Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria "Enzo Ferrari"
Tesi di Laurea in Ingegneria Informatica

Darkcloud: una darknet in chiave cloud.

Relatore:
Prof. Michele Colajanni

Correlatore:
Ing. Fabio Manganiello

Tesi di Laurea di
Gionatan Fortunato

Anno Accademico 2011-2012

Indice

Capitolo 1

Introduzione

A livello globale si può osservare una crescita esponenziale in termini di numero e gravità di attacchi informatici. Il problema però non tocca solo le grandi aziende, questo era vero una volta, perché solo le aziende di una certa dimensione avevano server interni e un sistema informatizzato raggiungibile dell'esterno. Oggi non esiste azienda che non abbia un calcolatore collegato ad internet. Nella stragrande maggioranza dei casi si hanno uffici con reti di computer, sedi dislocate per il mondo collegate tra di loro tramite lan, server interni per rendere disponibile materiale a consulenti e molto altro ancora. Si compie un grave errore pensando che solo le aziende che trattano materiale sensibile siano in pericolo. Per esempio aziende concorrenti potrebbero voler rubare tecnologie e brevetti aziendali, consultare liste clienti e fornitori, male intenzionati potrebbero infettare le macchine per usarle come zombie nelle proprie botnet.

Oggi più che mai bisogna rendere consapevoli le aziende della situazione nella quale il mondo dell'informatica si sta evolvendo.

La sicurezza delle organizzazioni più importanti risiede nel livello di protezione delle proprie informazioni.

Ci sono molti metodi per proteggere documenti informatici. Alcuni prevedono che il dato sia protetto fisicamente, altri che sia protetto da password o ancora sia raggiungibile dalla rete ma dietro solidi firewall.

Nella società attuale però non ci si può permettere di avere importanti moli di dati che non siano fruibili da persone autorizzate sparse per tutto il globo. Quindi la disponibilità in rete è un requisito fondamentale, che espone però i dati a gravi rischi. Riporre le proprie speranze in soluzioni monolitiche come pesanti crittografie o enti che raccolgono molti dati si è rivelato nel tempo una soluzione non vincente.

Dovendo destreggiarsi nella rete è conveniente sfruttare quelli che sono i suoi punti forti, come ad esempio l'enorme quantità di nodi che la compongono e la semplicità di scambio dei dati.

La condizione necessaria che un software di condivisione deve soddisfare è proteggere il file da utenti male intenzionati che potrebbero volersene appropriare.

La soluzione a questo problema è garantita da tre fattori, frammentazione del file, darknet e cloud computing.

Questi tre elementi non sono abitualmente usati insieme. Il cloud computing sta vedendo un forte momento di crescita e le sue implementazioni stanno crescendo molto. Però molte aziende che offrono servizi online usano i dati degli utenti per ricerche di mercato e altri motivi che non tutelano la privacy. Per tale motivo non è consigliabile usare tali servizi per dati confidenziali. Esistono strumenti di anonimato e reti di condivisione anonima che proteggono gli utilizzatori come le darknet ma sono spesso poco pratici a causa di tempi di condivisione troppo lunghi che rendono impraticabile

scambi di file di grosse dimensioni o scambi frequenti con altri utenti. L'utilizzo della frammentazione dei file oggi è molto diffusa per aumentare le velocità di condivisione. Questo strumento può anche essere utilizzato per aumentare la sicurezza. Unire questi elementi in un unico software permette invece di avere uno strumento innovativo ed estremamente sicuro.

1.1 Obiettivi del software

L'obbiettivo primario del software è la sicurezza dei dati che l'utente condivide con gli altri utilizzatori della sua rete. Vi sono anche aspetti che si rendono necessari legati all'architettura e al funzionamento del software cioè un codice con forte modularità e scalabilità.

Darkcloud è un software di condivisione file, cioè un programma che installato su almeno due computer e almeno un server permette ad ogni utente di salvare file sul server e condividere il file con l'altro utente. Il dato viene frammentato e salvato in vari nodi server sparsi sul web. Si ricorre ad un tipo di rete molto utilizzato nel'underground digitale, che viene definita *darknet*. In questo tipo di rete solo chi ne fa parte sa da chi è composta la rete.

Questo documento si pone l'obbiettivo di guidare il lettore in tutto il ciclo di creazione del programma Darkcloud, dallo studio fino alla fase di test.

Nel capitolo due viene analizzato lo stato in cui si trova al momento la ricerca nel campo delle reti di condivisione, inoltre viene fatta un analisi dei protocolli più utilizzati sottolineandone pregi e difetti.

Nel capitolo tre vengono motivate le scelte fatte in sede di progetto, si illustrano la struttura e il funzionamento del software. Inoltre si descrive la topologia della rete che sta dietro Darkcloud.

Il capitolo quattro contiene la descrizione di basso livello del software. Nei primi paragrafi si parla degli strumenti utilizzati come la crittografia, i database e le tecniche di connessione. Si procede all'esposizione dettagliata del funzionamento dei comandi e delle meccaniche che stanno dietro il software.

Il capitolo cinque tratta i test eseguiti al termine della fase di programmazione, comprende la descrizione degli ambienti usati per il testing, i risultati sperimentali e un analisi della scalabilità del software.

Il capitolo sei include considerazioni su tutto il progetto chiarendo i risultati ottenuti e la posizione in cui questo software si pone nella ricerca moderna. Include inoltre alcune prospettive per futuri sviluppi di ricerca.

Capitolo 2

Stato dell'arte

In questo capitolo vengono esposti i protocolli che oggi rappresentano lo standard de facto nella condivisione dei file. Anche se molti sono conosciuti come programmi usati per scambio di materiale multimediale i protocolli che vi sono alla base sono utilizzati da tutti i programmi presenti sul mercato usati dai professionisti per condividere file.

2.1 Napster

Napster[20] è stato il primo software di p2p ad avere una diffusione di un certo livello. Nacque nel 1999 da un progetto di Shawn Fanning. Non si tratta di un sistema p2p puro in quanto manteneva una lista di tutti gli utenti connessi e dei loro file su dei server centrali raggiungibili da tutti. Quindi se un nodo voleva trovare un file mandava la richiesta al server centrale che rispondeva con la lista dei nodi che lo avevano e il richiedente poteva fare partire il download da tali nodi. Quando un utente decideva di condividere un file mandava i dettagli al server centrale che li salvava. Quindi le condivisioni vere e proprie avvenivano tra gli utenti come un programma di istant messaging avanzato. I vantaggi di questo tipo in questo tipo di struttura risiedono nella velocità di reperibilità delle informazioni in quanto l'elenco dei file era disponibile a tutti direttamente. Altro punto a suo favore era la velocità di condivisione, in quanto i nodi si scambiavano i file creando una connessione diretta tra di loro, e il file era in chiaro e completo. Quelle che sono le sue caratteristiche positive in prima analisi si rivelano i suoi maggiori punti deboli se visti dalla prospettiva di utenti male intenzionati. Per esempio se il server centrale cade, tutta la rete smette di funzionare. Oppure se un ente vuole proibire la diffusione di certo materiale gli basterà confiscare il server per sapere chi ne era il detentore e con chi lo ha condiviso. La connessione diretta tra gli utenti porta inoltre tutta una serie di punti deboli come la facile trasmissione di infezioni informatiche, l'impossibilità di reperire un dato se il suo condivisore è offline e la mancanza di certificati di autenticità permette ad utenti di modificare contenuti e re immetterli in rete.

