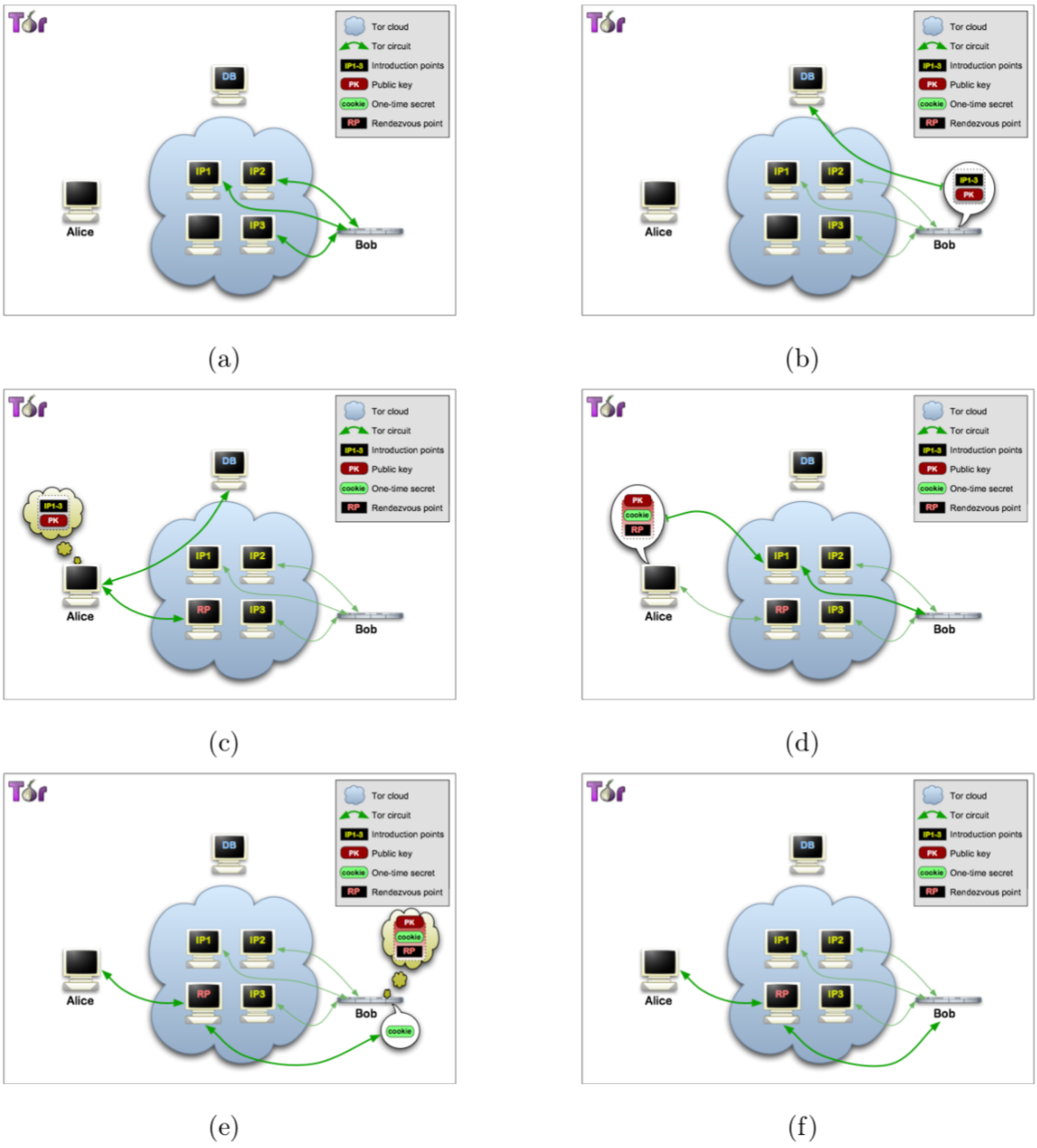

Figura 2.5: *Creazione di un hidden service nella rete Tor* [37]

La prima operazione da portare a termine riguarda l'*hidden service*, nella Figura 2.5 (a) indicato come Bob: dopo aver creato una coppia di chiavi a lungo termine per identificare il suo servizio, deve creare alcuni circuiti verso diversi OR all'interno della rete, noti come *introduction point* (IP). Fatto ciò, Bob può creare un descrittore di servizio nascosto, composto dalla sua chiave pubblica e da una lista degli IP selezionati nella fase precedente, per poi firmarlo con la propria chiave privata e caricarlo su una tabella hash distribuita come si evince dalla Figura 2.5 (b). A questo punto un OP, nel nostro caso Alice (Figura 2.5 (c)), è interessato ad usufruire del servizio offerto da Bob, magari dopo aver trovato il suo *onion address* su un motore di ricerca. Esso richiede informazioni su tale servizio al database e poi sceglie un *rendezvous point* (RP) che rappresenta il nodo utilizzato per connettersi a Bob. Il punto di incontro è pronto, il servizio nascosto è attivo ed il descrittore è presente: Alice deve preparare una lettera di presentazione (crittografata con la chiave pubblica di Bob), composta da un *one-time-secret*, ovvero una password valida per un unico utilizzo [39], e l'indirizzo del RP. Alice contatta uno degli IP di Bob e chiede di consegnargli il messaggio di presentazione (Figura 2.5 (d)). Il messaggio contiene anche la prima parte dell'handshake Diffie-Hellman, come descritto nella sezione precedente relativa alla creazione di un nuovo circuito. Bob decodifica il messaggio introduttivo di Alice: ora conosce l'indirizzo del RP selezionato da Alice e crea un circuito partendo da esso; successivamente invia un messaggio di appuntamento che include la seconda metà dell'handshake Diffie-Hellman ed un *one-time-secret* (Figura 2.5 (e)). A questo punto l'ultimo passaggio è mostrato nella Figura 2.5 (f): l'RP collega Alice a Bob, ma non è in grado di identificare né il mittente né il destinatario, né le informazioni trasmesse attraverso il circuito. Alice inizia la comunicazione con l'invio della cella relay con il classico comando *begin* lungo il circuito.

2.2 Invisible Internet Project - I2P

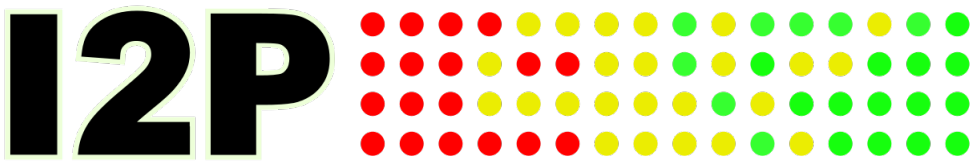

Figura 2.6: *Logo del progetto I2P* [40]

La seconda rete di anonimizzazione più utilizzata dopo Tor è I2P. Si tratta di un progetto che nacque nel corso del 2003 [41] e da allora gli sviluppatori stanno periodicamente rilasciando nuove funzionalità per riuscire a migliorare e rendere usabile il proprio software. Molto spesso questa rete viene messa a confronto con la più famosa ed utilizzata rete Tor e nella sezione successiva ne valuterò le analogie e le differenze. In questa sezione sottolineerò invece le caratteristiche di I2P, concentrandomi in particolar modo sui componenti della rete, sulla costruzione dei percorsi e sui servizi nascosti che vengono offerti agli utenti. Così come Tor, comunque, anche la rete I2P nel corso degli ultimi anni ha conosciuto un incremento dei suoi utilizzatori, con quasi 26.000 router che giornalmente si scambiano informazioni da ogni parte del mondo [42].

2.2.1 Caratteristiche di I2P

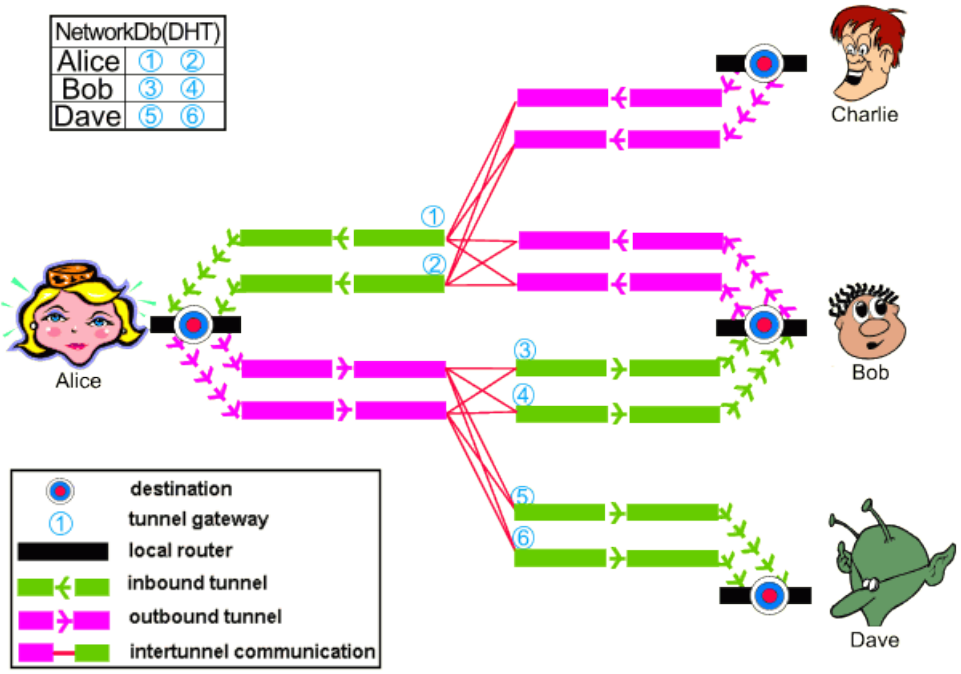

Figura 2.7: *Topologia della rete I2P con utenti che scambiano messaggi* [43]

I2P è una rete anonima a bassa latenza orientata ai messaggi e basata sul P2P [44]. All'interno della rete I2P è disponibile una vasta gamma di applicazioni come hosting di siti web, navigazione web, condivisione di file e servizio email: tutte queste applicazioni vengono gestite ovviamente in maniera anonima, così come l'interazione degli utenti all'interno di questa rete. Per poter entrare a far parte del mondo I2P e sfruttare tutte le sue potenzialità, l'utente deve eseguire sulla propria macchina un router I2P, che rappresenta la parte principale del software. Tutti i messaggi inviati vengono inoltrati attraverso specifici tunnel creati da ciascun router I2P; ogni tunnel è composto da un insieme di peer, scelti sulla base di specifiche caratteristiche che analizzerò in seguito. Una volta dunque che l'utente ha avviato il proprio router I2P e la creazione dei tunnel è avvenuta, è pronto per poter comunicare in maniera anonima con gli altri peer della rete. Per potersi far conoscere all'interno della rete I2P, l'utente può pubblicare le proprie informazioni all'interno di un database globale chiamato *netDB*, che conterrà, oltre alle informazioni relative ai singoli peer, anche quelle che riguardano i servizi offerti dalla rete. I messaggi inviati attraverso la rete I2P sono crittografati end-to-end mediante una crittografia *garlic*, ovvero ad aglio [45]. Tale crittografia è molto simile a quella usata in Tor, con la differenza che più messaggi di dati (anche per destinatari diversi) possono essere inseriti in un singolo messaggio contenitore. La

Figura 2.7 mostra la topologia di una piccola rete I2P che comprende quattro utenti. Nel prossimo paragrafo analizzerò in maniera più specifica la distinzione tra *inbound* e *outbound tunnel*; per il momento basti notare che un utente, per poter inviare un messaggio, ha bisogno di conoscere la destinazione dell'utente che vuole contattare e che deve disporre di un tunnel in ingresso e di un tunnel in uscita.

2.2.2 Componenti della rete I2P

Dopo aver descritto in maniera generale come funziona la rete I2P, è giunto il momento di analizzare nel dettaglio tutti i componenti che contribuiscono alla realizzazione di questa rete di anonimizzazione. Di seguito elencherò i principali componenti della rete I2P e una loro descrizione che permetterà di comprenderne meglio il funzionamento.

- **Peer**. L'elemento base della rete I2P è il peer. La rete è composta da più peer, ognuno dei quali può assumere un ruolo diverso sulla base di certe caratteristiche. In generale ogni peer manterrà in memoria una coppia di chiavi, una pubblica ed una privata, che servirà per la comunicazione con gli altri peer.

  - *Floodfill peer*. Questi peer sono i cosiddetti super peer e sono utilizzati per costruire e mantenere aggiornato il database *netDB*. Una delle loro principali responsabilità è quella di archiviare le informazioni sui peer e sui servizi nascosti nella rete in modo decentralizzato, utilizzando le chiavi di indicizzazione (ovvero le chiavi di routing). Nell'attuale progetto I2P ci sono due modi per diventare un *floodfill peer*. La prima opzione è abilitare manualmente la modalità *floodfill* dalla console del router I2P. L'altra possibilità è che un router a banda larga possa diventare automaticamente un *floodfill peer* dopo aver superato diversi test quali stabilità, tempo di attività nella rete e velocità di trasmissione della coda dei messaggi in uscita.
  - *Non-floodfill peer*. Questi peer non hanno la possibilità di gestire il database *netDB*, ma semplicemente svolgono il proprio ruolo venendo inseriti all'interno di specifici tunnel per poter far dialogare i client della rete.

- **Router I2P**. La rete I2P è formata da peer (detti anche client, nodi o router) che eseguono il software I2P, consentendo alle applicazioni di comunicare attraverso la rete I2P. La parte fondamentale di ogni peer in esecuzione è il router I2P. Esso è il responsabile del mantenimento delle statistiche del peer a lui associato, necessarie per la selezione dei peer, l'esecuzione di operazioni crittografiche, la costruzione di tunnel, la fornitura di servizi e l'inoltro di messaggi. Le applicazioni dipendono fortemente dai tunnel creati dal router I2P per rimanere anonimi.

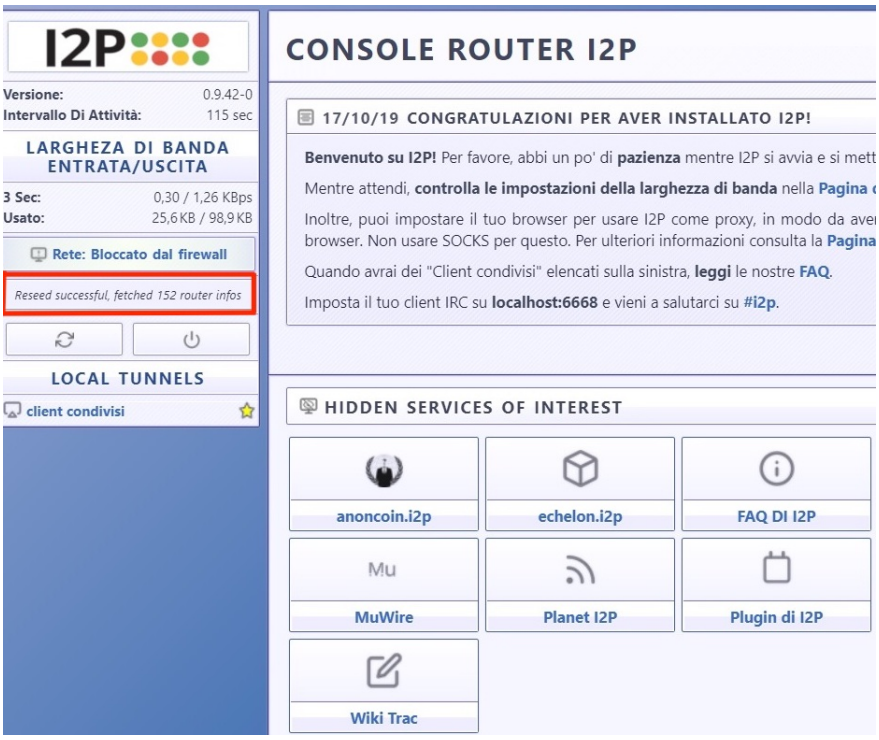

Figura 2.8: *Fase di reseeding nella rete I2P*

- **NetDB**. Si tratta di un database globale basato su una tabella hash distribuita e contiene tutte le informazioni note sulla rete I2P, quindi tutti i peer e servizi I2P attivi. Ogni *floodfill peer* è responsabile solo delle informazioni di una specifica parte della rete. La metrica della distanza di Kademlia XOR [46] viene utilizzata per determinare quale parte della rete debba essere sotto il controllo di uno specifico *floodfill peer*. I peer con una larghezza di banda sufficiente possono essere promossi come *floodfill peer* se la quantità di tali peer disponibili scende al di sotto di una certa soglia. Il *netDB* memorizza due tipi di dati: una struttura *routerInfo* che descrive un peer e un *leaseSet* per ciascun servizio noto.

  – *RouterInfo*. Tutti i peer sono identificati da una struttura dati chiamata *routerInfo*, contenente tutte le informazioni importanti sul peer (indirizzo IP, porta, identificativo del peer, numero di versione stabile I2P, capacità di trasporto e alcuni dati statistici), la sua chiave pubblica e un identificatore di hash a 256 bit. Per recuperare un elenco iniziale di peer disponibili, è possibile scaricare un elenco di *routerInfo* da un server Web non anonimo e noto, oppure sarà il router I2P a scaricarne uno non appena il sistema verrà messo in esecuzione, come mostrato nella Figura 2.8. Il recupero dell'elenco iniziale è chiamato *reseeding* [47].