2.2 Gnutella

Gnutella[19] a differenza di Napster è un software peer to peer puro. Cioè non ha alcun server che centralizza alcun tipo di informazione. I computer degli utenti sono gli unici nodi che compongono la rete. Quando un certo numero di utenti installa il software sulla propria macchina automaticamente Gnutella riconosce i nodi vicini e in questo modo la rete si amplia. Le ricerche avvengono secondo la tecnica del flodding,

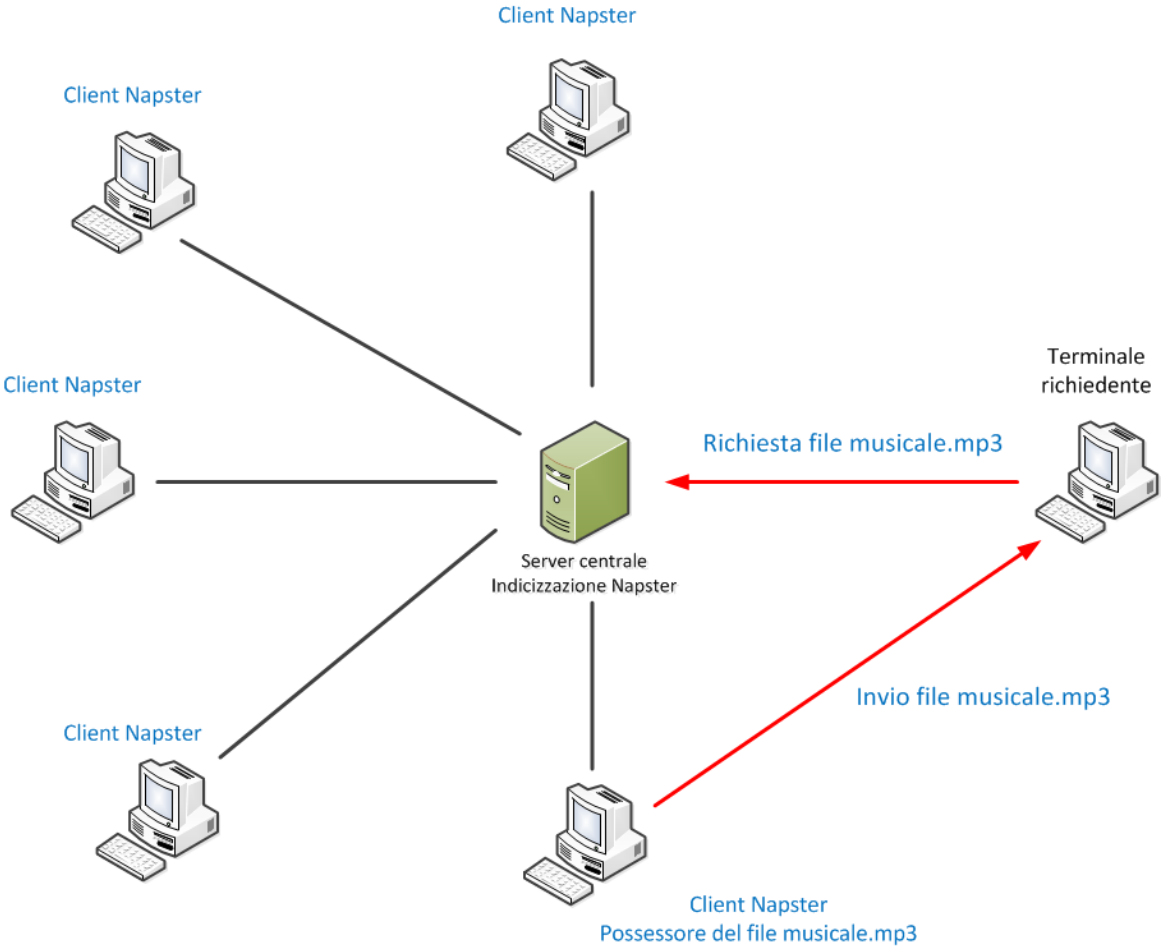

Figura 2.1: Architettura di Napster

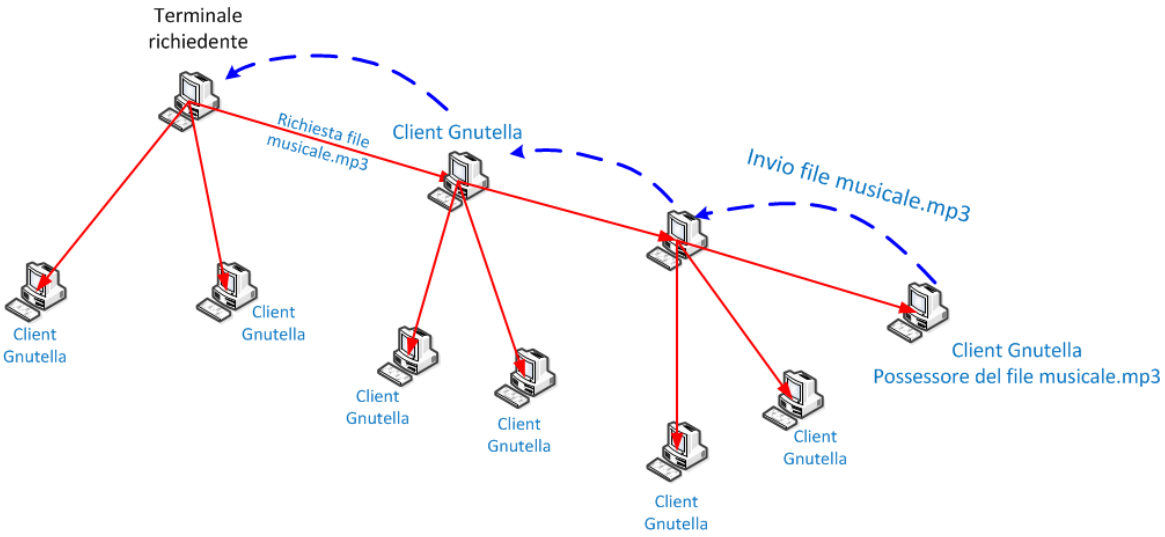

Figura 2.2: Architettura di Gnutella

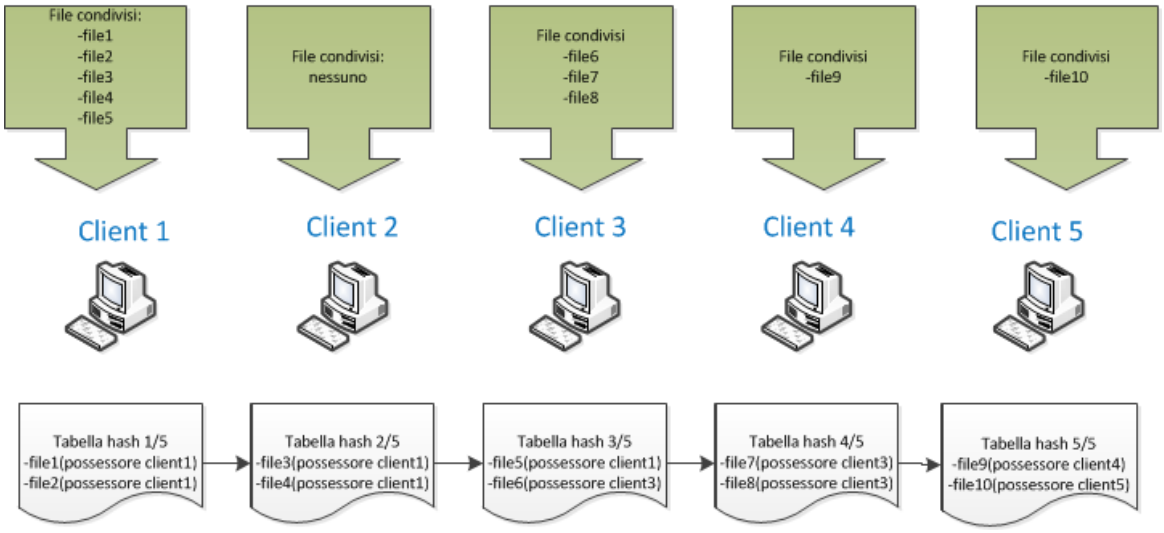

Figura 2.3: Indici dei file in Kademlia

ovvero quando un nodo deve reperire un file manda una richiesta ai nodi vicini, se i nodi vicini non hanno il file richiesto inoltrano a loro volta a tutti nodi che conoscono la stessa domanda tranne al nodo che gli ha mandato la richiesta originale. Per evitare il collasso della rete le richieste avevano un numero massimo di 'salti' tra nodi, al termine del quale venivano cancellate. Anche i trasferimenti di file vengono sostenuti dai nodi intermedi tra richiedente e possessore. Questo protocollo ha il vantaggio di essere potenzialmente inarrestabile, fortemente stabile e molto dinamico permettendo a nodi di entrare e uscire senza modificare le prestazioni della rete. Non teme quindi attacchi che ne arrestino il funzionamento. La tecnica di ricerca risulta molto rudimentale, non garantisce di trovare un file e i tempi per trovarlo comunque possono essere molto lunghi.

2.3 Kademlia

E' un protocollo di rete peer to peer decentralizzato ideato da Petar Maymounkov e David Mazières della New York University[6]. E' il primo protocollo analizzato che sfrutta la tecnologia delle DHT ovvero Distributed Hash Tables. La tecnica delle DHT nasce dalla collaborazione di alcune comunità