  – *LeaseSet*. Viene utilizzato per archiviare informazioni su come contattare un servizio I2P. Il *leaseSet* specifica un insieme di punti di ingresso, detti *leasing*, che dovranno essere contattati da chi fosse interessato ad accedere

a quel servizio anonimo. Per pubblicare un servizio, il proprietario deve inviare un *DatabaseStoreMessage* (DSM) a diversi *floodfill peer*, che inglobe-ranno dunque il suo *leaseSet* [48]. A quel punto la distribuzione del servizio è avvenuta ed un eventuale peer interessato ad accedervi dovrà inviare un *DatabaseLookupMessage* (DLM) ai *floodfill peer* che sono in possesso di quel-l'informazione. Nel caso della memorizzazione del *leaseSet*, il *floodfill peer* distribuirà il *leaseSet* ricevuto ai sette *floodfill peer* più vicini. Nel caso in-vece in cui un peer richieda il *leaseSet* di uno specifico servizio, verranno contattati i due *floodfill peer* più vicini che, qualora non fossero in possess-so di quell'informazione, risponderebbero al mittente con un elenco di altri peer che potrebbero contenere il *leaseSet* richiesto.

- **Destinazione**. Tutti i peer e i servizi all'interno della rete I2P sono identificati per mezzo di una destinazione. Tutte le destinazioni nella rete I2P sono identificate da una chiave crittografica da 516 byte, costituita da una chiave pubblica da 256 byte, la firma della chiave pubblica con la propria chiave privata da 128 byte e un certificato attualmente non utilizzato. Una destinazione in I2P si riferisce ad un servizio interno fornito da un router I2P o ad un peer presente all'interno della rete. Le destinazioni vengono utilizzate nella forma codificata in Base64 [49] o nella forma codificata in Base32 [50].

- **Tunnel**. L'ultimo componente della rete I2P è anche uno dei più importanti. Tutti i messaggi infatti vengono trasmessi attraverso i cosiddetti tunnel. Un tunnel è una connessione virtuale crittografata unidirezionale che utilizza in genere due, tre oppure quattro peer. A differenza di Tor, anche il router I2P che cerca di creare un tunnel fa parte del tunnel stesso. All'avvio ogni router I2P crea alcuni tunnel per il traffico in entrata, chiamati *inbound tunnel* e alcuni per il traffico in uscita, chiamati *outbound tunnel*. Il primo peer di un tunnel è chiamato *gateway tunnel* mentre l'ultimo è detto *endpoint tunnel*. Per i tunnel in uscita, il router I2P che ha istituito il tunnel è sempre il *gateway tunnel*. Per i tunnel in entrata, il router I2P che ha istituito il tunnel è sempre l'*endpoint tunnel*. La quantità e la lunghezza predefinite dei tunnel possono essere specificate dall'utente nelle impostazioni I2P. La lunghezza di un tunnel è un compromesso tra prestazioni e anonimato. I tunnel più lunghi aumentano l'anonimato, mentre diminuiscono le prestazioni e viceversa. Un'applicazione non è associata ad un tunnel specifico e può utilizzare tunnel diversi per inoltrare i messaggi. I messaggi crittogra-fati inviati da un peer attraverseranno prima il proprio *outbound tunnel* e poi l'*inbound tunnel* del destinatario. Per questo motivo ogni peer deve mantenere sia un tunnel in ingresso che un tunnel in uscita, non essendo tali tunnel bidire-

zionali. Esistono poi due ulteriori tipologie di tunnel: l'*exploratory tunnel* ed il *client tunnel*.

- *Exploratory tunnel*. Si tratta di tunnel a bassa larghezza di banda e non utilizzati per operazioni sensibili alla privacy. Un router utilizza questo tunnel per contattare i *floodfill peer* e recuperare infirmazioni dal *netDB*. Gli *exploratory tunnel* sono anche usati per costruire, gestire e distruggere altri tunnel.
- *Client tunnel*. Questi tunnel vengono utilizzati per inoltrare i messaggi dell'applicazione e recuperare i *leaseSet*, pertanto sono tunnel ad alta larghezza di banda. Questi tunnel hanno una durata massima di dieci minuti, dopodiché il tunnel viene distrutto e ne viene utilizzato uno nuovo. La ricostruzione costante dei tunnel cerca di prevenire attacchi di analisi del traffico.

2.2.3 Tunnel della rete I2P

Come introdotto in precedenza, nel momento in cui un peer entra all'interno della rete, ha bisogno di costruire alcuni tunnel per poter comunicare con gli altri peer. Come avviene però la creazione di questi tunnel? Quali parametri vengono presi in considerazione per selezionare un peer presente nella rete I2P piuttosto che un altro?

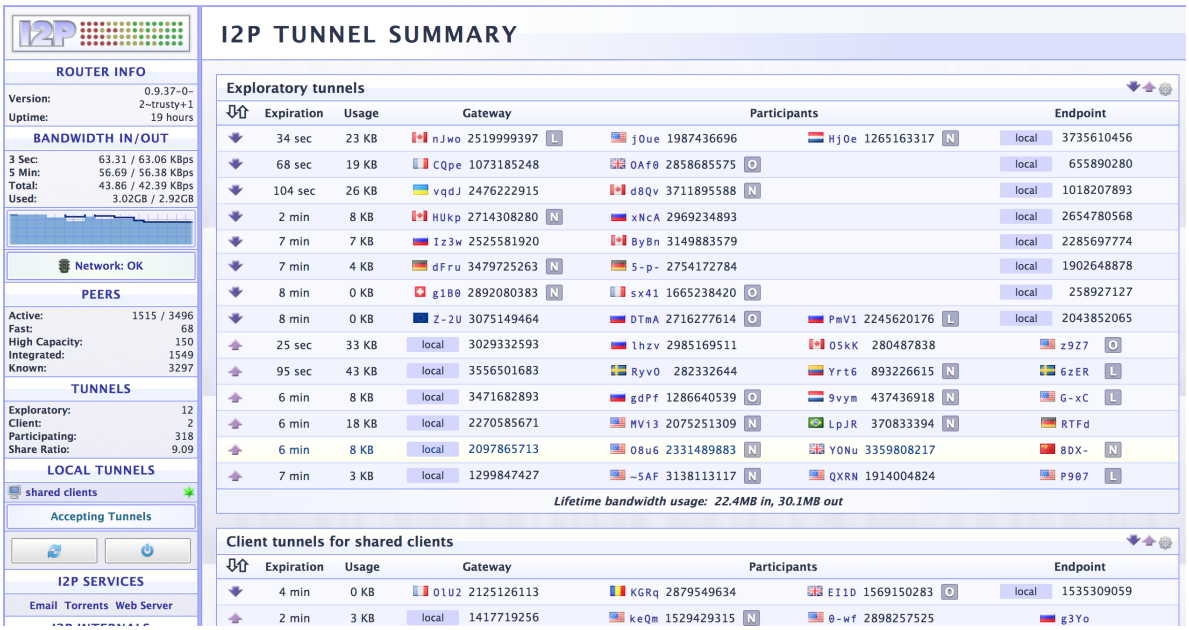

Figura 2.9: *Dashboard di I2P che mostra i tunnel attivi*

Nel momento in cui un peer si connette alla rete I2P, non conosce nessun altro peer, pertanto ha bisogno di contattare il *netDB* per poterne conoscere altri. Dopo aver ottenuto un elenco di peer, la costruzione di un nuovo tunnel viene effettuata

selezionando un set ordinato di tali peer. La scelta dei peer che formeranno il tunnel viene fatta mediante una selezione di peer basata sulla profilazione degli stessi che, sulla base di certe caratteristiche, vengono classificati in livelli. La profilazione dei peer viene eseguita dal router I2P, che tiene traccia delle varie statistiche sulle prestazioni di altri peer e mantiene un database locale contenente tali statistiche, chiamate profili. Tuttavia, non vengono utilizzati sondaggi o altre azioni che potrebbero generare ulteriore traffico. Ogni trenta secondi tutti i profili vengono ordinati in quattro livelli, presentati di seguito.

- **Not-failing**. Tutti i peer conosciuti dal peer in esecuzione. Solitamente il numero di questi peer varia da 300 a 500.
- **Well-integrated**. Peer che affermano di conoscere molti altri peer all'interno della rete.
- **High-capacity**. Peer che con ogni probabilità accetteranno di far parte di un tunnel I2P. Questa previsione è data dall'analisi di quante volte ogni peer abbia accettato o meno la proposta di far parte di un tunnel. Solitamente si tratta di 15-30 peer all'interno della propria rete.
- **Fast**. Peer che fanno parte del livello precedente e che in più possono contare su larghezza di banda elevata. Solitamente si tratta di 8-15 peer della propria rete.

Un *exploratory tunnel* viene utilizzato per inviare tante richieste di costruzione a quelli che il peer boss (colui che richiede la costruzione dei tunnel) ritiene i migliori router (solitamente peer di livello *fast* o, in mancanza di essi, di livello *high-capacity*) fino a quando non si è raggiunto il numero prestabilito. Solitamente sia l'*inbound tunnel* che l'*outbound tunnel* sono composti da tre peer, soluzione che rappresenta un buon compromesso per garantire l'anonimato e una latenza ridotta. La Figura 2.9 rappresenta la dashboard principale relativa alla gestione dei tunnel I2P e mostra la composizione dei tunnel: alcuni sono composti da quattro peer, mentre altri da tre peer, che come detto rappresenta la soluzione più efficiente. Ogni peer, una volta inserito all'interno di un tunnel, mantiene in memoria due informazioni quali la chiave simmetrica negoziata con il peer boss, utilizzata per crittare e decrittare il proprio strato del messaggio, e l'indirizzo del router successivo. I peer che vengono contattati per far parte di uno specifico tunnel sono liberi di rifiutare oppure accettare la richiesta di partecipare alla composizione del tunnel. Un tunnel già stabilito può comunque fallire in qualsiasi momento se, ad esempio, il peer non è in grado di gestire il traffico o esce dalla rete (passa offline). Quando il peer boss è riuscito a costruire l'*inbound tunnel* e l'*outbound tunnel*, è pronto per poter comunicare con altri peer all'interno della rete I2P. A questo punto è interessante valutare la tipologia di crittografia utilizzata all'interno di questi tunnel.

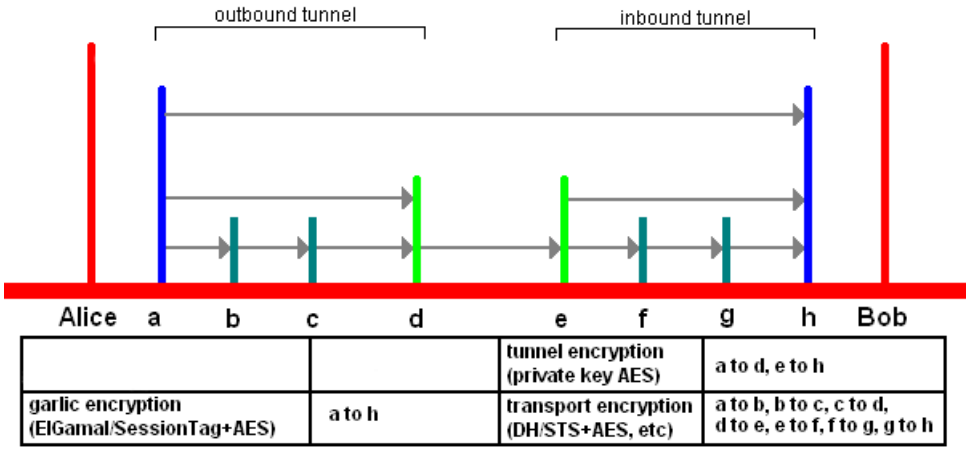

Figura 2.10: *Crittografia utilizzata in una comunicazione I2P* [51]

I2P utilizza il cosiddetto *garlic routing*, una variante dell'*onion routing* utilizzato da Tor. Questa particolare tipologia di routing sfrutta un unico messaggio che al suo interno può contenere più messaggi per più destinazioni diverse [52]. I messaggi di dati effettivi sono crittografati end-to-end con la chiave pubblica del destinatario, mentre il messaggio contenitore viene crittografato tante volte, quante sono i peer del tunnel creato, utilizzando le chiavi simmetriche negoziate con i componenti del tunnel stesso. Durante l'attraversamento dell'*outbound tunnel* ogni peer rimuove il proprio strato di crittografia fino a quando il messaggio contenitore non ha raggiunto l'*endpoint tunnel* dell'*outbound tunnel*. L'*endpoint tunnel* in uscita inoltra, a quel punto, ciascun messaggio al *gateway tunnel* del tunnel in entrata delle diverse destinazioni. Il *gateway tunnel* in entrata di ciascuna destinazione inoltrerà il messaggio ai peer successivi mentre ogni peer che partecipa al tunnel aggiungerà uno strato di crittografia (usando le chiavi simmetriche negoziate in precedenza). Solo il destinatario finale sarà dunque in grado di rimuovere tutti i livelli di crittografia del messaggio contenitore apposti dai peer del tunnel, nonché il livello finale di crittografia end-to-end del messaggio dati. La Figura 2.10 mostra lo scambio di messaggi tra Alice e Bob, due peer appartenenti alla rete I2P. In questo caso è Alice che spedisce un messaggio a Bob e come si può notare entrambi hanno deciso di costruire i propri tunnel con quattro peer. Quando Alice spedisce il proprio messaggio, lo ha già crittato sia con la chiave pubblica di Bob sia con le quattro chiavi simmetriche dei peer del tunnel negoziate all'atto della creazione del tunnel stesso. Man mano che il messaggio transita attraverso questi peer, ognuno di essi, mediante la propria chiave, sarà in grado di rimuovere uno strato e di inoltrare il messaggio al peer successivo. In questo caso non appena il messaggio raggiungerà il peer identificato nella foto come (d), ovvero l'*endpoint tunnel* dell'*outbound tunnel*, sarà protetto solamente dalla chiave pubblica di Bob. A quel punto l'*endpoint tunnel* invierà il messaggio al *gateway tunnel* dell'*inbound tunnel* di Bob, che critterà con la propria chiave tale messaggio, per poi inoltrarlo al peer successivo. Ogni peer all'interno del

tunnel dunque critterà il messaggio con la propria chiave, fino a quando l'*endpoint tunnel* dell'*inbound tunnel* non consegnerà il messaggio a Bob. Adesso il peer Bob sarà in grado di decrittare da una parte con le chiavi simmetriche gli strati relativi ai peer del proprio *inbound tunnel*, dall'altra con la propria chiave privata l'ultimo strato che era stato apposto da Alice dopo averlo firmato con la chiave pubblica del destinatario.

2.2.4 Eepsite della rete I2P

Anche la rete di anonimizzazione I2P, esattamente come Tor, permette la creazione di servizi anonimi detti *eepsite* [53]. Questi servizi possono essere realizzati e messi in esecuzione da qualunque peer della rete, cosicché possa poi renderlo disponibile a tutti gli altri utenti. Per poter far sì che i peer vengano a conoscenza di questi servizi, I2P offre la possibilità di scegliere un nome per il proprio *eepsite* che sia umanamente comprensibile e di pubblicare tale servizio all'interno del *leaseSet* o di specifici siti web che si occupano di pubblicizzare questi servizi anonimi. L'utente ha comunque la possibilità di identificare il proprio servizio mediante una destinazione in Base32, per quanto sia complessa da ricordare per l'uomo data la sua lunghezza (un esempio: iieqahqspvp3lh45i5h6xkgn27fkdnda364ou3zets72shehbysq.b32.i2p). Così facendo tutti gli utenti interessati potranno o contattare il *netDB* per ottenere queste informazioni oppure rivolgersi, come detto, ai siti web specializzati nella pubblicazione degli *eepsite*. La creazione del proprio *eepsite* è molto semplice, in quanto basterà collegarsi alla dashboard di I2P e scegliere la procedura guidata per poter creare un nuovo servizio anonimo.

2.3 Confronto tra Tor e I2P

Dopo aver analizzato i dettagli tecnici relativi alle due reti di anonimizzazione utilizzate all'interno del prototipo, è giunto ora il momento di fare una valutazione per quanto riguarda analogie e differenze. Al momento non prenderò in considerazione le valutazioni che riguardano le performance, poiché verranno trattate più approfonditamente nei capitoli successivi. Prima di procedere però è importante conoscere la terminologia di queste due reti, comparate all'interno della Tabella 2.1.

Tor	**I2P**
Cell	Message
Client	Router or Client
Circuit	Tunnel
Directory	NetDB
Directory Server	Floodfill Router
Entry Guards	Fast Peer
Entry Node	Inproxy
Exit Node	Outproxy
Hidden Service	Eepsite or Destination
Hidden Service Descriptor	LeaseSet
Introduction Point	Inbound Gateway
Node	Router
Onion Proxy	I2P Tunnel Client *
Relay	Router
Rendezvous Point	Inbound Gateway + Outbound Endpoint *
Router Descriptor	RouterInfo
Server	Router

Tabella 2.1: *Confronto tra le terminologie di Tor e I2P [54]. I termini contrassegnati dal simbolo '*' non corrispondono esattamente ma hanno un concetto simile.*

Analizzando entrambe le reti, emergono evidenti differenze in diversi ambiti. Mentre Tor si affida ai server forniti dai volontari per costruire circuiti, I2P utilizza peer con caratteristiche prestazionali elevate che partecipano alla rete. Inoltre, Tor è ottimizzato e progettato per garantire il traffico anonimo nella rete Internet mediante numerosi router di uscita, mentre I2P è progettato per fornire servizi all'interno della propria rete e presenta solo un piccolo insieme di *outproxy*. Tuttavia, entrambe le reti hanno come obiettivo quello di fornire un forte anonimato a bassa latenza. Di seguito l'analisi relativa alla comparazione tra Tor e I2P.