di file sharing e viene subito adottata in diversi software[21]. Il metodo di mantenimento dell'indice dei file presenti sulla rete, DHT appunto, è il suo punto forte. Fino a quel momento il reperimento dei file era totalmente centralizzato mantenendo le liste in uno o più server oppure totalmente distribuito il che rendeva necessario contattare moltissimi nodi per trovare i file. Per capire come funzionino invece le DHT si immagini una lista di tutti i file presenti in una rete p2p, ora si frammenti questa lista in tante parti e le si distribuisca ai nodi che la compongono. Questi pezzi di lista sono replicati per permetterne il reperimento anche quando alcuni nodi sono offline. Ogni nodo oltre ad avere il suo pezzetto di lista sa anche in che direzione la lista si propaga, in questo modo se un nodo a lui vicino gli chiede dove si trova un file, e lui non lo sa, potrà indicargli in che direzione continuare a cercarlo. Una volta che il nodo richiedente trova nella lista il file che vuole, legge gli ip dei nodi che hanno il file e procede al recupero del file. La pubblicazione 'A distributed hash table'[5] di Frank Dabek, professore in Computer Science del MIT tratta in modo approfondito ed accurato le DHT. Nelle DHT il reperimento dei file è molto più veloce

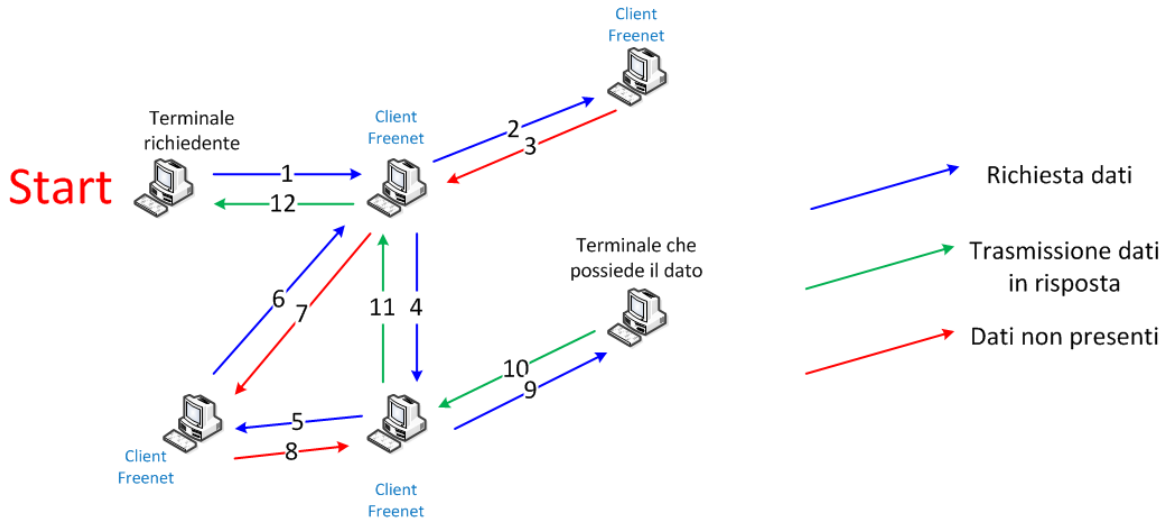

Figura 2.4: Esempio protocollo Freenet

di quanto non lo sarebbe in una rete completamente decentralizzata dove è necessario fare rimbalzare la richiesta verso moltissimi nodi. E' anche più sicuro del metodo centralizzato, non solo dal punto di vista della privacy ma anche di stabilità della rete stessa e di reperibilità dei file. Dato che non esiste nessun tipo di centralizzazione e i nodi sono responsabili dei file che caricano però, se tutti i nodi che conoscono un file si spengono l'intera rete non riuscirà più a raggiungere quel file.