Disponibilità delle applicazioni

Sia Tor che I2P dispongono di una vasta gamma di applicazioni. Molte delle applicazioni I2P sono realizzate esclusivamente per accedere ai servizi all'interno della rete I2P, con alcune eccezioni, come ad esempio Susimail [55], che è in grado di inviare e ricevere mail dalla rete Internet. Le applicazioni Tor invece, grazie al fatto che questa rete utilizzi l'interfaccia SOCKS [30], possono essere utilizzate ovunque possa essere configurato un proxy SOCKS.

Sicurezza dei messaggi e anonimato

I pacchetti inviati attraverso queste reti seguono il *garlic routing* nel caso dell'I2P e l'*onion routing* nel caso di Tor, con le differenze dei due protocolli già analizzate in precedenza. La connessione dell'utente al tunnel/circuito è sempre crittata e pertanto la comunicazione all'interno di queste reti sarà più lenta rispetto a quella della rete Internet (per questi dettagli si veda il capitolo relativo alle performance). Finché interagiscono all'interno della rete, i pacchetti I2P sono crittati end-to-end, mentre nel caso di Tor tale crittografia non può essere garantita poiché dipende dal protocollo del livello di trasporto utilizzato. I protocolli non sicuri non devono quindi essere utilizzati, poiché un *exit node* corrotto potrebbe essere impersonato da un malintenzionato [56]. In Tor solo il primo OR di un circuito conosce l'indirizzo IP dell'utente reale mentre tutti gli OR successivi conoscono solo chi li precede e chi li segue. L'ultimo OR nel circuito conosce invece esclusivamente il destinatario finale. Nel caso di I2P anche il primo peer non conosce il mittente: non lo identificherebbe infatti nemmeno se fosse contattato dal mittente stesso, poiché anche chi invia il messaggio fa parte del tunnel realizzato.

Scalabilità

L'aumento del numero di client che partecipa alla rete anonima influenza sia Tor che I2P. Sebbene il set di utenti anonimi diventi più grande e quindi possa essere presente un anonimato più forte, aumentando il traffico di rete potrebbero verificarsi problemi come la congestione. Nel caso di Tor questo significa che potrebbe essere necessario aumentare la quantità di router utilizzati per costruire circuiti. Questo problema potrebbe peggiorare ulteriormente a causa del fatto che solo un piccolo sottoinsieme di tutti gli OR viene utilizzato come *entry guard* ed *exit node*. Questo potrebbe, a seconda della quantità di nuovi client che si uniscono alla rete, portare a problemi di congestione facendo aumentare la latenza e diminuire la larghezza di banda disponibile. Per quanto riguarda I2P invece i nuovi peer che si uniscono alla rete potrebbero anche diventare protagonisti nella costruzione dei tunnel, supponendo che forniscano

capacità e larghezza di banda sufficienti. Pertanto, è molto meno probabile che nella rete I2P si verifichino problemi di congestione rispetto alla rete Tor.

Centralizzazione

La rete Tor non è completamente distribuita come invece quella I2P. Le informazioni relative ai nodi della rete e agli *hidden service* sono fornite da directory autorizzate che si trovano negli Stati Uniti e in Europa. Queste directory tengono traccia dei cambiamenti nella rete e distribuiscono queste informazioni a tutti i nodi che ne fanno richiesta. In I2P invece tale centralizzazione non esiste, dal momento in cui ciascun relay partecipante mantiene localmente un elenco di tutti i peer noti dopo aver contattato il *netDB*, il database distribuito che fornisce informazioni sui peer della rete.

Routing e selezione dei nodi

Una delle caratteristiche più importanti di queste due reti è la costruzione del circuito/tunnel all'interno del quale dovranno transitare i pacchetti. Tor effettua una netta distinzione classificando i diversi nodi della rete come *entry guard*, *entry node* ed *exit node*. Per poter effettuare tale classificazione utilizza meccanismi di sondaggio relativi alla larghezza di banda per misurare le prestazioni di ogni OR, per poi scegliere quelli migliori. Questo comporta il fatto che non tenga in considerazione parametri come la distanza degli OR, ma solamente la larghezza di banda e la sua capacità. Così facendo il rischio è quello di scegliere gli OR migliori per quanto riguarda la larghezza di banda ma in alcuni casi molto distanti tra di loro, con la conseguenza di avere una latenza più elevata rispetto alla selezione di due OR magari meno prestanti ma maggiormente vicini. I2P invece per la selezione dei nodi si affida ai valori delle prestazioni monitorati nel corso del tempo, non avendo dunque bisogno di effettuare sondaggi e di includere ulteriore traffico nella rete come invece fa Tor. Esattamente come Tor però, anche la rete I2P valuta i nodi sulla base della classificazione effettuata, privilegiando i nodi *fast*: questa scelta, come nel caso precedente, potrebbe dunque favorire una latenza più elevata.

Congestion avoidance

Tor utilizza la commutazione di circuito mentre I2P utilizza la commutazione di pacchetto, quindi Tor deve spesso far fronte ad un'elevata congestione che porta ad avere un'elevata latenza [57]. In I2P invece la commutazione di pacchetto porta ad un implicito bilanciamento del carico ed aiuta ad evitare congestioni e interruzioni del servizio, che favorisce i trasferimenti di file di grandi dimensioni.

2.4 Recap di algoritmi e script utilizzati nel prototipo

Dopo aver analizzato con attenzione le due reti di anonimizzazione presenti all'interno del prototipo, è giunto il momento di valutare nel dettaglio il background relativo agli algoritmi e agli script utilizzati in questo progetto. In questa sezione mostrerò dunque le funzionalità degli script e lo pseudocodice degli algoritmi con una breve descrizione per ciascuno, prima di affrontare, nel prossimo capitolo, l'inserimento delle funzionalità relative alla rete I2P.

2.4.1 Algoritmi del prototipo

I tre principali servizi utilizzati all'interno del prototipo sono il Peer Sampling Service, l'Aggregation Service e il Bootstraping Service. Mentre i primi due si servono di algoritmi omonimi, il Bootstraping Service utilizza l'algoritmo TMan. Di seguito una breve analisi di questi algoritmi.

Peer Sampling Service

Il Peer Sampling Service ha come obiettivo quello di creare un grafo tra i calcolatori presenti all'interno della rete, la cui topologia tende ad un *random graph*. Ogni nodo mantiene una *partial view*, ovvero un elenco contenente un sottoinsieme dei peer che partecipano al sistema. Sulla *partial view* agiranno concorrentemente l'*active thread* ed il *passive thread*. Il primo avrà come compito quello di scegliere un peer e contattarlo, mentre il secondo si occuperà di ricevere e gestire i messaggi provenienti da altri peer. Ogni *partial view* è composta da una lista di oggetti `Node Descriptor`, ognuno dei quali identifica uno specifico peer mediante nome, identificativo univoco, indirizzo ed età. Di seguito l'elenco dei parametri richiesti dall'algoritmo.

- *peerSelection*: indica il metodo di selezione dei peer da contattare presenti all'interno della propria *partial view*. Nel prototipo la scelta è ricaduta sulla modalità `rand`, che seleziona casualmente un peer, mentre la seconda possibilità prevede di utilizzare la modalità `tail`, che seleziona l'ultimo peer della *partial view*.

- *c*: indica la dimensione della *partial view*. Sebbene i peer presenti al suo interno mutino costantemente, la dimensione non subisce variazioni.

- *H (healed)*: indica i `Node Descriptor` che vengono spostati in fondo alla *partial view* nelle istruzioni di permutazione. La *partial view* non viene mai ordinata sulla base dell'età dei peer, ma la permutazione evita che gli elementi più vecchi e non attivi (i cosiddetti *churn*) vengano propagati ad altri peer.

- *S (swapped)*: indica i `Node Descriptor` che vengono prelevati dal messaggio in ingresso $buffer_p$ ed inseriti nella propria *partial view*.
- *push-pull*: comandi che indicano le modalità di propagazione della *partial view* tra i peer. Nel caso del comando *push* si invia parte della propria *partial view* senza attendere alcuna risposta, mentre nel caso del comando *push-pull* si dà luogo ad uno scambio reciproco di parte delle *partial view* tra mittente e destinatario.

Algorithm 1 Peer Sampling Service [58]

Active thread (a)

```
1:  loop
2:      wait (T time units)
3:      p ← view.selectPeer()
4:      if push then
5:          buffer ← ((MyAddress, 0))
6:          view.permute()
7:          move oldest H items to view's end
8:          buffer.append(view.head(c/2-1))
9:          send buffer to p
10:     else
11:         send (null) to p
12:     end if
13:     if pull then
14:         receive buffer_p from p
15:         view.select(c, H, S, buffer_p)
16:     end if
17:     view.increaseAge()
18: end loop
```

Passive thread (b)

```
1:  loop
2:      receive buffer_p
3:      if pull then
4:          buffer ← ((MyAddress, 0))
5:          view.permute()
6:          move oldest H items to view's end
7:          buffer.append(view.head(c/2-1))
8:          send buffer to p
9:      end if
10:     view.select(c, H, S, buffer_p)
11:     view.increaseAge()
12: end loop
```

Nel momento in cui l'*active thread* viene messo in esecuzione, crea un messaggio *buffer* al cui interno inserisce il proprio indirizzo e imposta la propria età a 0. Il meccanismo dell'età è molto interessante: ad ogni ciclo, ovvero dopo ogni messaggio ricevuto, i thread incrementano l'età di tutti i componenti della propria *partial view* (righe 17 e 11), mentre, ogni volta che un peer viene contattato da un altro peer, l'età del mittente viene impostata a 0. Nel momento in cui i thread applicano la funzione *permute*, sarà proprio l'età la discriminante per poter escludere alcuni dei peer presenti all'interno della *partial view*. Più un peer è anziano, più è probabile che venga escluso dalla rete. In questo modo ogni peer propaga elementi giovani, mentre quelli non più raggiungibili (i *churn*) vengono via via eliminati. Nel momento in cui viene ricevuto un messaggio *buffer*, esso viene passato alla funzione *select* (righe 15 e 10) che ne aggiunge

il contenuto all'attuale *partial view* e la modifica rimuovendone duplicati ed elementi anziani, mantenendone comunque la dimensione immutata.

Aggregation Service

Algorithm 2 The Gossip-Based Aggregation Algorithm [59]

Active thread (a)

```
1: loop
2:     wait (T time units)
3:     q ← getNeighbour()
4:     msg_p ← (MyDescriptor, s_p)
5:     send msg_p to q
6:     msg_p ← receive(q)
7:     if msg_q != null then
8:         exchange ← epochCheck(MyEpoch, msg_q.epoch)
9:         if exchange == true then
10:            s_p ← update(s_p, msg_q.s_q)
11:            elapsedCycle++
12:        else
13:            synchronizeEpoch(msg_q)
14:        end if
15:    end if
16: end loop
```

Passive thread (b)

```
1: loop
2:     s_q ← receive(*)
3:     exchange ← epochCheck(MyEpoch, msg_q.epoch)
4:     if exchange == true then
5:         msg_p ← (MyDescriptor, s_p)
6:         send msg_p to q
7:         s_p ← update(s_p, s_q)
8:         elapsedCycle++
9:     else
10:        synchronizeEpoch(msg_q)
11:    end if
12: end loop
```

L'obiettivo dell'Aggregation Service è quello di calcolare un valore comune dopo aver applicato una funzione di *update* alle due variabili passate come input da due peer distinti. Il valore di questa variabile locale viene stabilito a priori, nel momento in cui

viene messo in esecuzione il sistema. Questo servizio viene utilizzato per monitorare lo stato della rete e all'interno del prototipo si è scelto di calcolare la media dei valori scambiati da due peer. Il funzionamento è molto simile a quello dell'Algoritmo 1. Anche in questo caso le due componenti principali sono rappresentate dall'*active thread (a)* e dal *passive thread (b)*. Il primo si occupa di selezionare un peer casualmente (riga 3 (a)) e di inviargli il valore della propria variabile locale (riga 5 (a)), mentre il secondo ha il compito di ricevere messaggi in ingresso da parte di altri peer (riga 2 (b)). A questo punto chi ha ricevuto il messaggio invia a sua volta al mittente il proprio valore (riga 6 (b)) per poi applicare la funzione di *update* (riga 7 (b)) passando come parametri il proprio valore e quello appena ricevuto. L'istruzione *update* aggiorna il valore della variabile locale di ogni peer applicando la funzione scelta per l'aggiornamento, che deve essere nota a priori a tutti i nodi della rete. Così facendo entrambi i peer avranno ottenuto lo stesso valore e quest'ultimo sarà pronto per essere scambiato con altri peer presenti nella rete. Nell'Algoritmo 2 è presentata una versione avanzata (la stessa utilizzata nel prototipo) dell'Aggregation Service, che prevede di gestire la sincronizzazione attraverso le epoche. Allo scadere di ogni epoca infatti la variabile s viene nuovamente impostata al valore iniziale. In questo modo si rende l'algoritmo robusto ai guasti ed all'ingresso di nuove macchine.

Bootstraping Service

Algorithm 3 The T-Man Algorithm [60]

Active thread (a)

1: **loop**
2: wait (Δ)
3: $p \leftarrow$ view.selectPeer(ψ, rank($myDescriptor, view$))
4: buffer $\leftarrow$ merge($view, myDescriptor$)
5: $buffer \leftarrow$ rank(p, $buffer$)
6: send first m buffer's entries to p
7: receive $buffer_{\mathrm{p}}$ from p
8: $view \leftarrow$ merge($buffer_{\mathrm{p}}, view$)
9: **end loop**

Passive thread (b)

1: **loop**
2: receive $buffer_{\mathrm{q}}$ from q
3: $buffer \leftarrow$ merge($view, myDescriptor$)
4: $buffer \leftarrow$ rank($q, buffer$)
5: send first m buffer's entries to q
6: $view \leftarrow$ merge($buffer_{\mathrm{q}}, view$)
7: **end loop**

L'Algoritmo 3 rappresenta l'ultimo servizio del prototipo, ovvero il Bootstraping Service. In questo caso si è scelto di utilizzare l'algoritmo TMan [60], un protocollo di bootstraping che, partendo da una subcloud, crea una topologia di rete specifica sulla base della funzione di `rank` scelta. Il suo funzionamento si basa sulla classificazione degli elementi della *partial view* secondo un criterio di preferenza dettato dalla funzione `rank`, che deve essere definita a priori. Di seguito i parametri necessari per l'esecuzione dell'algoritmo.

- Δ: indica la lunghezza di un ciclo (riga 2 (a));
- ψ: indica i nodi selezionati dai vicini coi quali interagire scegliendoli dai ψ elementi con la maggior preferenza (riga 3 (a));
- m: indica il numero massimo di `Node Descriptor` che può essere inserito in un messaggio destinato ad un altro peer (righe 6 (a) e 5 (b));
- *rank(node, view)*: indica la funzione attivata sulla vista in ingresso *view* applicando l'algoritmo prendendo come punto di riferimento il nodo inserito *node*.

Questo algoritmo viene messo in esecuzione dopo la creazione di una specifica subcloud, alla quale parteciperà un insieme di peer. Ogni nodo della subcloud eseguirà il modulo BS e manterrà una *partial view* contenente un sottoinsieme dei nodi della subcloud. Lo scambio continuo e l'ordinamento dei nodi all'interno delle *partial view* mediante la funzione `rank` porta alla creazione di una topologia di rete specifica dipendente dalla funzione. L'*active thread* seleziona un peer tra i peer della sua *partial view* che preferisce di più (in base alla funzione `rank`) e gli invia i primi peer ordinati sulla base delle preferenze espresse dalla funzione. Quindi attende che il nodo destinatario faccia lo stesso e gli invii i suoi primi m peer preferiti. Alla fine entrambi inseriranno i peer ricevuti all'interno della propria *partial view*.

2.4.2 Script del prototipo

Dopo aver analizzato lo pseudocodice degli algoritmi, è importante valutare le caratteristiche degli script messi a disposizione all'interno del prototipo. Tutte le API sono composte da una parte di script e da una parte di algoritmo. La parte di script è quella eseguita all'esterno del cloud e contiene le istruzioni per avviare una comunicazione con un nodo. La parte di algoritmo è ciò che viene eseguito all'interno del nodo (quindi all'interno del cloud) dopo aver ricevuto la comunicazione dello script corrispondente. I nomi degli script sono composti da lettere minuscole separate dal carattere *underscore* mentre i nomi degli algoritmi contengono le stesse lettere dello script corrispondente ma ogni nuova parola inizia con una lettera maiuscola.

run_nodes

Questo script permette di poter creare una subcloud dopo aver contattato un nodo esistente all'interno della rete. I parametri da dover utilizzare riguardano il nome che si vuole assegnare alla subcloud ed il numero di partecipanti, che verranno selezionati casualmente tra quelli presenti all'interno della *partial view* del peer contattato. Ogni peer può appartenere ad una sola subcloud, sia in qualità di partecipante che di capo della subcloud.

RunNodes

Eseguito su un nodo, utilizza l'elenco dei peer locale per ottenere il numero richiesto di nodi liberi e quindi dice loro di unirsi alla subcloud avviando i loro moduli BS.

terminate_nodes

Questo script permette di poter richiedere l'eliminazione di uno o più peer da una specifica subcloud. Per far sì che vada a buon fine, il peer da contattare dovrà essere

il capo di una delle subcloud presenti nella rete e ovviamente i peer che si vogliono eliminare devono fare parte di tale subcloud.

TerminateNodes

Invia un messaggio ai peer da rimuovere dicendo loro che sono stati rimossi. Successivamente invia messaggi ai peer rimanenti dicendo di aggiornare le loro *partial view* dopo la rimozione del peer in questione.

add_new_nodes

Questo script permette di poter aggiungere un peer ad una subcloud esistente. Per far sì che vada a buon fine, il peer da contattare dovrà essere il capo di una delle subcloud presenti nella rete e ovviamente il peer che si vuole aggiungere dovrà essere attivo.