2.4 Le darknet e Freenet

Le darknet sono reti per condividere file nate per tutelare la privacy e combattere la censura. Infatti in alcuni paesi lo stato proibisce di condividere file che dicano cose che non sono da esso approvate, e per permettere la libera diffusione di informazioni alcuni Informatici hanno pensato ad un modello di rete altamente dinamico e anonimo. Il termine darknet è stato coniato prima della nascita di internet nel 1970, quando l'unica rete riconosciuta era ARPANET e identificava una rete che fosse isolata da ARPANET per motivi di sicurezza. Questo non vuol dire che ne fosse separata, infatti i terminali che appartenevano a quelle darknet potevano ricevere ed inviare dati attraverso ARPANET, ma non risultavano registrati da nessuna parte e non potevano essere rintracciati in quanto non rispondevano ai segnali che chiedevano di identificarsi. Con il tempo e l'avvento di internet si iniziò ad usare il termine darknet per indicare reti private che passavano per internet usate da alcuni utenti per scambiarsi file, ma sempre e soltanto in piccoli gruppi che si conoscevano. Questo termine prima sconosciuto ai profani è stato dato in pasto ai media a seguito della pubblicazione del documento 'The Darknet and the Future of Content Distribution' [1] scritto da quattro ricercatori della Microsoft. Questo documento elencava tre punti essenziali che permettevano di identificare una rete come darknet:

1. ogni oggetto distribuito deve essere condiviso sulle macchine di una parte degli utenti, in una forma che ne consenta la copia

2. tutti gli utenti possono ottenere una copia dei file se ne hanno interesse

3. tutti gli utenti devono essere connessi tramite banda larga

Questo documento fu condiviso su larga scala e interpretato dai tutori del diritto di autore come il principale impedimento alla diffusione di materiale protetto in forma elettronica. Quindi si può spesso sentire parlare di programmi di condivisione come furono Napster, Gnutella, Kademlia e molti altri sotto la definizione di darknet da parte dei media, ma in realtà non è così. Questi sono programmi di p2p, peer to peer, differenti dalle darknet a causa della accessibilità delle reti a chiunque installi semplicemente il software. La tipologia di rete che si avvicinava di più al concetto di darknet erano le reti f2f, friend to friend. Ovvero reti dove si era collegati solo a nodi fidati e conosciuti, tramite porte e protocolli non standard. Il concetto di darknet però è stato ulteriormente rivoluzionato attorno al 2005 con la nascita di reti p2p ANONIME. Reti nelle quali è possibile condividere file senza il rischio di essere rintracciati. Proprio per questa caratteristica le reti darknet sono molto criticate, perché oltre che per preservare la libertà di informazione possono essere usate per condividere materiale illegale e dannoso.

Al momento molti ricercatori stanno studiando protocolli alternativi, che integrano complicati metodi di routing per garantire l'anonimato e al tempo stesso mantenere performance adeguate. Si può menzionare il protocollo R5N[3] sviluppato al TUM, Università tecnica di Monaco, che utilizza le DHT e metodi di randomizzazione delle comunicazioni tra nodi, o ancora un programma fatto all'università svedese di Uppsala che si chiama OneSwarm[12] nel quale si è pensato di realizzare una rete bitTorrent usando dei nodi di cloud computing.

Il programma studiato nella categoria darknet è Freenet[2]. Questa scelta è stata dettata dalla grande comunità che gli sta alle spalle, dal fatto che sia disponibile il codice sorgente e dalle caratteristiche che più si avvicinavano al modello Darkcloud. Molto utile in questo senso è stato lo studio della pubblicazione 'Beyond Simulation: Large-Scale Distribuited Emulation of P2P Protocols'[4], prodotta da due ricercatori tedeschi, nella quale si evidenza la forte differenza di scalabilità a seconda del tipo di topologia di una rete. Freenet è una rete decentralizzata dove non ci sono server e tutte le informazioni sono distribuite sui nodi. E' una rete realizzata puntando ad un alto livello di sicurezza ed anonimato. I file sono salvati sui nodi in modo crittografato e replicati su diversi nodi. Questo fa si che chi ha installato il programma non conosca cosa stia condividendo. Non esiste una struttura gerarchica tra i nodi, i collegamenti si basano sulla vicinanza, ogni nodo conosce solo quelli vicino a lui. Quindi quando vengono aggiunti nuovo nodi non si può prevedere a chi saranno collegati. Ogni file è contraddistinto da un id univoco, e il meccanismo per gestire l'indicizzazione dei file è simile alle DHT però invece che basarsi su una lista dei file che un nodo possiede si basa sulla velocità con la quale un nodo può avere un file. Dopo che un nodo ha trovato il file la richiesta passa di nodo in nodo, fino a raggiungere il nodo con il file. In seguito il nodo possessore del file lo invia, e il file viene trasmesso tra tutti i nodi intermedi. Quando vengono caricati nella rete i file non rimangono nel nodo che li ha condivisi, ma vengono replicati in modo da essere disponibili ad una certa velocità per tutti i nodi della rete. Dato che il modo per indicizzare i file si basa sulla velocità alla quale un file è disponibile questo vuol dire creare una ridondanza dei file su tutta la rete in modo da essere disponibili più velocemente per tutti! Oltre al funzionamento sulla topografia sopra descritta, cioè sfruttando tutti i pc con installato Freenet presenti sulla rete internet, il software offre una modalità di funzionamento friend to friend. In questo setting è necessario specificare una lista di nodi manualmente questi saranno gli unici con cui andremo a condividere dei file. E' una opportunità in più tramite la quale si condivide solo con i propri amici. Può essere utile per esempio per alcuni gruppi di utenti che condividono file riservati e non vogliono renderli disponibili a tutta la rete