AddNewNodes

Invia un messaggio al nodo da aggiungere, dicendo di unirsi alla subcloud. Successivamente invia messaggi ai nodi appartenenti alla subcloud dicendo di aggiornare le loro *partial view* dopo l'aggiunta del peer in questione.

describe_instances

Questo script permette di poter visualizzare le caratteristiche di una certa subcloud contattando un peer che ne fa parte oppure il capo della subcloud. Verranno mostrate informazioni come il nome della subcloud e tutti i vicini del peer contattato che compongono tale subcloud. Qualora si dovesse contattare un peer che non fa parte di alcuna subcloud, verrebbe mostrato un messaggio d'errore.

DescribeInstances

Si limita a raccogliere le informazioni sulle richieste effettuate dai peer della rete.

monitor_instances

Questo script permette di collegarsi ad uno specifico peer e di avviare il suo sistema di monitoraggio che sfrutta l'Aggregation Service per stimare il numero totale di peer nel cloud. Al momento questo script non funziona in quanto l'algoritmo di Aggregation Service si occupa di calcolare una media tra due valori e non di stimare il numero totale di peer nella rete, pertanto l'esecuzione di tale script porterà sempre ad avere un numero di peer stimato pari a 0.

MonitorInstances

Avvia il *polling* del modulo di Aggregation Service per stimare il numero totale di peer nel cloud.

unmonitor_instances

Questo script rappresenta il duale della *monitor_instances* e permette di poter interrompere il monitoraggio della stima del numero totale di peer nel cloud.

UnmonitorInstances

Rappresenta il duale dell'algoritmo *MonitorInstances*.

Capitolo 3

Implementazione della rete I2P

Dopo aver preso in considerazione tutti gli elementi utili per poter comprendere questo lavoro di tesi, analizzerò ora le funzionalità relative alla rete I2P che ho aggiunto all'interno del prototipo esistente. La scelta di questa rete di anonimizzazione racchiude diverse motivazioni. Prima di tutto ho ritenuto molto interessante implementare tale rete per poterla poi confrontare con le funzionalità della rete Tor, sia dal punto di vista pratico sia dal punto di vista delle performance. Un'altra ragione che mi ha convinto ad affrontare questo lavoro è stata la totale assenza di progetti simili nella panoramica del mondo Internet. Tutte le funzionalità relative alla rete I2P che ho aggiunto, infatti, non sono mai state trattate in relazione ad un sistema cloud P2P anonimo e ciò mi ha portato a maturare una forte motivazione nei confronti di questo argomento. Infine è importante sottolineare come all'interno del prototipo precedente fosse stata proposta la possibilità di sviluppare la rete I2P in aggiornamenti futuri, pertanto ho deciso di approfittare di quest'occasione. L'obiettivo di questo capitolo è dunque quello di spiegare, passo dopo passo, come sia riuscito ad aggiungere le funzionalità della rete I2P. Prima di tutto elencherò le difficoltà riscontrate in fase di progettazione, sottolineando in particolar modo l'incompatibilità di Java Remote Method Invocation (JRMI) [61], la tecnologia utilizzata nel precedente prototipo, con I2P. Successivamente mi addentrerò nei dettagli tecnici di I2P ponendo l'attenzione su come sia stata configurata la rete, sul protocollo utilizzato e sulle funzionalità inserite. Infine valuterò i cambiamenti apportati rispetto alla rete Tor ed indicherò possibili miglioramenti futuri della versione finale del prototipo.

3.1 Incompatibilità con Java RMI e soluzione proposta

Nel momento in cui ho deciso di aggiungere le funzionalità relative alla rete I2P, mi sono domandato se l'architettura presente nel prototipo, utilizzata sia per la gestione della versione non anonima sia per la gestione della versione anonima della rete Tor, fosse compatibile con la rete di anonimizzazione I2P. Dopo qualche settimana di ricerca mi sono però reso conto di come Java RMI, impiegato nella realizzazione del prototipo precedente, non fosse compatibile con I2P. Nello specifico, Java RMI richiede che la connessione anonima ad un nodo presente all'interno della rete avvenga mediante SOCKS [62], ovvero un particolare tipo di proxy che permette di effettuare connessioni TCP tra computer su reti diverse nel caso in cui un instradamento diretto non fosse disponibile [63]. I2P invece, a fronte di un'evoluzione non ancora totalmente completata nei confronti di questa tematica [64], permette di utilizzare SOCKS in maniera piuttosto limitata. Ad esempio non supporta le destinazioni nella forma Base64, rendendo di fatto impossibile la comunicazione tra peer. Inoltre SOCKS in I2P supporta solamente le connessioni in uscita e quindi, se anche fosse possibile contattare un altro peer, non si avrebbe poi la possibilità di ricevere eventuali risposte. Giunto a questo punto mi sono reso conto di dover individuare una soluzione alternativa rispetto all'utilizzo di Java RMI. Non potendo sfruttare la base del prototipo, ho scelto di utilizzare I2P Client Protocol (I2CP) [65], un protocollo scritto in Java che consente lo scambio di messaggi asincrono tra i peer appartenenti alla rete I2P e che mette a disposizione delle API per poter gestire al meglio la ricezione e l'invio degli stessi. Nella sezione 3.3 analizzerò nel dettaglio le funzioni di questo protocollo e di come sia stato utilizzato per replicare il comportamento del prototipo seppur utilizzando una rete di anonimizzazione diversa e soprattutto modalità di comunicazione ed interazione tra peer differenti.

3.2 Configurazione della rete I2P

Prima di analizzare nel dettaglio la soluzione proposta, è interessante capire come configurare la rete I2P per poter replicare il comportamento del prototipo precedente. Per automatizzare l'avviamento della rete, ho modificato lo script che si occupa di far partire il router I2P (*runplain.sh*), aggiungendo un'istruzione che permetta di poter salvare su file il Process Identifier (PID) del router, ovvero l'identificativo univoco assegnato a tale processo. Questo script viene chiamato dallo script principale (*startNode.sh*) solamente se il PID salvato su file non corrisponde ad un processo attualmente attivo. L'utilizzo del PID si è reso necessario in quanto, una volta messo in esecuzione il router I2P, esso verrà mostrato tra i processi attivi con il nome *java*, rendendo di fatto impossibile capire se si tratti del router I2P o di un'altra applicazione scritta con lo stesso linguaggio. Per questo motivo ho scelto di salvare in un file (*pidI2PService.txt*) il PID del router: se il router fosse già in esecuzione sulla macchina, il PID presente all'interno del file di testo risulterebbe essere l'identificativo di un processo già attivo e in quel caso non servirebbe avviare nuovamente il processo. Qualora invece il PID indicato su file non fosse presente tra i processi attivi, allora lo script si occuperebbe di mettere in esecuzione il router I2P.

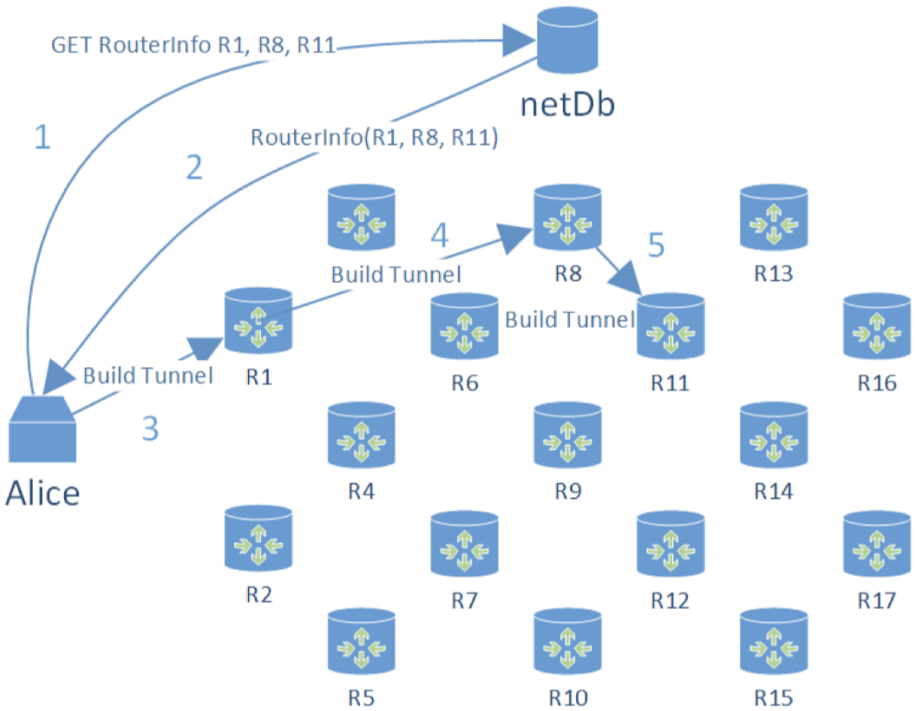

Figura 3.1: *Fase di creazione dell'outbound tunnel* [66]

Dopo aver messo in esecuzione il router, le due fasi successive riguardano operazioni di bootstrap. La prima, come riportato nella Figura 3.1, prevede che il router, una volta

avviato, contatti il *netDB* per poter iniziare a costruire i propri tunnel in entrata ed in uscita. Questa operazione può risultare abbastanza lunga se il router viene avviato per la prima volta, in quanto I2P prevede di valorizzare i router più anziani e con più tempo di utilizzo continuativo all'interno della rete [67]. La seconda operazione invece riguarda esclusivamente il prototipo.

```
GTZj7z1GbpV0G8EcC9uqxYNpZp9Y1p709bJBZopa3SJRd0U3~IuV
GKupKnLidVrHsQ38vXZIx0ZAff-8P1sI69-
jW6lgqaVPz~FkIydtS3YIlcowMEo~yzGuM7urvTXq46fdsrHMCl45xz
Nc0rmLpl0T0zHbWcqFACYLtbjeMnECvCQZsF6V3zSDJBi0Zxpzcdg
WS~MFYijbYHUs9uz5BrmWW41mDG3ZeRSxQpTFHcoqFa~TvmPt
WhxHTOjmYccBqlP1rsCvXPPVpox03FjMNHiZ9e~MC6w3eR2LMsz
1WFx6IrkI5a~lyi4kWCsiOo71oE3utHW29sn2iJQvC36U4RyEiIA18oh
pi23HP9K9JpKfo88JDqF-
qYXIJixIMw7MjBjVHJVEfFA3w1dNr0DveN6id-
gITIb9JT7oFXFTgbTwVeu-YJtx-
0eYlCUaDTmRHbxNv5vhw8BUDCl6bVOq5tgp9AdEhThitMI4mcdtu
O0rOOkUl6VH0d656QYv5uFKBQAEAAcAAA==
```

Figura 3.2: *Esempio di destinazione I2P codificata in Base64*

Per poter iniziare a dialogare con altri peer della rete, è necessario che il peer avviato conosca di default almeno un peer attivo. Per questo motivo, così come succedeva nel prototipo precedente, ho utilizzato un file di testo (*anonymousMachinesI2P.txt*) che contenesse le destinazioni di alcuni peer per permettere al peer in esecuzione di iniziare a contattarli per venire a conoscenza di altre entità attive. Come detto in precedenza, le destinazioni possono essere gestite utilizzando due codifiche diverse. Avendo scelto di utilizzare il protocollo I2CP la codifica necessaria per poter contattare un altro peer mediante la sua destinazione è quella in Base64. Si tratta di una codifica che prevede l'utilizzo di una chiave a 516 byte, come mostrato nella Figura 3.2. Nel momento in cui un peer verrà messo in esecuzione ed inizierà a dialogare con altri peer, tutte le destinazioni mostrate all'utente saranno convertite in Base32. Ho deciso infatti di effettuare tale conversione per motivi pratici, essendo molto complicato distinguere le destinazioni in Base64. Non mi è stato però possibile utilizzare esclusivamente le destinazioni in Base32 convertendole, se necessario, nella forma più estesa, in quanto una destinazione in Base32 è l'hash codificato della destinazione in Base64, motivo per il quale non è permessa la conversione [68].

3.3 Funzionalità aggiunte al prototipo

A questo punto è giunto il momento di scendere nel dettaglio e analizzare le funzionalità inserite all'interno del prototipo relative alla rete di anonimizzazione I2P. Per poter avere un quadro generale della situazione, ho deciso di seguire questo percorso. Prima di tutto mostrerò le caratteristiche del protocollo I2CP e le relative API, valutando poi la creazione di una destinazione e il suo riutilizzo da parte di un peer in momenti successivi a tale generazione e la gestione dell'invio e della ricezione dei messaggi. Infine prenderò in considerazione le differenze tra le entità utilizzate nella versione precedente del prototipo (*active thread* e *passive thread*) e in quella aggiornata, specificando poi eventuali cambiamenti negli algoritmi e negli script proposti.

3.3.1 Protocollo I2CP

Grazie al protocollo I2CP la realizzazione della parte relativa alla comunicazione tra peer non è stata eccessivamente complessa. I2CP, per poter essere utilizzato all'interno di un progetto Java, richiede l'inclusione di tre librerie (già presenti nella cartella in cui è stato installato I2P): `streaming.jar, mstreaming.jar` e `i2p.jar`. Mediante queste librerie sarà possibile accedere a tutte le classi e a tutti i metodi necessari per poter gestire lo scambio di informazioni tra peer all'interno della rete I2P. Per quanto riguarda la fase di creazione di una destinazione, lo sviluppatore può utilizzare la classe `Destination` per gestire la destinazione di un peer. Le classi `I2PClientFactory` e `I2PClient`, invece, se utilizzate in combinazione, permettono la creazione di un peer. A questo punto il peer ha bisogno di connettersi ad una sessione, realizzabile mediante la classe `I2PSession` e il metodo *createSession()* da chiamare sul peer stesso. A questo metodo devono essere passati due argomenti, ovvero il file che contiene la chiave relativa alla destinazione creata in precedenza e una variabile di sistema della classe `Properties`. Una volta fatto ciò, il peer è pronto per potersi connettere ad altri peer inviando e ricevendo informazioni. Il protocollo I2CP prevede un meccanismo asincrono per la ricezione di messaggi: basterà infatti implementare l'interfaccia `I2PSessionListener` nella classe che avrà il compito di gestire tale incarico per far sì che il peer venga notificato di eventuali messaggi in entrata. I messaggi inviati attraverso questo protocollo dovranno necessariamente essere degli array di byte, per cui ho predisposto una funzione privata che permetta di convertire qualunque messaggio in un oggetto di questo tipo.

3.3.2 Creazione e gestione di una destinazione I2P

Per creare una destinazione I2P si può procedere in due modi. Il primo prevede di entrare all'interno della dashboard del router I2P e di configurare manualmente il

proprio servizio. Il secondo invece, come mostrato nel Codice 3.1, è la soluzione che ho adottato all'interno del prototipo.

```
1  String home = System.getProperty("user.home") + "/keys";
2  File dir = new File(home);
3
4  if (!dir.exists()) {
5     dir.mkdir();
6  }
7
8  File keyfile = new File(dir, nodeHostName);
9  I2PClientFactory i2pClientFactory = new
      I2PClientFactory();
10 I2PClient i2pClient = i2pClientFactory.createClient();
11
12 if (!keyfile.exists()) {
13    FileOutputStream os = new FileOutputStream(keyfile);
14    Destination myDest = i2pClient.createDestination(os,
         SigType.EdDSA_SHA512_Ed25519);
15    os.close();
16 }
```

Codice 3.1: *Creazione di una destinazione I2P*

All'atto della creazione di una destinazione, I2P crea un nuovo file denominato **.dat*, che rappresenta la chiave per riconoscere ed eventualmente riutilizzare una specifica destinazione. Per prima cosa dunque è necessario predisporre una cartella nella quale saranno salvate le chiavi che faranno riferimento alle destinazioni create. Qualora la cartella non fosse presente, il sistema si occuperebbe della sua inizializzazione (righe dalla 1 alla 6). Ora bisogna creare il peer e per farlo si utilizza la classe `I2PClientFactory` fornita dal protocollo I2CP (righe 9-10). Al peer deve essere poi associata una destinazione, che dovrà essere recuperata all'interno della cartella se già utilizzata in precedenza oppure creata da zero. Potrebbero verificarsi dunque due situazioni: il nome della chiave passato come parametro in input (*nodeHostName*) quando viene eseguito lo script *startNode.sh* non esiste oppure è presente. Nel primo caso la chiave associata al nome *nodeHostName* non viene trovata nella cartella, pertanto bisogna procedere alla creazione di una nuova destinazione (righe dalla 12 alla 16). Da notare che per poter creare una nuova destinazione sia necessario invocare il metodo *createDestination()* passando due parametri: il primo rappresenta l'oggetto di tipo `FileOutputStream` creato mediante l'oggetto *keyfile*, mentre il secondo indica l'algoritmo di crittografia utilizzato per creare la destinazione in Base64. Nel secondo caso invece la destinazione è già stata inizializzata e non è necessario crearla nuovamente. Così è possibile gestire e riutilizzare le destinazioni affinché ad ogni peer sia associata sempre la stessa. Se così non fosse, sarebbe impossibile per i peer attivi

riuscire a dialogare dopo un'eventuale disconnessione, poiché la destinazione presente nel file *anonymousMachinesI2P.txt* per effettuare il bootstrap non sarebbe più quella associata al peer da contattare. Da una parte se la disconnessione riguarda il peer in esecuzione non potrà più essere contattato dai peer che erano a conoscenza della vecchia destinazione, dall'altra se riguarda invece i peer conosciuti non potranno più essere contattati dal peer in esecuzione avendo mutato la propria destinazione.