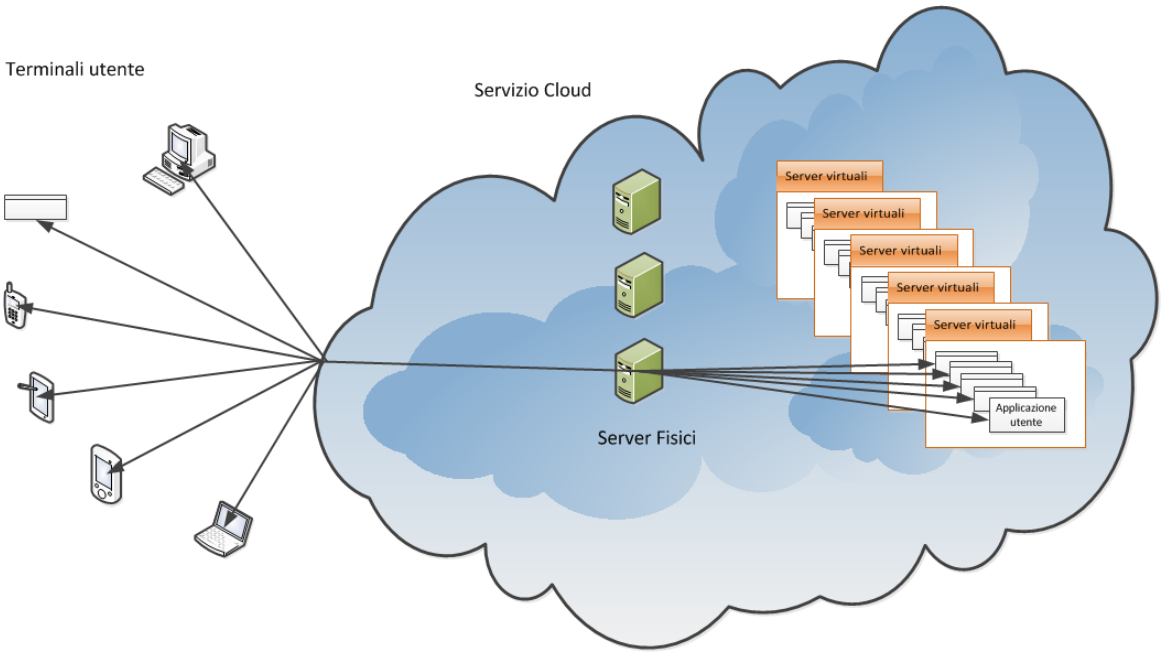

Figura 2.5: Struttura Cloud Computing

Freenet. Ogni caratteristica di questo programma è atta ad aumentarne l'anonimato e si può constatare che funziona molto bene. Tuttavia queste scelte sono state fatte essendo consapevoli che i file condivisi sarebbero stati documenti di piccole dimensioni. Questo rende lenta e difficile la gestione di file di dimensioni maggiori.

2.5 Cloud computing

Di cloud computing si sente parlare già dal 1960, quando lo scienziato informatico John McCarthy ipotizza un futuro nel quale la capacità di calcolo sarebbe stata organizzata come un servizio di pubblica utilità. Naturalmente i primi passi furono ben distanti, infatti le grandi aziende che con gli anni iniziavano ad usare data center avevano delle macchine che erano dedicate a sopperire solo alle richieste di un cliente. Gli studiosi si accorsero che per la maggior parte del tempo però le macchine lavoravano al minimo, registrando picchi di calcolo solo in certi momenti della giornata. In questo modo vi era un enorme spreco di energia elettrica e di risorse. La prima multinazionale che notati questi sprechi si mise al lavoro per trovare una soluzione fu la americana Amazon. Infatti nel 2006 presentò Amazon Web Service[14] un insieme di servizi web. All'interno dei servizi proposti ve ne erano diversi di cloud computing, per esempio servizi che offrivano spazio online dinamico e veloce con costi in base al consumo, servizi che offrivano sistemi operativi virtuali utilizzabili online con costi basati sul tempo di utilizzo e server per applicazioni web nel quale il costo variava in base alle ore che il servizio era operante. Si può dire che è stata proprio questa la nascita del vero e proprio cloud computing ovvero servizi web che offrono spazio di archiviazione, capacità computazionale, sistemi operativi, hardware e banda in base alle esigenze in modo dinamico. Come la rete di distribuzione idrica o del servizio elettrico, dove si paga quello che si consuma.

Negli ultimi tempi si è vista una esplosione di questo tipo di servizi, ormai quasi tutti i provider hanno a listino un servizio di cloud. Viste le potenzialità molte aziende del calibro di Apple e Microsoft stanno utilizzando questa tecnologia per dare un servizio

importante ai propri utenti come il centralizzare i dati personali tra i tanti terminali che oggi si hanno, notebook, desktop, smartphone, tablet. E' facile prevedere che il cloud cambierà l'informatica da come la si conosce adesso.

Il cloud computing è una risorsa molto preziosa visti i moderni strumenti che mette a disposizione, non solo perché offre una grande potenza di calcolo o di spazio di archiviazione con un basso costo vista la dinamicità, ma anche perché i moderni strumenti usati dal cloud per virtualizzare permettono di avere un qualsiasi modello di hardware subito e di fare le modifiche necessarie a piacimento. Dato che la rete Darkcloud deve rendere i dati sempre disponibili, non è possibile scegliere una struttura di rete che mantenga il dato sui client, ciò porterebbe molti rischi. Invece la tecnologia del cloud computing permette di avere una serie di macchine sempre online e con uno spazio di archiviazione e una banda passante di capacità che va ben oltre le necessità del caso studiato.

I servizi che vengono offerti nel campo del cloud computing si possono dividere in tre tipologie. I servizi SaaS ovvero Sofware as a Service, PaaS Platform as a Service o IaaS Infrastructure as a Service. Il servizio IaaS permette l'utilizzo di risorse hardware in remoto, servizio simile al Grid Computing ma con una caratteristica imprescindibile: le risorse vengono elargite in base all'utilizzo effettivo. Tramite un servizio IaaS per esempio si possono avere diversi sistemi operativi attivi su server remoti o ancora servizi di storage online nei quali viene dato spazio di archiviazione. I servizi PaaS permettono di eseguire in remoto una piattaforma software formata da diversi programmi. Infine i servizi SaaS consentono l'utilizzo in remoto di singoli programmi. I programmi utilizzabili su SaaS però devono essere realizzati con ottimizzazioni per il servizio infatti ogni provider di servizi cloud ha una propria serie di librerie, che contengono i comandi ottimizzati per le loro infrastrutture. Questo offre il vantaggio di avere software molto veloce ed efficiente, essendo ottimizzato, d'altra parte però obbliga a sviluppare software ad hoc per ogni diverso servizio di cloud computing. Per esempio le API del servizio cloud di Google sono diverse da quelle di Amazon che sono ancora diverse di Microsoft. E' importante realizzare Darkcloud in modo che sia facile in un secondo momento adattarlo a diversi servizi di cloud, e questo è possibile utilizzando l'incapsulamento e la modularità nel codice in modo da renderlo facilmente riutilizzabile.

Lo stato attuale dell'utilizzo del cloud computing è molto avanzato, infatti la quasi totalità del siti web ora sono ospitati sul web sono su server che sfruttano tale tecnologia. Oltre a questo molti software che hanno una parte web sono ospitati su questi servizi. Molti ancora riservano dubbi a riguardo della sicurezza in questi campi, ma secondo l'opinione di diversi ricercatori[13][15] il cloud offrirà possibilità che favoriranno molto la sicurezza, la differenza sarà nelle implementazioni.

Capitolo 3

Progetto

In questo capitolo si procederà a illustrare le scelte progettuali, il funzionamento a livello logico funzionale del software e della rete che sta dietro Darkcloud.

3.1 Obbiettivi di dettaglio

L'avvento del cloud computing ha dato uno strumento molto potente da poter sfruttare per questo progetto, lo studio delle tecnologie di condivisione di file peer to peer ha dato una visione di insieme di quello che nel corso del tempo è stata l'evoluzione di queste reti. Questo studio ha permesso di avere molti spunti per poi tracciare quella che sarebbe stata la struttura di Darkcloud. La dinamicità di alcune reti p2p pure, ad una prima analisi pare una ottima soluzione, visto che avrebbe permesso di realizzare reti che si ampliassero in modo automatico e quindi non sarebbe stato necessario ogni volta che si aggiungeva un server o un client riconfigurare tutti i nodi. Il problema della sicurezza però in una soluzione del genere renderebbe molto più complicato verificare che i nodi aggiunti siano nodi fidati. In una architettura p2p pura inoltre il carico di lavoro è equamente distribuito su tutti nodi in maniera indistinta, questo però nel caso Darkcloud non sarebbe stato adatto, in quanto avendo a disposizione una grande potenza di calcolo e grande larghezza di banda da parte dei nodi cloud contenuti in grandi data-center si sarebbe inutilmente sobbarcato di lavoro i calcolatori degli utenti ultimi che devono invece solo caricare file e poi scaricarli. Un altro aspetto che ha spinto a scartare l'approccio p2p puro è il carico di lavoro che avrebbe portato realizzare una soluzione altamente dinamica rispetto ad una soluzione statica. La programmazione di un software dinamico richiede molte più parti e una complessità totale molto maggiore. Quindi bisogna considerare a chi è indirizzato il software, quale sarà il suo bacino di utenza, se si intende farlo crescere in futuro e cosi via cercando quindi di impiegare la giusta quantità di risorse. Non ha senso realizzare un software e una architettura che sopporti migliaia di utenti quando è specificamente richiesto che funzioni per poche decine di persone.

Oggi non è difficile utilizzare un server centrale per concentrare tutte le informazioni e i dati di una azienda, anzi è proprio quello che fanno la maggior parte delle grandi, medie e piccole realtà. Anche in Darkcloud si sarebbe potuto adottare una soluzione di condivisione con server centralizzati, questo avrebbe reso più veloce la rete e avrebbe permesso la condivisione di file di più grosse dimensioni, portando una spesa in termini di sviluppo e mantenimento molto inferiore. Essendo la sicurezza il punto centrale del progetto questa possibilità è stata scartata a priori. I rischi di tenere i dati tutti si alcuni server sono troppi. Per esempio affidando tutti i dati ad una compagnia di server si rischia nell'eventualità che la compagnia abbia problemi tecnici o subisca attacchi

che i dati rimangano inaccessibili per un periodo di tempo o addirittura vengano persi. Ancora se la sicurezza della azienda venisse compromessa si rischierebbe che male intenzionati si approprino dei dati e che questi vengano diffusi.

La tecnologia delle DHT è molto interessante in quanto rappresenta un compromesso tra il modello a server centralizzati e il modello totalmente distribuito. Però porta una difficoltà di programmazione notevole e un carico di lavoro dei nodi distribuito non in modo ottimale.

Freenet è il programma che più assomigliava all'idea di Darkcloud. La modalità di lavoro friend to friend avrebbe permesso di usare solo nodi specifici, la sicurezza era nettamente al di sopra di quella necessaria, l'informazione in Freenet viene spezzettata in tante parti e inviata con ridondanza a nodi diversi, cifratura dei file e delle connessioni. A fronte di queste considerazioni si era preso in considerazione di utilizzare il codice di Freenet come base per sviluppare Darkcloud. Considerando che per sfruttare la topologia Darkcloud sarebbe dovuta essere di due livelli, server e client, inoltre che i server non dovevano comunicare tra di loro è stato chiaro che le modifiche necessarie sarebbero state più impegnative che sviluppare da zero un programma con le specifiche ormai delineate.

Il software doveva essere prima di tutto realizzato con l'obbiettivo di funzionare ottimamente su servizi di cloud computing. Basandosi su una topologia a due livelli, client e server, in modo da sfruttare le potenzialità offerte dal cloud e alleggerire il carico dei nodi client. Non doveva basarsi su un solo server ma su diversi in modo da poter attuare una splitting dell'informazione verso questi ultimi.

Data l'opportunità di programmare da zero il software si è deciso di renderlo più adattabile a future espansioni dal lato client. Invece che dare una classica veste grafica dalla quale gestire il client si è preferito creare un altra entità di interfacciamento con l'utente. Infatti il client non ha interfaccia grafica ma viene comandato da messaggi xml. Per utilizzare il client è stato creato uno script in python da usare come terminale per inviare i comandi. Questo è solo un esempio di terminale, ma è utile perché in futuro si potrà comandare il client tramite interfaccia web, piuttosto che con un applicazione per cellulari.

3.2 Struttura di rete

La rete Darkcloud è stata progettata con l'obbiettivo di proteggere i dati in essa condivisa.

La struttura di Darkcloud si compone di due tipi di nodi: i *server* e i *client*. I nodi *client* sono quelli che eseguono le operazioni che gli utenti inviano tramite i terminali. I nodi *server* sono i delegati al mantenimento del'informazione, costituiti da istanze di cloud computing. I client e i server possono collegarsi tra di loro sia tramite rete locale che tramite internet. La topologia della rete permette a diversi terminali dello stesso utente di collegarsi ad un unico nodo. *Fully connected* sul lato client, cioè tutti i nodi client possono comunicare tra di loro e con tutti i server. Invece il lato server può comunicare con tutti i client, ma i server non possono comunicare tra di loro. Questo semplicemente perché non è utile ai fini del loro compito.

Gli indirizzi dei vari nodi, sia client che server, sono specificati manualmente nella prima configurazione dei nodi. Quindi i nodi non si aggiungono dinamicamente. Dato le dimensioni contenute che la rete si propone di avere questo è stato più semplice e più sicuro di un approccio dinamico. Per identificare un nodo all'interno della rete si usa una chiave, che è un codice alfanumerico generato da una apposita funzione che ogni nodo conosce, funzione che ricevendo in ingresso l'ip e la porta del nodo restituisce in