3.3.3 Creazione di una sessione I2P

Dopo aver associato al peer creato la propria destinazione, è giunto il momento di creare la sessione alla quale connetterlo. La sessione, se associata ad un peer, permette l'assegnamento dei tunnel creati dal router I2P al peer stesso, che in questo modo potrà iniziare a dialogare con altri peer per mezzo dei propri *inbound* e *outbound tunnel*. Questo processo è mostrato nel Codice 3.2.

```
1 Properties props = (Properties)
     System.getProperties().clone();
2 I2PSession session = i2pClient.createSession(new
     FileInputStream(keyfile), props);
3 Destination myDest = session.getMyDestination();
4
5 try {
6     session.connect();
7 } catch (I2PSessionException e) {
8     myPrintUtils.printConnectionErrors();
9 }
```

Codice 3.2: *Creazione di una sessione I2P*

La classe `I2PSession` richiede che venga chiamato il metodo *createSession()* sul peer passando come parametri la chiave *keyfile* e la variabile *props* (righe 1-2). Per recuperare la destinazione del peer basterà chiamare il metodo *getMyDestination()* sulla sessione (riga 3). A questo punto il peer è pronto ad entrare all'interno della rete I2P: per farlo si chiama sulla sessione il metodo *connect()*. La variabile *session* verrà poi successivamente utilizzata per poter inviare messaggi all'interno della rete I2P mediante il proprio *outbound tunnel*.

3.3.4 Invio e ricezione di messaggi

Dopo aver effettuato la connessione, il peer può iniziare a dialogare con altri peer presenti all'interno della rete. Non mi soffermerò sulla fase di bootstrap, poiché il processo, che prevede l'aggiunta di peer noti alla *partial view* prelevandoli dal file *anonymousMachinesI2P.txt*, è rimasto lo stesso del prototipo precedente. La situazione delineata è la seguente: il peer, dopo essere stato associato ad una destinazione e dopo essersi connesso alla rete, ha conosciuto i primi peer con i quali scambiare informazioni per poter allargare il proprio raggio di conoscenze. Senza entrare per il momento nel dettaglio della tipologia dei servizi del prototipo, è interessante verificare come un peer possa inviare messaggi (Codice 3.3) e come possa riceverli (Codice 3.4).

```
1  PssObject pss = new PssObject(true, buffer, node);
2
3  try {
4     session.sendMessage(
5        createDestination(peerToContacted.getNodeHostName()),
6        createBytesObject(pss)
7     );
8  } catch (I2PSessionException e) {
9     e.printStackTrace();
10 }
```

Codice 3.3: *Invio di un messaggio attraverso la rete I2P*

Per poter inviare un messaggio il peer deve ovviamente passare attraverso la sessione. Nel Codice 3.3 viene creato un oggetto di classe `PssObject`, che al momento non è interessante analizzare. Con il metodo *sendMessage* è possibile inviare un messaggio ad un peer la cui destinazione sia nota, passando come parametri la stessa destinazione (riga 5) e l'oggetto sotto forma di array di byte (riga 6). Come analizzato, dunque, l'invio di un messaggio è un'operazione piuttosto semplice, se non fosse per l'obbligo di dover inviare esclusivamente array di byte. Per poter aggirare questa limitazione, come detto in precedenza, ho creato una funzione (*createByteObjects*) che permette di poter convertire qualunque tipo di oggetto in un array di byte.

```
1  public void messageAvailable(final I2PSession sess,
       final int id, final long size) {
2
3  try {
4      ByteArrayInputStream in = new
           ByteArrayInputStream(sess.receiveMessage(id));
5      ObjectInputStream is = new ObjectInputStream(in);
6      Object response = is.readObject();
7
8      /* ##### PSS SERVICE ##### */
9      if (response instanceof PssObject) {
10         ConcretePartialView responsePss =
               (ConcretePartialView) ((PssObject)
               response).getView();
11         ConcreteNodeDescriptor senderPss = ((PssObject)
               response).getSenderNode();
12         pss.managePss(responsePss, senderPss, response,
               sess);
13     /* ##### AGGREGATION SERVICE ##### */
14     } else if (response instanceof AggregationObject) {
15         AggregationObject responseAgg =
               (AggregationObject) response;
16         agg.manageAggregation(pss.getNode(), responseAgg,
               sess);
17     /* ##### BOOTSTRAPING SERVICE ##### */
18     } else if (response instanceof
           InitializatorSubcloud) {
19         Destination peerToInformatedSubcloud =
               ((InitializatorSubcloud)
               response).getSenderNode();
20         tman.initializatorSubcloud(peerToInformatedSubcloud,
               sess, response, pss.getView(), pss.getNode());
21     } else if (response instanceof SendRequestSubcloud) {
22          tman.sendRequestSubcloud(pss.getView(),
               pss.getNode(), sess);
23     .....
24     } catch (final Exception e) {}
25 }
```

Codice 3.4: *Ricezione di un messaggio attraverso la rete I2P*

La ricezione di un messaggio invece, come mostrato nel Codice 3.4, si è rivelata piuttosto complicata da gestire. Ho creato una nuova classe `ManageReceivingI2P` con il compito di occuparsi della ricezione di tutti i messaggi diretti al peer in maniera asincrona. Tale classe implementa l'interfaccia `I2PSessionListener`, che permette di accedere al metodo *messageAvailable* per poter ricevere i messaggi. Nel momento in cui il peer dovesse essere contattato da un altro peer, il messaggio giungerebbe all'interno di questo metodo, il quale notificherebbe il peer stesso fornendo anche la sessione per poter

eventualmente rispondere al messaggio. Dal momento in cui tutti i messaggi giungono sotto forma di array di byte, ho deciso di assegnarne il contenuto ad una variabile *response* (righe dalla 4 alla 6). Per poter distinguere i messaggi che fanno riferimento ai diversi servizi del prototipo come Peer Sampling Service, Aggregation Service e Bootstraping Service, ho utilizzato la keyword di Java `instanceof` per capire il tipo di messaggio ricevuto e come lo si debba gestire. Una delle problematiche legate a questo tipo di approccio è che il peer mittente, qualora non dovesse ottenere una risposta, non riuscirà a comprenderne il motivo. Potrebbe essere capitato che il messaggio sia andato perso all'interno della rete I2P, oppure che il destinatario non fosse in linea nel momento in cui il messaggio è stato spedito. Questa è forse una delle limitazioni più grandi di questo protocollo, poiché se il peer mittente non ottiene alcuna risposta non potrà tentare di inviare il messaggio fino ad ottenerne una, altrimenti rimarrebbe bloccato ad inviare lo stesso messaggio in attesa di una risposta che potrebbe non arrivare mai, poiché il destinatario potrebbe semplicemente essere offline e non riceverne alcuno. D'altro canto però se il messaggio fosse semplicemente andato perso, basterebbe inviarne un altro, ma non essendoci un meccanismo per poter verificare lo stato di un peer all'interno della rete anche questa strada non può essere percorsa. Allo stato attuale un peer invia il messaggio una volta sola sia che ottenga una risposta sia che non la ottenga. Una delle modifiche future potrebbe prevedere un approccio differente qualora gli sviluppatori di I2CP scegliessero una modalità di gestione alternativa a tale situazione, oppure si potrebbe implementare un protocollo basato su un meccanismo di timeout e Acknowledge (ACK) [69].

3.3.5 Gestione di algoritmi ed entità

La progettazione della configurazione della rete e della comunicazione tra peer è stata impegnativa, così come l'analisi effettuata per poter capire quali parti di codice del precedente prototipo riutilizzare e quali invece riscrivere nella loro interezza. Prima di tutto mi sono concentrato sullo studio delle due entità principali del prototipo, ovvero l'*active thread* ed il *passive thread*. Per le funzionalità della rete I2P sono dovuto intervenire in maniera importante sul *passive thread*, mentre l'*active thread* ha subito poche variazioni relativamente alla gestione della destinazione e della creazione della sessione. I tre *passive thread* sono stati inglobati all'interno della classe `ManageReceivingI2P` analizzata nella sezione precedente. Per poter mantenere il codice quanto più ordinato e leggibile possibile, ho creato per ogni servizio (Peer Sampling Service, Aggregation Service e Bootstraping Service) il relativo *passive thread* per gestire i messaggi in arrivo.

```
1  protected void describeInstancesResponse(final
       String subcloudResponse, final PartialView
       viewResponse) throws DataFormatException {
2      myPrintUtils.printDescribeInfo(subcloudResponse,
           viewResponse);
3      myUtils.stopCommunication();
4  }
5
6  protected void describeInstancesRequest(final
       Destination myDestination, final I2PSession
       sess) throws DataFormatException {
7      int okMessage = 0;
8      myPrintUtils.printRequestDescribe();
9      DescribeInstancesResponse dir = null;
10
11     if (this.imBossOfSubcloud) {
12         dir = new
               DescribeInstancesResponse(this.subcloud,
               this.nodeOfMySubcloud);
13         okMessage++;
14     } else if (this.isPartOfSubcloud) {
15         dir = new
               DescribeInstancesResponse(this.subcloud,
               this.neighboursTMan);
16         okMessage++;
17     }
18
19     try {
20         if (okMessage != 0) {
21             sess.sendMessage(myDestination,
                   myUtils.createBytesObject(dir));
22         } else {
23             sess.sendMessage(myDestination,
                   myUtils.createBytesObject(new
                   SubcloudDoesntExists()));
24         }
25         okMessage = 0;
26      } catch (I2PSessionException e) {
27             e.printStackTrace();
28      }
29 }
```

Codice 3.5: *Due metodi del passive thread del Bootstraping Service*

I metodi mostrati nel Codice 3.5 appartengono all'oggetto `TManReceiverI2P`, che rappresenta il *passive thread* del Bootstraping Service. Oltre a questo oggetto sono stati creati anche quelli relativi agli altri due servizi presenti all'interno del prototipo (`PssReceiverI2P` e `AggregationReceiverI2P`). Nella classe `ManageReceivingI2P` vengono recapitati tutti i messaggi destinati al peer, ma poi al suo interno vengono smistati ai tre diversi *passive thread* sulla base del messaggio ricevuto. In questo modo sono riuscito a non avere un'unica grande classe all'interno della quale gestire tutti i messaggi in entrata e a rendere più sensato il meccanismo inerente ai *passive thread*. Riassumendo, quindi, gli algoritmi utilizzati per Peer Sampling Service, Aggregation Service e Bootstraping Service sono rimasti gli stessi del precedente prototipo, mentre è cambiata la gestione di *active thread* e *passive thread*. Bisogna anche aggiungere che queste due entità nel precedente prototipo erano state realizzate estendendo l'interfaccia `Thread`, mentre nella versione I2P ho deciso di utilizzare l'estensione `TimerTask`, che permette di mandare in esecuzione il codice contenuto all'interno del metodo *run()* ogni tre secondi (valore utilizzato all'interno del prototipo) e di mantenere attiva la stessa destinazione I2P senza doverla ricreare ad ogni nuova esecuzione del thread. In questo modo ho replicato il comportamento delle due entità fondamentali di questo progetto, presenti in tutti e tre i servizi offerti.

3.3.6 Gestione degli script

L'ultimo argomento che rimane da analizzare riguarda gli script. Come spiegato nel secondo capitolo, gli script realizzati per la precedente versione del prototipo prevedevano una parte di script ed una di algoritmo. A causa della gestione delle comunicazioni e dello scambio di informazioni fornita da I2CP, ho dovuto rinunciare alla parte di algoritmo, che in sostanza prima permetteva di inviare la richiesta al Dispatcher System, il quale l'avrebbe poi inoltrata ai diretti interessati. Adesso ogni script che viene eseguito fuori dal sistema cloud innescherà la creazione di un nuovo peer, il quale avrà il compito di contattare direttamente il peer prescelto e di impartirgli le istruzioni necessarie affinché lo scopo dello script venga raggiunto. Prendiamo ad esempio in considerazione lo script `run_nodes`, che ha come obiettivo quello di creare una subcloud all'interno del sistema cloud P2P anonimo. In questo caso lo script prende in ingresso quattro parametri: il nome del peer che diventerà il capo della subcloud, la sua destinazione (ovvero il nome della chiave alla quale è associato, ad esempio *key1.dat*), il nome da attribuire alla subcloud e il numero di partecipanti. Una volta che tale script viene messo in esecuzione, verrà creato un peer che avrà come compito quello di contattare il peer che diventerà il capo della subcloud mediante la destinazione fornita in input. Il nuovo peer dovrà dunque comunicare al peer scelto come futuro capo il nome e il numero di partecipanti della subcloud, dopodiché il suo incarico sarà terminato. A quel punto sarà il peer contattato che prenderà in mano la situazione. Di tutti gli script

presi in considerazione ho modificato il comportamento del solo `terminate_nodes`, che nella versione I2P non permetterà di eliminare più nodi alla volta facenti parte di una stessa subcloud ma solamente uno ad ogni esecuzione dello script. Questa scelta è stata fatta in virtù della gestione delle destinazioni dei peer: per non appesantire eccessivamente lo script con troppi nomi e destinazioni (nella versione precedente infatti bastavano i nomi senza prendere in considerazione gli indirizzi) ho preferito concedere l'eliminazione di un unico peer per volta.

Capitolo 4

Valutazione delle performance

Come descritto nei capitoli precedenti, uno degli obiettivi principali di questo lavoro di tesi è quello di valutare le performance delle tre tipologie di rete presenti all'interno del prototipo. Dopo aver aggiunto le funzionalità della rete I2P è molto interessante considerare il comportamento del prototipo nella versione non anonima e in quelle anonime, per capire quali siano i compromessi da accettare in termini di velocità e stabilità per garantire l'anonimato in un sistema cloud P2P. Prima di procedere alla valutazione delle performance del prototipo, però, ho scelto di analizzare le performance dei tre diversi tipi di rete, per poter capire a priori che tipo di situazione avrei dovuto affrontare. In generale mi aspetto di ottenere performance inferiori nelle reti anonime in termini di Round Trip Time (RTT), ovvero il tempo che trascorre dal momento in cui un messaggio viene inviato al momento in cui si ottiene la relativa risposta, e in termini di throughput, che specifica la capacità effettivamente utilizzata di un canale di trasmissione. Dopo aver effettuato questo tipo di valutazioni, procederò con l'analisi del prototipo, concentrandomi prima sull'affidabilità del prototipo stesso utilizzando la versione anonima I2P, poi sull'effettiva valutazione delle performance che coinvolgerà tutte e tre le reti.

4.1 Performance delle reti

La prima considerazione da effettuare è relativa all'importanza di quest'analisi, che permetterà di capire se le reti anonime siano effettivamente meno performanti di quella classica. Qualora questa congettura dovesse rivelarsi corretta, si potrebbe accettare di avere all'interno del prototipo performance inferiori per quanto riguarda le reti di anonimizzazione, ma soprattutto si potrebbero trarre conclusioni sull'effettiva convenienza di utilizzare una rete più lenta ma anonima rispetto ad una rete più veloce ma non anonima. Per analizzare questa situazione ho effettuato diversi test grazie ai quali ho potuto comprendere più precisamente il comportamento delle reti di anonimizzazione. Di seguito elencherò dunque i risultati delle mie analisi corredati da una spiegazione che permetterà poi di affrontare la valutazione delle performance del prototipo.

4.1.1 Topologia del sistema cloud P2P

Per poter effettuare i test per valutare le performance delle reti, ho deciso di creare una topologia che prevedesse l'utilizzo di tre nodi all'interno di un sistema cloud P2P.

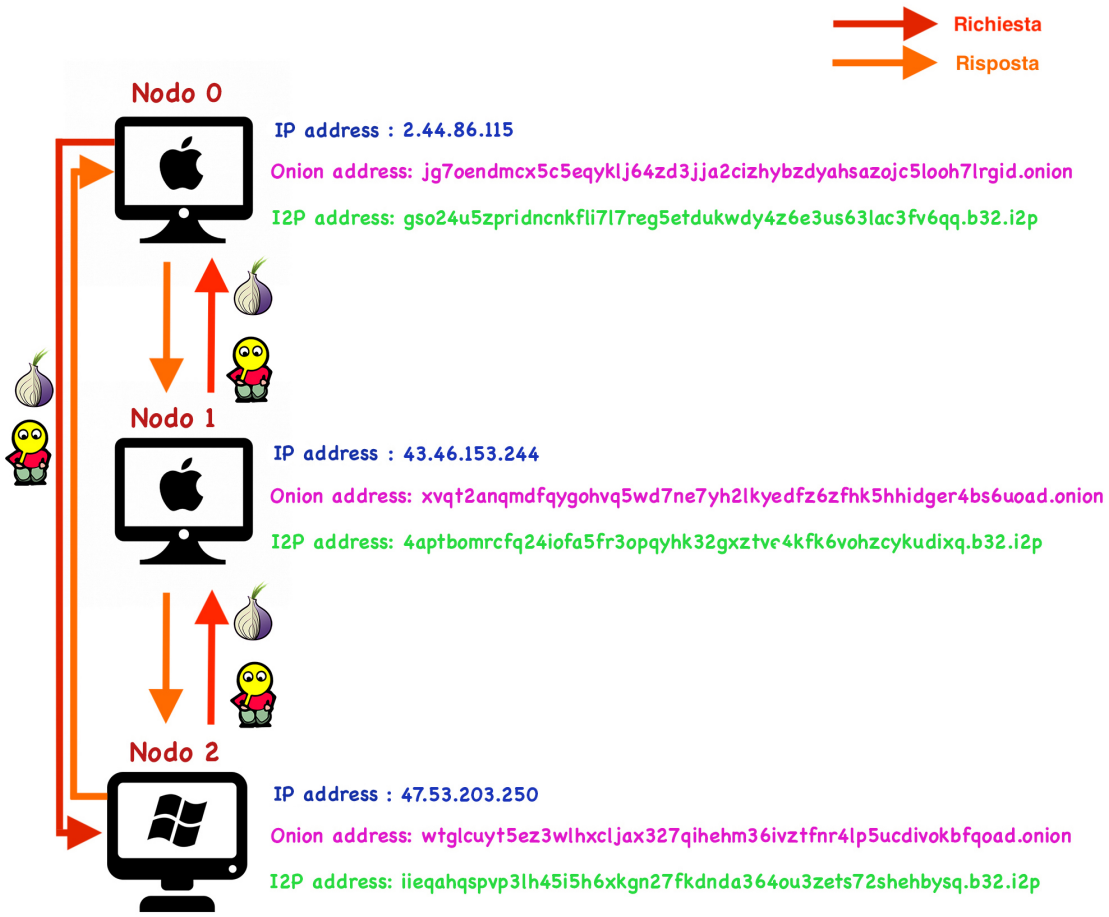

Figura 4.1: *Topologia utilizzata per i test della rete*

La Figura 4.1 mostra la topologia scelta per i test relativi alla rete. Il sistema cloud P2P è costituito da tre calcolatori, che d'ora in avanti chiamerò nodi, ognuno dei quali è connesso ad una rete diversa rispetto agli altri. Si consideri che le performance generiche delle reti alle quali sono connessi i nodi siano tutte sullo stesso livello e che, nel caso delle reti anonime, i sistemi Tor e I2P risultino già avviati e stabili, ovvero con circuiti e tunnel dei nodi già costruiti. L'obiettivo di questo sistema è quello di mettere in comunicazione coppie di nodi per calcolare il RTT medio, il throughput medio e la deviazione standard [70] per ognuna delle tre reti prese in considerazione. Per poter calcolare il throughput ho deciso di inviare al nodo destinatario la richiesta dello stesso documento presente su tutti e tre i nodi, cosicché le prestazioni possano essere comparate. Dopo aver terminato l'esecuzione dei test, l'obiettivo è quello di mettere in relazione i dati ottenuti ed effettuare alcune valutazioni in merito. Per l'esecuzione dei test ho deciso di procedere come segue. Per ogni coppia di nodi ho scelto di valutare le prestazioni unidirezionalmente, nel senso che il primo nodo avrà il compito di inviare una richiesta mentre il secondo nodo quello di rispondere. Non ho dunque preso in considerazione il contrario, ovvero l'invio della richiesta da parte del secondo nodo e la risposta del primo. Di seguito le specifiche hardware relative ai tre nodi utilizzati.

- **Nodo 0**. Processore: 2,9 GHz Intel Core i7. Scheda grafica: Intel HD Graphics 630 1536 MB. RAM: 16 GB 2133 MHz.
- **Nodo 1**. Processore: 2,5 GHz Intel Core i7. Scheda grafica: Intel Iris Pro 1536 MB. RAM: 16 GB 1600 MHz.
- **Nodo 2**. Processore: 2,5 GHz Intel Core i5-7200U. Scheda grafica: GeForce 940MX. RAM: 8 GB 1600 MHz.

Si tratta di tre macchine fisiche: quelle che ospitano Nodo 0 e Nodo 1 si trovano a Cesena mentre la terza si trova a Gambettola, un comune in provincia di Forlì-Cesena distante circa 15 km dalle altre due. La prima macchina è collegata alla rete Internet mediante Wi-Fi, una tecnologia per reti locali senza fili che utilizza dispositivi basati sugli standard IEEE 802.11 [71], così come la terza. Per quanto riguarda la seconda, invece, trovandosi sotto la stessa rete della prima, ho deciso di collegarla alla rete Internet attraverso l'hotspot del cellulare, così da sfruttare la connessione dati del telefono in maniera tale da poter utilizzare tre calcolatori collegati a tre reti differenti. Per ogni coppia di nodi verranno inviati 1000 messaggi suddivisi in quattro sessioni da 250 messaggi. Tali sessioni sono state tenute in momenti diversi della giornata, per avere un campione quanto più ampio possibile delle effettive prestazioni dei nodi, e verranno analizzate in maniera più approfondita nelle prossime sezioni. Ad ogni nodo sono stati associati tre indirizzi relativi alle tre reti utilizzate. Il primo,

l'*IP address*, rappresenta l'indirizzo della rete classica. Il secondo, l'*onion address*, rappresenta l'indirizzo della rete Tor, mentre l'*I2P address* rappresenta l'indirizzo della rete I2P.

4.1.2 Strumenti utilizzati

Per eseguire i test ho utilizzato alcuni strumenti che mi permettessero di ottenere i risultati ricercati. Per poter inviare richieste ai nodi della rete la scelta è ricaduta su *httping* [72], un software che permette di replicare il comportamento dell'applicativo *ping* [73] inviando però richieste di tipo Hypertext Transfer Protocol (HTTP) [74]. Non ho potuto utilizzare il software *ping* dal momento in cui sia la rete Tor che la rete I2P non permettono l'utilizzo di Internet Control Message Protocol (ICMP) [75], il protocollo che si occupa di trasmettere informazioni riguardanti malfunzionamenti, informazioni di controllo o messaggi tra i vari componenti di una rete di calcolatori. *Httping* si è comunque rivelato uno strumento molto potente. Ad esempio, uno dei comandi utilizzati per i test è stato il seguente:

httping -Gb 4aptbomrcfq24iofa5fr3opqyhk32gxztve4kfk6vohzcykudixq.b32.i2p -x 127.0.0.1:4444 -i 2 -c 250 -v

Httping mi ha permesso di utilizzare alcuni parametri per effettuare le richieste e per ottenere in output i risultati attesi. Di seguito una breve spiegazione dei parametri utilizzati.

- *G*. Permette di effettuare una GET (richiesta di un file) invece che una HEAD (richiesta del solo header) nei confronti dell'indirizzo specificato.
- *b*. Utilizzato insieme al parametro precedente, permette di visualizzare il throughput espresso in kB/s.
- *x*. Permette di specificare un proxy server attraverso il quale inoltrare la richiesta. In particolare questo parametro è stato utilizzato per la rete I2P, il cui proxy server fa riferimento alla porta 4444.
- *i*. Specifica l'intervallo di tempo misurato in secondi che intercorre tra l'invio di una richiesta e l'altra. Per i test effettuati ho deciso di inviare una richiesta ogni due secondi.
- *c*. Specifica il numero di pacchetti da inviare per singola sessione.
- *v*. Permette di calcolare la deviazione standard dell'intera sessione.

Mediante questo comando, dunque, sono stato in grado di calcolare, per ogni sessione da 250 pacchetti, il RTT medio, il throughput medio e la deviazione standard della

comunicazione. *Httping* permette poi di ottenere anche i valori minimi e massimi oltre a quelli medi elencati in precedenza, che però non prenderò in considerazione all'interno di questa valutazione. Un altro strumento utilizzato è *torsocks*, un software che permette di intercettare il traffico in uscita indirizzandolo attraverso un server SOCKS [76]. Questo strumento è stato utile per poter gestire l'invio delle richieste attraverso la rete Tor, in quanto tale rete non supporta il parametro x analizzato in precedenza. L'ultimo strumento preso in considerazione è MAMP (acronimo di MacOs, Apache, MySQL e PHP) [77], utilizzato per poter gestire un server web al quale verranno indirizzate le richieste dei nodi del sistema. Tale software è stato installato nei primi due nodi, mentre nel terzo ho scelto di utilizzare lo stesso ma nella versione per Windows (XAMPP, dove la X sottintende la disponibilità multipiattaforma del software [78]).

4.1.3 Risultati dei test

Dopo aver introdotto la topologia del sistema cloud P2P e gli strumenti utilizzati, mostrerò di seguito i risultati ottenuti dopo aver eseguito i test sui nodi della rete. Prima di tutto prenderò in considerazione i valori conseguiti per ogni coppia di nodi. Una volta fatto ciò, condenserò all'interno di una tabella le medie relative alle singole reti, per procedere poi con una valutazione sulle stesse. All'interno delle tabelle segnalerò i migliori risultati con il colore verde e quelli peggiori con il colore rosso.

4.1.3.1 Coppia di nodi Nodo 0 - Nodo 2

La prima coppia di nodi considerata è quella composta dal Nodo 0 (mittente) e dal Nodo 2 (destinatario). Di seguito i risultati dei test effettuati.

	Rete classica	**Tor**	**I2P**
Round Trip Time (RTT)	**247,9 ms**	**1735,27 ms**	**1781,17 ms**
Throughput	**822,59 kB/s**	**241,20 kB/s**	**228,43 kB/s**
Standard deviation	**168,95 ms**	**684,85 ms**	**810,1 ms**

Tabella 4.1: *Risultati dei test per la coppia Nodo 0 - Nodo 2*

I test effettuati per questa coppia di nodi hanno dato come risultati quanto elencato nella Tabella 4.1. Come si può notare, i risultati migliori sono stati quelli forniti dalla rete classica, mentre quelli peggiori dalla rete I2P per quanto non si discostino eccessivamente da quelli della rete Tor.

4.1.3.2 Coppia di nodi Nodo 1 - Nodo 0

La seconda coppia di nodi considerata è quella composta dal Nodo 1 (mittente) e dal Nodo 0 (destinatario). Di seguito i risultati dei test effettuati.

	Rete classica	**Tor**	**I2P**
Round Trip Time (RTT)	**270,55 ms**	**1161 ms**	**2100,62 ms**
Throughput	**526,82 kB/s**	**42,96 kB/s**	**60,42 kB/s**
Standard deviation	**178,9 ms**	**534,5 ms**	**1832,77 ms**

Tabella 4.2: *Risultati dei test per la coppia Nodo 1 - Nodo 0*

I test effettuati per questa coppia di nodi hanno dato come risultati quanto elencato nella Tabella 4.2. Come in precedenza, i risultati migliori sono stati quelli forniti dalla rete classica, mentre quelli peggiori dalla rete I2P ad eccezione del dato relativo al throughput, con la rete Tor che ha ottenuto risultati inferiori.

4.1.3.3 Coppia di nodi Nodo 2 - Nodo 1

La terza ed ultima coppia di nodi considerata è quella composta dal Nodo 2 (mittente) e dal Nodo 1 (destinatario). Di seguito i risultati dei test effettuati.

	Rete classica	**Tor**	**I2P**
Round Trip Time (RTT)	**269,57 ms**	**1607,95 ms**	**1676,85 ms**
Throughput	**536,7 kB/s**	**38,98 kB/s**	**43,04 kB/s**
Standard deviation	**178,6 ms**	**835,22 ms**	**928,22 ms**

Tabella 4.3: *Risultati dei test per la coppia Nodo 2 - Nodo 1*

I test effettuati per questa coppia di nodi hanno dato come risultati quanto elencato nella Tabella 4.3. Esattamente come nelle due valutazioni precedenti, i risultati migliori sono stati quelli forniti dalla rete classica, mentre quelli peggiori dalla rete I2P ad eccezione del dato relativo al throughput, con la rete Tor che ha ottenuto risultati inferiori.

4.1.3.4 Risultati complessivi

Dopo aver valutato i risultati dei test per ogni coppia di nodi, è ora interessante considerare tali dati aggregati, per analizzare ad ampio raggio le prestazioni delle tre reti.

	Rete classica	**Tor**	**I2P**
Round Trip Time (RTT)	**262,67 ms**	**1501,40 ms**	**1852,88 ms**
Throughput	**628,70 kB/s**	**107,61 kB/s**	**110,63 kB/s**
Standard deviation	**175,48 ms**	**684,85 ms**	**1190,36 ms**

Tabella 4.4: *Risultati aggregati dei test effettuati*

La Tabella 4.4 mostra i risultati conclusivi dei test, ottenuti come media delle medie di tutti i risultati mostrati nelle precedenti tabelle. Appare dunque evidente che l'assunzione fatta all'inizio di questo capitolo si sia rivelata fondata. La rete classica è estremamente più performante rispetto alla rete Tor e alla rete I2P, con quest'ultima che in generale si è rivelata meno veloce dell'altra rete di anonimizzazione considerata. Stando ai risultati, infatti, per quanto riguarda il RTT la rete classica ha impiegato un quinto del tempo rispetto a quanto fatto da Tor e un settimo rispetto a I2P. Anche il throughput è nettamente favorevole alla rete classica, con velocità di trasferimento sei volte superiore rispetto alla concorrenza. Per concludere, persino la deviazione standard risulta essere a vantaggio della rete classica, che dunque dal punto di vista delle prestazioni non conosce rivali. Del resto però questi risultati sono in linea con quanto atteso e sarà interessante valutare all'interno della prossima sezione le principali motivazioni di questo comportamento. Accantonando per un momento le prestazioni della rete classica, è il momento di prendere in considerazione le due reti di anonimizzazione. I risultati sono prevalentemente favorevoli alla rete Tor, sebbene nel caso della valutazione del throughput sia stata la rete I2P ad ottenere risultati migliori, comunque di pochi kB/s trasferiti. In generale però le performance delle due reti non si discostano eccessivamente, considerando che la rete I2P ha ottenuto risultati nettamente negativi rispetto a Tor solamente nelle sessioni di una delle tre coppie di nodi.

4.1.4 Motivazioni dei risultati ottenuti

Per concludere il discorso legato alle performance delle tre reti prese in considerazione, è interessante valutare le motivazioni per le quali la rete classica sia riuscita ad avere prestazioni nettamente favorevoli nei confronti delle reti anonime. Si può considerare, prima di tutto, la questione relativa al percorso che un pacchetto debba seguire prima di giungere all'effettiva destinazione. Nelle reti anonime, come mostrato nel secondo capitolo di questa tesi, il pacchetto deve attraversare un percorso formato da più nodi (solitamente tre, sia nel caso di Tor che nel caso di I2P) con l'obiettivo di scongiurare un'eventuale analisi del traffico o altri attacchi da parte di utenti malintenzionati. È importante ricordare che i router intermedi che compongono il circuito potrebbero trovarsi geograficamente in luoghi molto distanti ma soprattutto potrebbero disconnettersi dalla rete in qualunque momento, costringendo il sistema a chiudere il circuito e ad aprirne un altro, con le performance che, inevitabilmente, ne risentirebbero. Oltre a questa situazione bisogna poi considerare anche la fase relativa alla cifratura e alla decifratura del messaggio: sia in Tor che I2P infatti il pacchetto viene crittato con più chiavi, cosicché ogni router possa poi decrittare il proprio strato di protezione. Nel caso di Tor basterà che tutti gli Onion Router che costituiscono il circuito decrittino il proprio strato e lo inoltrino al router successivo, mentre nel caso di I2P bisogna ricordare che, oltre al percorso seguito all'interno dell'*outbound tunnel*, il pacchetto dovrà attraversare anche l'*inbound tunnel* del destinatario, all'interno del quale verrà nuovamente cifrato dai router che lo compongono. È lecito pertanto aspettarsi che le prestazioni della rete Tor siano leggermente migliori rispetto a quelle della rete I2P, a discapito di una minor sicurezza e protezione del proprio anonimato. Sia per Tor che per I2P è possibile intervenire sui file di configurazione per specificare eventuali preferenze relative alla costruzione del circuito che i pacchetti seguiranno per ottenere performance migliori. Ad esempio si potrebbe fornire un elenco di nodi i quali agirebbero come *exit node* ed *entry node* nel caso di Tor e come *gateway* ed *endpoint* nel caso di I2P, oppure richiedere che i nodi intermedi facciano parte di uno specifico paese per evitare che i router si trovino geograficamente molto distanti. Ad ogni modo nella valutazione delle performance delle reti ho deciso di non intervenire su questi file ma di procedere con la configurazione di default, fermo restando che i risultati ottenuti hanno il solo scopo di offrire una visione generale delle performance di tali reti e che non possono essere considerati una valutazione empirica.

4.2 Affidabilità del prototipo

Dopo aver analizzato le performance delle tre reti ed aver notato come quelle anonime abbiano performance minori rispetto a quella classica, è interessante valutare invece le performance relative al prototipo. Prima di elencare però i test effettuati e comparare i risultati ottenuti, ho ritenuto corretto verificare l'affidabilità del prototipo. Dal momento in cui all'interno di questo lavoro di tesi mi sono occupato di aggiungere le funzionalità della rete I2P e che quelle della rete classica e della rete Tor sono già state testate, ho scelto di verificare l'affidabilità della sola rete I2P prima di procedere con la valutazione delle performance del prototipo. All'interno di questa sezione mostrerò dunque i test effettuati sul prototipo utilizzando la rete I2P per verificarne l'affidabilità, mentre in seguito ne valuterò le performance utilizzando tutte e tre le reti per riuscire a confrontare i risultati ottenuti.