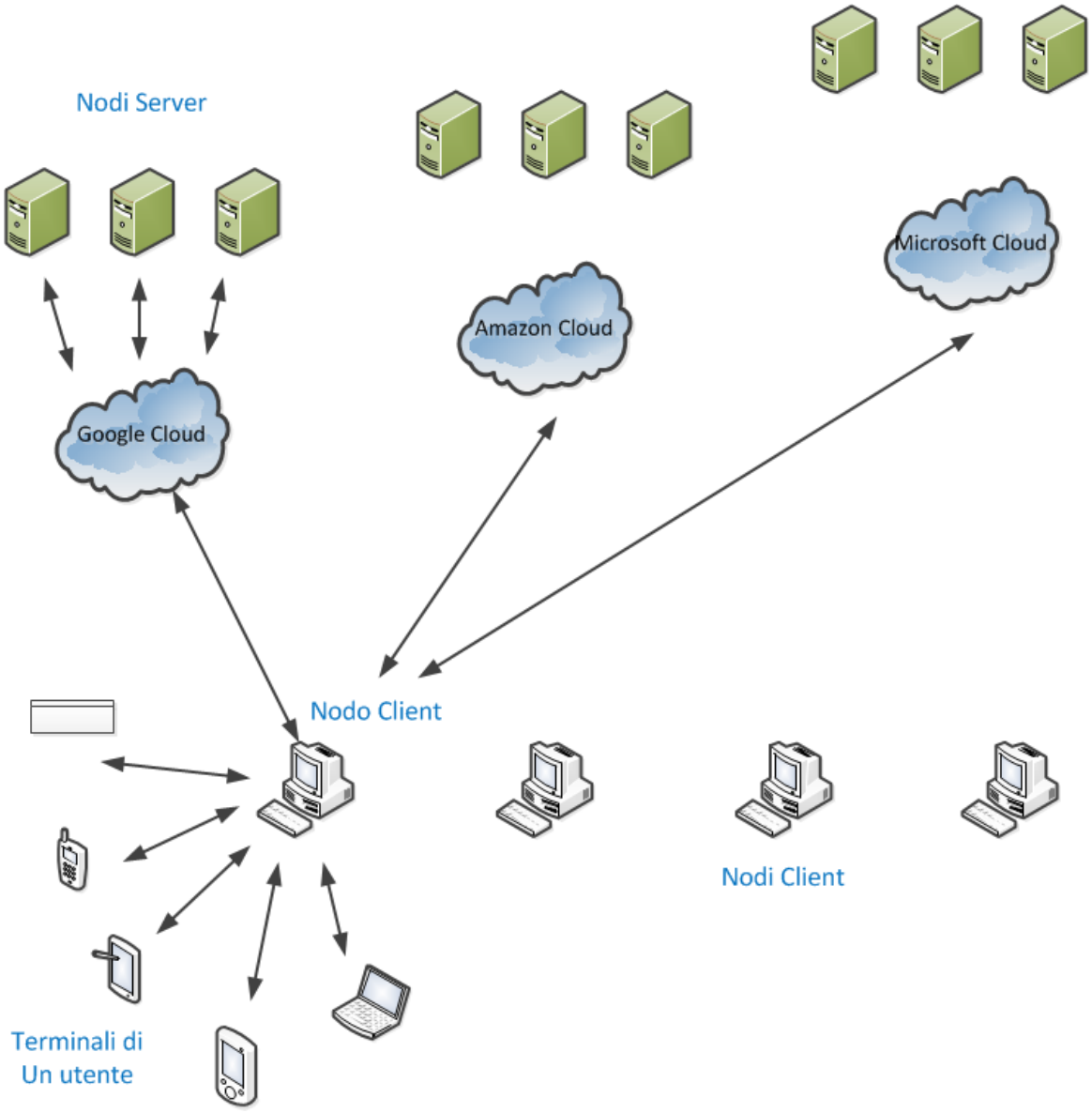

Figura 3.1: Architettura di Darkcloud

uscite la chiave di riconoscimento del nodo. In questo modo si crea un nuovo network virtuale sopra la rete internet.

Dopo aver avviato una certa quantità di nodi server e alcuni nodi client, il programma permette in modo trasparente di salvare un file nella rete. Questo file viene elaborato da un protocollo che si vedrà nel dettaglio nei paragrafi successivi. Questo protocollo spezzetta i file e ne manda una parte ad ogni server. Ora il nodo client ha riposto al sicuro il suo file, e può ricomporlo soltanto lui, quindi quello che può ora fare è condividerne la conoscenza con altri nodi o ad esigenza richiedere il file al programma. Di primo acchito le perplessità sulla sicurezza possono essere tante, ma si invita il lettore a leggere il paragrafo che parla della sicurezza del nodo.

I servizi di cloud computing che nella implementazione ultima andranno a mantenere i file dovranno essere molteplici. Al momento se qualcuno ha bisogno di un certo numero di server su cui fare girare il proprio software può chiederlo direttamente ad un unico fornitore di servizi cloud. Per esempio per salvare i file in 100 parti diverse si potrebbe chiedere ad un solo fornitore, uno tra Google Amazon Microsoft e altri, di caricare il software su 100 istanze che rappresentano sistemi operativi. A questo punto il file sarebbe salvato in parti sui server del fornitore scelto. Però questo non sarebbe astuto per tutelare i dati del professionista che li vuole mantenere al sicuro. In quanto essendo dati tutti a disposizione del fornitore nulla gli impedirebbe di ricomporre il file e di provare poi a decrittarlo. Se a questo punto il provider ottenesse i dati interi potrebbe usarli per ricerche di mercato o profilazione degli utenti. Essendo spesso questi provider di servizi in paesi diversi dall'utilizzatore il tutelarsi in tali problemi di privacy diventa complicato. Oppure se un attaccante che trovasse una falla nei sistemi dello stesso fornitore potrebbe fare lo stesso. Ancora se si affidano tutti i dati allo stesso fornitore in caso di malfunzionamento delle strutture i dati potrebbero essere irraggiungibili per un dato periodo di tempo. Per questi motivi si preferirà utilizzare diversi servizi di cloud computing per avere un certo grado di ridondanza dei dati e non lasciarli tutti nelle mani di un unico operatore.

3.3 Architettura software

Lo scopo di Darkcloud è quello di immagazzinare documenti in modo che nessuno che non abbia i permessi possa accedervi, permettendo anche di condividere il file tra più utenti.

Nel software si trovano 2 tipi di entità utilizzate per identificare i diversi nodi della rete. Le entità Darkcloud e le entità Netnode. Le entità Darkcloud sono quelle che ogni nodo istanza quando nasce. Invece le entità Netnode sono copie degli altri nodi, utilizzate per memorizzare in locale i dati degli altri nodi, come indirizzo, porta di ascolto e molto ancora.

Una nuova tecnologia che sta nascendo e di cui si sente molto parlare è il *cloud-computing*, grazie a questa tecnologia è possibile avere molte istanze di una applicazione senza dover per forza avere un server dedicato per ognuna di esse. A patto di utilizzare le API di quel determinato fornitore di servizi cloud. Infatti i metodi all'interno delle classi sono stati sviluppati tramite la *reflection*, metodo di programmazione che permette di creare facilmente comandi incapsulati, in modo da poterli in un secondo momento adattare alle varie API dei diversi servizi di *cloud-computing* senza dover toccare il resto del codice.