4.2.1 Invio e ricezione dei messaggi

Il primo e più importante test da effettuare riguarda l'invio e la ricezione dei messaggi all'interno della rete I2P. La maggior parte di essi viene inviata all'interno del Peer Sampling Service, il servizio che permette ad un peer di scoprire nella rete altri peer con i quali scambiare informazioni. I miei test pertanto si sono concentrati su questo servizio e avevano lo scopo di valutare quanti dei messaggi inviati dai peer all'interno del sistema cloud P2P giungessero correttamente agli effettivi destinatari. Ad ogni peer ho dunque assegnato un contatore per i messaggi in entrata ed un contatore per i messaggi in uscita. Per poter verificare l'affidabilità del prototipo, ho scelto di far inviare a ciascun peer 100 messaggi per cinque sessioni, per un totale di 500 messaggi inviati da ciascun peer. La topologia scelta è quella presentata all'interno della sezione relativa alle performance della rete e sarà la stessa utilizzata per la valutazione delle performance del prototipo, anche se in questi casi ovviamente non ho preso in considerazione le coppie di nodi ma ciascun peer, utilizzando l'algoritmo del PSS, decideva autonomamente chi contattare. L'obiettivo finale è quello di verificare quanti dei 300 messaggi inviati in ogni sessione siano correttamente stati recapitati ai destinatari, e per farlo basta sommare i risultati forniti dai contatori in entrata di ogni peer. In questo modo infatti, dal momento in cui vengono inviati da ciascun peer 100 messaggi a sessione, un prototipo affidabile dovrebbe permettere la ricezione di tutti i 300 messaggi inviati. In tutte e cinque le sessioni effettuate i peer hanno sempre ricevuto 100 messaggi e ne hanno sempre inviato lo stesso numero, soddisfacendo dunque il requisito iniziale.

4.2.2 Creazione ed operazioni della subcloud

Un altro obiettivo dei test relativi all'affidabilità del prototipo riguarda la creazione e la gestione delle subcloud grazie al Bootstraping Service e agli script presentati nelle sezioni precedenti. Essendo questa funzionalità una delle più importanti del prototipo ho deciso di testarne l'affidabilità prima di procedere con la valutazione delle performance. I test effettuati sono stati molteplici e mirati a verificare il corretto funzionamento di ogni singolo script. La prima fase prevede la creazione di una subcloud da parte di uno dei tre peer all'interno della rete. L'obiettivo è quello di combinare tale creazione (utilizzando lo script `run_nodes.sh`) con la sua verifica (mediante lo script `describe_instances.sh`). Ogni peer dunque è stato utilizzato come capo della subcloud in cinque occasioni per tre sessioni, per un totale di 45 tentativi comprendenti tutti i peer. Anche in questo caso la percentuale di subcloud create correttamente è stata del 100%, percentuale verificata grazie al secondo script elencato. Dopo averla correttamente creata, un altro test interessante riguarda la richiesta al peer capo o ai peer ad essa appartenenti di creare un'altra subcloud e valutarne il comportamento. Come atteso, tutti i peer hanno risposto di appartenere già ad una subcloud e si sono rifiutati di crearne una nuova. Anche le operazioni per poter aggiungere un nodo ad una subcloud esistente (`add_new_nodes.sh`) e per poterlo rimuovere (`terminate_nodes.sh`) si sono rivelate corrette, con il prototipo che ha sempre risposto positivamente a tutti i comandi effettuati. In questo modo ho dunque verificato l'affidabilità del prototipo, che per tutte le funzionalità principali ha sempre mostrato il comportamento atteso.

4.3 Performance del prototipo

L'ultima sezione riguarda la valutazione delle performance del prototipo, che costituisce il secondo grande obiettivo del lavoro di questa tesi dopo l'implementazione delle funzionalità relative alla rete I2P. Dopo aver valutato le prestazioni delle tre reti prese in considerazione e l'affidabilità del prototipo, è giunto dunque il momento di analizzare i risultati ottenuti dai test effettuati sul prototipo utilizzando la rete classica, la rete Tor e la rete I2P. Di seguito dunque elencherò le analisi effettuate, prima di trarre le dovute conclusioni sulla base di quanto emerso.

4.3.1 Inizializzazione delle reti e dei peer

Il primo test riguarda l'inizializzazione delle tre reti utilizzate e la partenza dei peer all'interno del sistema cloud P2P. È molto interessante infatti valutare quanto tempo impieghino le tre reti per la fase di bootstrap prima che il peer venga correttamente avviato ed inizi a comunicare con altri peer presenti all'interno della rete. Per ogni peer e per ogni rete ho avviato il prototipo dieci volte, così da poter confrontare, per i tre peer del sistema, la media dei risultati ottenuti.

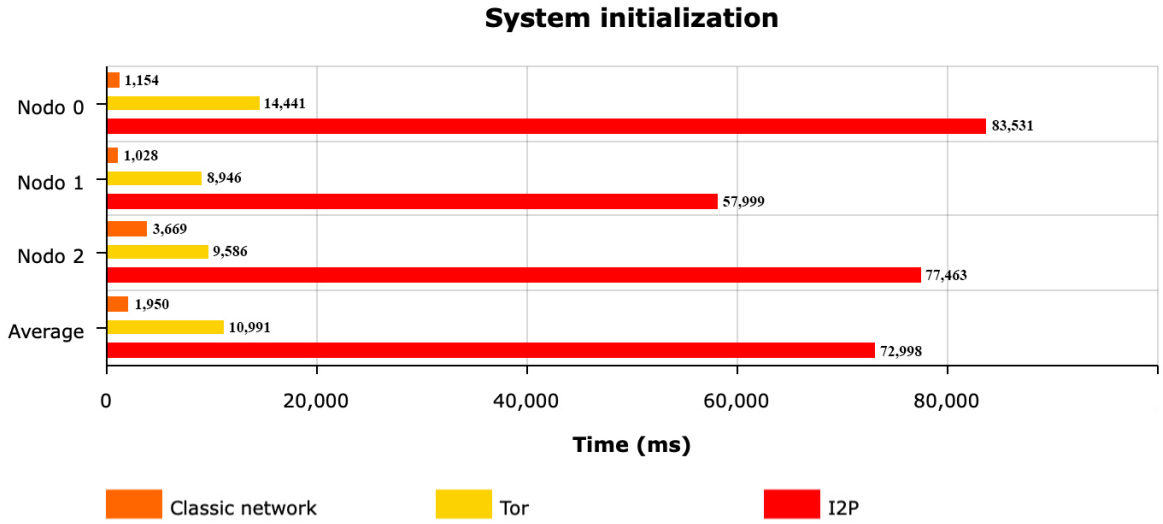

Figura 4.2: *Grafico del tempo richiesto per inizializzare le reti*

La Figura 4.2 mostra, per ogni peer appartenente al sistema cloud P2P anonimo, la media delle somme del tempo (in millisecondi) impiegato da ogni rete per poter essere avviata e quello relativo all'inizializzazione del peer. Questo test è risultato molto interessante in quanto sono emerse le prime, grosse differenze nei confronti delle due reti di anonimizzazione utilizzate nel prototipo. La rete classica infatti non ha subito eccessivi rallentamenti dovuti all'inizializzazione, mentre le prestazioni della rete I2P si sono dimostrate nettamente inferiori rispetto a quelle della rete Tor. Tutti i nodi hanno

infatti impiegato più tempo ad avviare la rete I2P rispetto alla rete Tor, con risultati sorprendenti data l'evidente differenza di prestazioni delle due reti anonime. La media delle medie dei tre peer considerati mostrata nel grafico dimostra quanto la fase di bootstrap sommata all'inizializzazione del peer nella rete I2P sia più lunga rispetto a quella della rete Tor. Mediamente, per poter essere avviato, un peer ha impiegato 1950 ms nella rete classica, 10991 ms nella rete Tor e addirittura 72998 ms nella rete I2P.

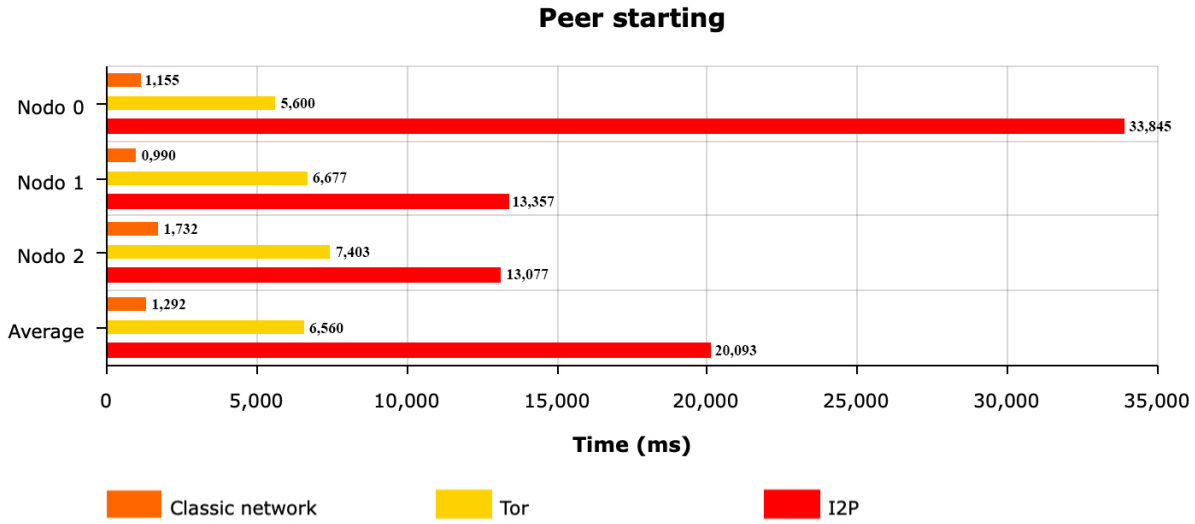

Figura 4.3: *Grafico del tempo richiesto per avviare i peer con le reti già inizializzate*

Dopo aver preso in considerazione quanto esposto nel paragrafo precedente, è interessante ora valutare quanto il peer impieghi ad essere inizializzato con il sistema già avviato. Dai test precedenti infatti emerge un dubbio che è necessario risolvere: le prestazioni della rete I2P sono nettamente inferiori a causa della fase di bootstrap, dell'inizializzazione del peer o di entrambe queste situazioni? La Figura 4.3 mostra come il divario tra la rete Tor e la rete I2P sia effettivamente diminuito se si prende in considerazione solamente l'inizializzazione del peer con il sistema già avviato. Mediamente infatti, per poter essere avviato, un peer ha impiegato 1292 ms nella rete classica, 6560 ms nella rete Tor e 20093 ms nella rete I2P. La differenza tra le due reti di anonimizzazione è dunque scesa a 13533 ms rispetto ai 62006 ms dei test effettuati in precedenza. Questo dimostra come effettivamente la fase di bootstrap delle due reti abbia influito sulle prestazioni del prototipo, con la rete I2P che rimane comunque più lenta rispetto a Tor anche per quanto riguarda l'inizializzazione di un peer.

4.3.2 Invio e ricezione di messaggi

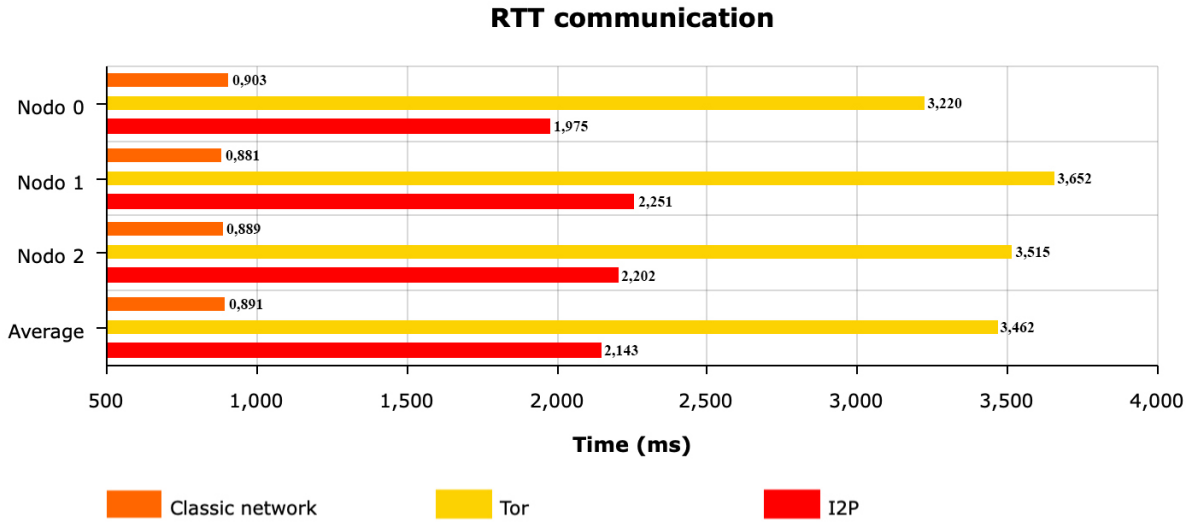

Figura 4.4: *Grafico relativo ai valori dei RTT misurati*

Terminata l'analisi relativa all'inizializzazione del sistema e dei peer, è giunto il momento di valutare la reattività delle reti prendendo in considerazione il RTT delle comunicazioni. Come spiegato in precedenza, si tratta del tempo che intercorre tra l'invio di un messaggio e la ricezione della risposta a quel messaggio ed è risultato molto utile per poter valutare le performance delle reti utilizzate. Ogni messaggio del Peer Sampling Service che viene inviato da un peer all'interno del prototipo comprende, oltre alle informazioni necessarie per gestire il servizio in questione, una variabile di tipo `long` che rappresenta il *timestamp*, ovvero una sequenza di cifre che identifica il momento specifico in cui è stato inviato il messaggio. Chi riceve il messaggio non dovrà fare altro che rispondere includendo nuovamente tale variabile, cosicché il mittente possa poi definire un secondo *timestamp* nel momento in cui riceverà la risposta del destinatario ed infine calcolare la differenza tra i due valori, ottenendo così il RTT (in millisecondi) relativo a quella comunicazione. Per tutte e tre le reti prese in considerazione, ogni peer mantiene in memoria 50 RTT per poi calcolarne la media. In questo modo, dopo l'invio di 50 messaggi da parte del peer, il sistema calcolerà in automatico la media di quei tempi restituendo il RTT medio. A quel punto i 50 RTT già valutati verranno eliminati in attesa dei successivi. Per riuscire a portare a termine il test, ho messo in esecuzione i tre peer nello stesso momento e ho atteso che ciascuno di essi avesse inviato agli altri peer 500 messaggi, così da ottenere come risultato 10 RTT per ogni peer all'interno di ogni rete. Una volta fatto ciò, ho inserito i risultati all'interno della Figura 4.4 per tenere traccia della media di ogni peer in ogni rete ed infine ho calcolato la media delle medie per poter analizzare più genericamente i risultati ottenuti. Come è facilmente intuibile, la prima novità che si evince guardando il grafico è relativa alla

differenza delle performance della rete Tor e della rete I2P. Mentre nell'inizializzazione del sistema e del peer era stata la rete Tor ad ottenere risultati migliori, in questo caso la situazione è stata completamente ribaltata. Mediamente infatti all'interno della rete Tor è necessario attendere 3462 ms per poter completare uno scambio di informazioni tra due peer, mentre nella rete I2P bastano appena 2143 ms. Rimangono invece irraggiungibili le prestazioni della rete classica, con uno scambio di informazioni che in media viene completato in 891 ms. Alla luce di questi risultati, una domanda sorge però spontanea: come mai nella valutazione delle performance delle reti i risultati ottenuti erano favorevoli alla rete Tor, mentre utilizzando il prototipo è la rete I2P a beneficiarne? La risposta a tale questione riguarda Java RMI, la tecnologia utilizzata all'interno del prototipo per la versione non anonima e per la versione anonima con Tor. Ogni peer infatti, per poter dialogare con altri peer, deve prima creare una connessione con il registro RMI, che fornirà la chiave per poter contattare il server, e successivamente con il server RMI, al quale comunicare il messaggio che si desidera inviare. Per questo motivo, soprattutto all'interno della rete anonima, la latenza avrà un impatto doppio per ogni comunicazione, dovendo aprire ad ogni interazione tra peer un canale sia con il registro che con il server RMI. I risultati medi inseriti nella Tabella 4.4 relativa alle performance delle reti mostrano chiaramente come Tor abbia tempi quasi dimezzati rispetto a quanto visto nel prototipo (1501 ms contro 3462 ms) proprio per il fatto che Java RMI tende ad aumentare la latenza della comunicazione. I tempi relativi ad I2P sono invece peggiorati di pochi ms: si è passati infatti dai 1852 ms delle performance della rete ai 2143 ms delle performance del prototipo. Considerando però che nel secondo caso si aggiungono ulteriori computazioni relative al servizio del PSS, il risultato è accettabile.

4.3.3 Creazione delle subcloud

L'ultima analisi effettuata riguarda la creazione delle subcloud all'interno del prototipo. Come analizzato in precedenza si tratta di un servizio molto importante che permette di poter creare insiemi ristretti di peer che fanno parte del sistema cloud P2P anonimo. Per poter valutare le performance relative alla creazione di tale servizio ho scelto di eseguire, per ogni peer all'interno di ogni rete, tre volte lo script relativo alla creazione di una subcloud, analizzando al termine di queste operazioni i risultati ottenuti.

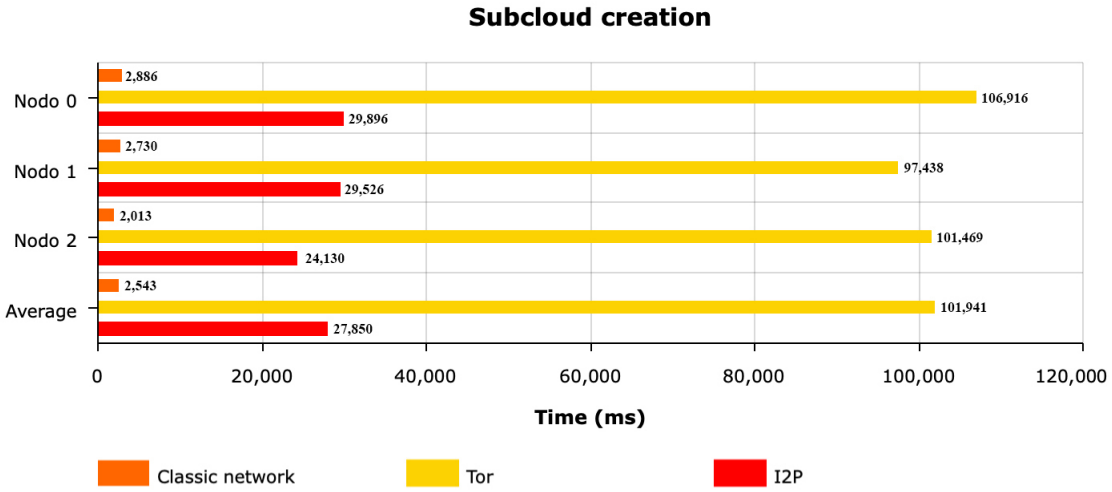

Figura 4.5: *Grafico del tempo richiesto per creare una subcloud*

La Figura 4.5 mette in evidenza come il prototipo che utilizza la rete Tor abbia prestazioni nettamente inferiori rispetto alle altre due reti. Prima di tutto però bisogna segnalare il diverso approccio che caratterizza la rete I2P e la rete Tor nella creazione di una subcloud. Oltre alla latenza introdotta da Java RMI, le performance del prototipo che utilizza Tor vengono ridotte a causa del meccanismo di ricerca dei peer messo in atto dal Dispatcher System che rallenta la creazione della subcloud. Nel caso del prototipo che utilizza invece la rete I2P la maggior parte del tempo viene impiegata dal sistema per inizializzare il peer che avrà il compito di contattare il peer capo della subcloud, pertanto le performance dipenderanno esclusivamente dal periodo di avviamento del peer. La differenza tra le due versioni è notevole: da un lato il prototipo con Tor impiega mediamente 101941 ms per creare una subcloud, dall'altro I2P garantisce lo stesso risultato in appena 27850 ms. Bisogna inoltre considerare che, all'aumentare del numero di peer da inserire all'interno di una subcloud, le prestazioni di I2P non dovrebbero peggiorare in quanto è l'inizializzazione del peer a richiedere tempo, mentre non si può dire lo stesso per Tor, che ad ogni peer aggiuntivo da contattare vedrà le proprie performance sensibilmente ridotte.

Conclusioni

Lavorare sulla rete di anonimizzazione I2P è stato molto stimolante, poiché la documentazione della stessa rete e del protocollo I2CP utilizzato non è sicuramente allo stesso livello di quella fornita da Tor. Nonostante qualche difficoltà riscontrata nella progettazione e nella realizzazione del prototipo, sono soddisfatto di come sia riuscito a portare a termine questo lavoro. È stato molto gratificante riuscire a riutilizzare parte del codice presente nel prototipo precedente ma soprattutto inserire le istruzioni di I2P all'interno degli script già esistenti, in modo tale che, sia che si voglia utilizzare la rete Tor sia che si voglia utilizzare la rete I2P, i file da eseguire siano gli stessi e il livello di automatizzazione sia elevato. Per quanto riguarda eventuali sviluppi futuri, l'attenzione è posta sulla gestione della conferma di avvenuta ricezione di un messaggio. Come detto in precedenza, infatti, al momento I2CP non permette al mittente di gestire eventuali mancate risposte né consente di verificare se il peer con il quale si vuole interagire sia attivo oppure no. Qualora il protocollo I2CP aggiungesse una funzionalità per permettere tutto questo, sarebbe interessante inserirla all'interno del prototipo, cosicché il mittente possa inviare un messaggio per capire se il peer sia attivo oppure no, prima ancora di inviare dati relativi al sistema cloud P2P anonimo. In un sistema all'interno del quale sono presenti numerosi peer, il problema potrebbe non porsi in quanto i *churn* verrebbero eliminati immediatamente e rimpiazzati da peer attivi, cosicché non si presenti neppure il pericolo di contattare un peer disconnesso. Nel caso invece di una rete piuttosto piccola, il non capire perché un peer non risponda potrebbe rappresentare una situazione negativa. Un altro sviluppo molto interessante potrebbe riguardare la fase di bootstrap. Al momento infatti un peer ha bisogno di recuperare le destinazioni di altri peer da file per poter iniziare a scambiare messaggi. La situazione ideale però potrebbe prevedere di recuperare queste informazioni dal *netDB*, con la difficoltà principale che riguarderebbe il riconoscimento delle destinazioni dei peer che fanno effettivamente parte del sistema cloud P2P anonimo. Il *netDB* infatti contiene le informazioni relative a tutte le destinazioni presenti nella rete e non solo quelle dei peer del nostro sistema e al momento non è possibile distinguerle. Dal punto di vista delle performance invece i risultati sono stati abbastanza sorprendenti, soprattutto per quanto riguarda il prototipo. Quando ho iniziato a lavorare alla rete I2P non pensavo potesse competere con Tor, più famosa ed utilizzata, invece mi sono

dovuto ricredere. Dal punto di vista delle performance della rete i risultati ottenuti sono stati simili, con le prestazioni di Tor leggermente superiori a quelle di I2P. All'interno del prototipo però la situazione è stata ribaltata, con la rete I2P che è risultata migliore sia in termini di RTT relativi alla comunicazione tra peer sia in termini di tempi relativi alla creazione di una subcloud. In queste valutazioni ha certamente influito la tecnologia utilizzata nel prototipo precedente, con Java RMI che purtroppo introduce maggiore latenza che rischia di compromettere le performance del prototipo che utilizza la rete Tor. In futuro sarebbe interessante riprogettare la parte relativa alla versione non anonima e alla versione con Tor in maniera tale da poter eliminare Java RMI in favore di una tecnologia più performante.

Bibliografia

[1] Michele Tamburini. ≪Progettazione e Sviluppo di un Sistema Cloud P2P≫. Tesi di laurea mag. 2010-2011.

[2] *Tor Project: Anonymity Online*. URL: https://www.torproject.org/.

[3] Michele Amati. ≪Design and implementation of an anonymous peer-to-peer iaas cloud.≫ Tesi di laurea mag. 2013-2014.

[4] *The Invisible Internet Project*. URL: https://geti2p.net/.

[5] Marcosiris A. O. Pessoa et al. *Industry 4.0, How to Integrate Legacy Devices: A Cloud IoT Approach*. Rapp. tecn. 2018. URL: https://ieeexplore.ieee.org/document/8592774.

[6] Peter Mell e Timothy Grance. *The NIST Definition of Cloud Computing*. Rapp. tecn. NIST, 2011. URL: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf.

[7] *The Enterprise Cloud. Best practice for transforming legacy IT*. James Bond, 2015.

[8] Klaithem Al Nuaimi et al. *A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms*. Rapp. tecn. College of Information Technology, UAEU Al Ain, United Arab Emirates, 2012. URL: https://www.researchgate.net/profile/Klaithem_Nuaimi/publication/261277641_A_Survey_of_Load_Balancing_in_Cloud_Computing_Challenges_and_Algorithms/links/53f234a10cf2bc0c40e7271c/A-Survey-of-Load-Balancing-in-Cloud-Computing-Challenges-and-Algorithms.pdf.

[9] Ozalp Babaoglu e Moreno Marzolla. *Peer-to-Peer Cloud Computing*. Rapp. tecn. Department of Computer Science e Engineering University of Bologna, Italy, 2014. URL: http://www.cs.unibo.it/babaoglu/papers/pdf/P2Pclouds.pdf.

[10] Tobias Kurze et al. *Cloud Federation*. Rapp. tecn. Karlsruhe Institute of Technology (KIT), 2011. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.682.658&rep=rep1&type=pdf.

[11] David Barkai. *Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net*. Intel Press, 2001.

[12] Steven Levy. ≪How the NSA Almost Killed the Internet≫. In: *Wired* (2014). URL: https://www.wired.com/2014/01/how-the-us-almost-killed-the-internet/.

[13] Gloria Ghioni. ≪Tra anonimi e pseudonimi: viaggio nell'editoria italiana del '700≫. In: *illibrario.it* (2019). URL: https://www.illibraio.it/anonimato-scrittori-1041915/.

[14] Stallings William. *Computer Security: Principles And Practice*. Pearson Education, 2011.

[15] Benjamin George Sanders, Paul Dowland e Steven Furnell. *Implications and Risks of MMORPG Addiction: Motivations, Emotional Investment, Problematic Usage and Personal Privacy*. Rapp. tecn. Centre for Security, Communications, Network Research, University of Plymouth, Plymouth, UK, School of Computer e Information Science, Edith Cowan University, Perth, Western Australia, 2010. URL: https://bit.ly/35wnUdt.

[16] Yvonne Nouwen, Alessia Altamura e Andrea Varrella. *Online child sexual exploitation. An Analysis of Emerging and Selected Issue*. Rapp. tecn. Ecpat, 2017. URL: https://www.ecpat.org/wp-content/uploads/2017/04/Journal_No12-ebook.pdf.

[17] *Freenet*. URL: https://freenetproject.org.

[18] Mark Sportack. *IP Addressing Fundamentals*. Cisco Press, 2002.

[19] Aws Naser Jaber e Mohamad Fadli Bin Zolkipli. *Use of cryptography in cloud computing*. Rapp. tecn. 2013. URL: https://ieeexplore.ieee.org/abstract/document/6719955.

[20] *Crittografia*. URL: http://www.treccani.it/enciclopedia/crittografia.

[21] Michael Ammann. ≪Design and Implementation of a Peer-to-Peer based System to enable the Share Functionality in a Platform-independent Cloud Storage Overlay≫. Tesi di laurea mag. 2012-2013.

[22] *PiCsMu*. URL: https://www.csg.uzh.ch/csg/en/research/picsmu.html.

[23] Risto Laurikainen. *Secure and anonymous communication in the cloud*. Rapp. tecn. Aalto University School of Science e Technology, 2011. URL: http://www.cse.hut.fi/en/publications/B/11/papers/laurikainen.pdf.

[24] *Logo Tor Project*. URL: https://www.muyseguridad.net/2017/07/24/proyecto-tor-programa-recompensas-hackerone/.

[25] *The Tor Project Inc. Tor history*. URL: https://www.torproject.org/about/history/.

[26] David M. Goldschlag, Michael G. Reed e Paul F. Syverson. *Proceedings of the First International Workshop on Information Hiding*. Springer-Verlag, 1996. URL: https://link.springer.com/chapter/10.1007/3-540-61996-8_37.

[27] *The Tor Project Inc. Tor metrics - direct users by country*. 2019. URL: https://metrics.torproject.org/userstats-relay-country.html.

[28] Roger Dingledine, Nick Mathewson e Paul Syverson. *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13,* USENIX Association, 2004. URL: http://dl.acm.org/citation.cfm?id=1251375.1251396.

[29] *The Onion Routing - Packet*. URL: https://en.wikipedia.org/wiki/Onion_routing.

[30] Bassam Zantout Ramzy A. Haraty. *The TOR data communication system*. Rapp. tecn. 2014. URL: https://ieeexplore.ieee.org/abstract/document/6896565.

[31] Kevin Bauer et al. *Low-resource routing attacks against tor*. Rapp. tecn. 2007. URL: https://dl.acm.org/citation.cfm?id=1314336.

[32] *End-to-end encryption*. URL: https://searchsecurity.techtarget.com/definition/end-to-end-encryption-E2EE.

[33] TT. Dierks e E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. Rapp. tecn. 2008. URL: http://www.hjp.at/doc/rfc/rfc5246.html.

[34] Zhen Ling et al. «Equal-sized cells means equal-sized packets in Tor?» In: giu. 2011. URL: https://ieeexplore.ieee.org/abstract/document/5962653.

[35] Ian Goldberg. *On the Security of the Tor Authentication Protocol*. Springer-Verlag Berlin Heidelberg, 2006.

[36] Nan Li. *Research on Diffie-Hellman key exchange protocol*. Rapp. tecn. 2012. URL: https://ieeexplore.ieee.org/abstract/document/5485276.

[37] *Tor Hidden Service*. URL: https://2019.www.torproject.org/docs/onion-services.html.en.

[38] Nan Li. *Empirical analysis of Tor Hidden Services*. Rapp. tecn. 2016. URL: https://digital-library.theiet.org/content/journals/10.1049/iet-ifs.2015.0121.

[39] *One-time-secret*. URL: https://it.wikipedia.org/wiki/One-time_password.

[40] *Logo Invisible Internet Project - I2P*. URL: https://upload.wikimedia.org/wikipedia/commons/a/ae/I2P_logo.svg.

[41] *Invisible Internet Project History*. URL: http://www.i2pproject.net/it/about/intro.

[42] *Invisible Internet Project Metrics*. URL: https://i2p-metrics.np-tokumei.net/overview.

[43] *Route I2P packets*. URL: http://www.geti2p.org/how_intro.

[44] Bernd Conrad e Fatemeh Shirazi. *A Survey on Tor and I2P*. Rapp. tecn. Department of Computer Science, TU Darmstadt Darmstadt, Germany, 2014. URL: http://www.i2pproject.net/_static/pdf/icimp_2014_1_40_30015.pdf.

[45] Bassam Zantout e Ramzi A. Haraty. *I2P Data Communication System*. Rapp. tecn. 2011. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.927.1044&rep=rep1&type=pdf.

[46] P. Maymounkov e D. Mazieres. *Kademlia: A peer-to-peer information system based on the xor metric, in Peer-to-Peer Systems*. Springer, 2002.

[47] zzz (Pseudonym) e Lars Schimmer. «Peer Profiling and Selection in the I2P Anonymous Network». In: *Proceedings of PET-CON 2009.1*. Dresden, Germany, mar. 2009, pp. 59–70.

[48] Nguyen Phong Hoang et al. *An Empirical Study of the I2P Anonymity Network and its Censorship Resistance*. Rapp. tecn. 2018. URL: https://censorbib.nymity.ch/pdf/Hoang2018a.pdf.

[49] *Base64*. URL: https://en.wikipedia.org/wiki/Base64.

[50] *Base32*. URL: https://en.wikipedia.org/wiki/Base32.

[51] *Route I2P encryption*. URL: http://www.geti2p.org.

[52] Matt Hooks e Jadrian Miles. *Onion Routing and Online Anonymity*. Rapp. tecn. 2006. URL: https://bit.ly/31e0iqI.

[53] Yue Gao et al. *Large scale discovery and empirical analysis for I2P eepSites*. 2017, pp. 444–449. URL: https://ieeexplore.ieee.org/abstract/document/8024569.

[54] *Comparison of Tor and I2P Terminology*. URL: https://geti2p.net/en/comparison/tor.

[55] *I2Pmail - Susimail*. URL: https://geti2p.net/it/docs/how/tech-intro#app.i2pmail.

[56] Timothy G. Abbott et al. *Browser-Based Attacks on Tor*. 2007, pp. 184–199. URL: https://link.springer.com/chapter/10.1007/978-3-540-75551-7_12.

[57] Roger Dingledine e Steven J. Murdoch. *Performance improvements on Tor or, why Tor is slow and what we're going to do about it*. Rapp. tecn. 2009. URL: http://tor.void.gr/press/presskit/2009-03-11-performance.pdf.

[58] Mark Jelasity et al. *Gossip-based peer sampling*. 2007. URL: https://dl.acm.org/citation.cfm?id=1275520.

[59] Mark Jelasity, Alberto Montresor e Ozalp Babaoglu. *Gossip-based Aggregation in Large Dynamic Networks*. 2005. URL: http://www.cs.unibo.it/babaoglu/papers/pdf/acm-tocs-2005.pdf.

[60] Mark Jelasity, Alberto Montresor e Ozalp Babaoglu. *T-MAN: Gossip-based fast overlay topology construction*. 2009. URL: http://cs.unibo.it/babaoglu/papers/pdf/tman.pdf.

[61] *Java RMI*. URL: https://docs.oracle.com/javase/tutorial/rmi/index.html.

[62] *SOCKS definition*. URL: https://en.wikipedia.org/wiki/SOCKS.

[63] Craig Alan Bennett et al. *Client side socks server for an internet client*. Rapp. tecn. 2006. URL: https://patentimages.storage.googleapis.com/4c/bf/15/3bdde96a52d382/US7020700.pdf.

[64] *SOCKS and SOCKS proxies in I2P*. URL: https://geti2p.net/it/docs/api/socks.

[65] *I2P Client Protocol*. URL: https://geti2p.net/it/docs/protocol/i2cp.

[66] Tim De Boer e Vincent Breider. *Invisible Internet Project (I2P)*. Rapp. tecn. 2019. URL: https://www.delaat.net/rp/2018-2019/p63/report.pdf.

[67] *Setup I2P*. URL: https://help.ubuntu.com/community/I2P.

[68] *Why Are Hashes Irreversible?* URL: https://learncryptography.com/hash-functions/why-are-hashes-irreversible.

[69] *Acknowledge definition*. URL: https://en.wikipedia.org/wiki/Acknowledgement_(data_networks).

[70] *Standard deviation definition*. URL: https://en.wikipedia.org/wiki/Standard_deviation.

[71] B.P. Crow et al. *IEEE 802.11 Wireless Local Area Networks*. Rapp. tecn. 1997. URL: https://ieeexplore.ieee.org/abstract/document/620533.

[72] *Httping*. URL: https://www.vanheusden.com/httping.

[73] *Ping*. URL: https://en.wikipedia.org/wiki/Ping_(networking_utility).

[74] *HTTP definition*. URL: https://www.w3schools.com/whatis/whatis_http.asp.

[75] *Tor protocols*. URL: https://bit.ly/2NrGNaq.

[76] *Torsocks*. URL: https://linux.die.net/man/8/torsocks.

[77] *MAMP*. URL: https://www.mamp.info/en/.

[78] *XAMPP*. URL: https://www.apachefriends.org/it/index.html.