Vista la continua evoluzione delle soluzioni hardware portatili come smartphone e tablet si è pensato di fare evolvere anche il software in modo da poterlo fare collaborare con queste tecnologie. Infatti quando si avvia l'applicazione non è direttamente questa

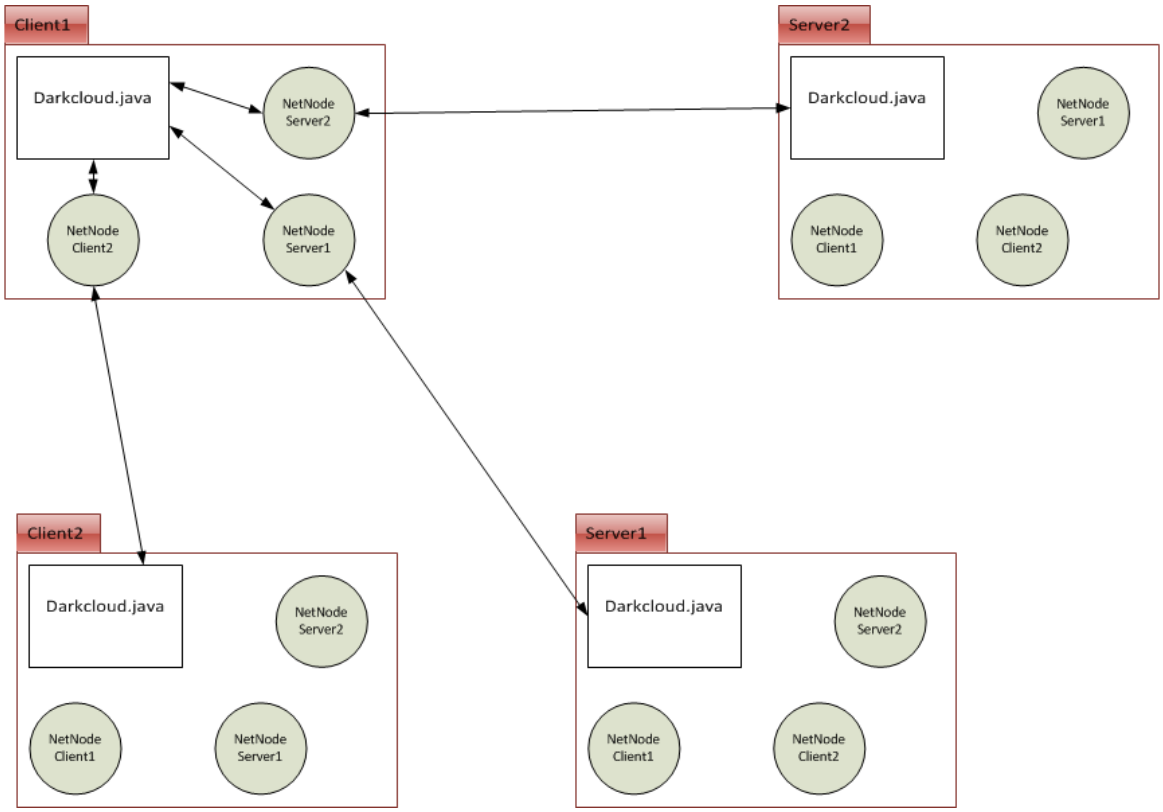

Figura 3.2: Darkcloud e NetNode

che si va ad usare per caricare i dati, ma attraverso terminali i quali poi contattano il client. Per esempio al momento è stato sviluppato uno script in python che si chiama client.py tramite il quale, dopo aver avviato il client, si può inviare ricevere e condividere file sulla rete. Adottando una soluzione del genere quando in un secondo momento, o a seconda di esigenze, si vorrà sviluppare una applicazione per tablet, o una applicazione web che permetta di usare la rete da browser o ancora integrare l'uso della rete in altri software non sarà necessario andare a modificare il programma client ma semplicemente creare una interfaccia. Per poter rendere questo possibile l'utente dovrà avere un pc connesso ad internet che gestirà tutte le richieste che le sue diverse interfacce inoltreranno. Un po come un convogliatore di richieste ed invii. Questo rende molto più semplice lo sviluppo di nuove soluzioni per usare Darkcloud.

3.4 Protocollo di comunicazione tra nodi

Le connessioni tra i vari nodi avvengono tramite connessioni criptate con il protocollo *SSL*. Il Secure Socket Layer, SSL, è un protocollo crittografico che permette una comunicazione sicura su reti TCP/IP, come reti lan o internet nel caso di Darkcloud. La cifratura avviene al di sopra del livello di trasporto. Questo protocollo consente alle applicazioni client/server di comunicare attraverso una rete in modo tale da prevenire il 'tampering' (manomissione) dei dati, la falsificazione e l'intercettazione. L'autenticazione è bilaterale, cioè entrambi le parti si autenticano scambiandosi i certificati. Certificati che vengono generati alla creazione del nodo. In seguito questi certificati si scambiano tra i diversi nodi e tramite uno script apposito vengono aggiunti tra i certificati riconosciuti come 'trusted' per la rete.

I messaggi che viaggiano tra i nodi criptati tramite SSL sono messaggi in formato XML. Questo formato è stato scelto perché permette in maniera molto semplice di aggiungere al messaggio campi e attributi.

I messaggi XML scambiati si dividono principalmente in due tipi: i messaggi Request e i messaggi Response. Quando un nodo vuole comunicare con un altro crea un istanza di un oggetto request, successivamente specifica di che tipo di richiesta si tratta, aggiunge i campi con le informazioni necessarie affinché il nodo ricevente possa soddisfare la richiesta, e tramite il metodo send invia il messaggio all'altro nodo. Il metodo send si preoccupa di creare la connessione con il secondo nodo e di raccoglierne la risposta cioè un messaggio contenente un oggetto response che contiene la risposta del secondo nodo. Infatti una volta che il nodo ricevente ha la richiesta si preoccupa di vedere di che tipo si tratta, e in base a quello esegue il codice corrispondente a quel tipo di richiesta. Elabora i dati contenuti nella richiesta, crea una istanza di response, vi inserisce i risultati e li invia al richiedente.

I tipi di messaggi che i nodi si possono scambiare sono:

- PING
- PUT
- GET
- SHARE
- RECEIVE

Il primo comando che si analizzerà è il PING. Questo tipo di request viene usato dai nodi per verificare se gli altri sono online e quanto sono distanti calcolando i tempi di latenza. Infatti è un comando che può essere usato anche dallo script client.py per verificare se i nodi client, tramite i quali si può mandare i file sulla rete, sono vivi. Ancora i nodi client effettuano un ping regolarmente verso i server in modo che quando devono caricare file sulla rete sappiano già quanti server sono disponibili e quindi in quante parti possono suddividere il file. Oltre a effettuare il ping regolare dei server si effettua anche quello degli altri client, in modo da effettuare le condivisioni dei file con loro in modo veloce.

3.5 Salvataggio e recupero dei file

Quando un client decide di caricare un file sulla rete Darkcloud, utilizza il comando PUT. Come si è accennato prima parlando dell'architettura del software, si ha un interfaccia o molteplici, tramite le quali si può comandare il client. Quando si invia da un interfaccia un comando per salvare sulla rete il file, dallo script client.py per esempio, si specifica il nome del file locale, il nome con cui si vuole che il file venga salvato e il client da usare. L'interfaccia specificherà quindi al client che si è indicato il contenuto del file e un comando di put tramite un messaggio XML, il client controllerà prima l'integrità del file poi lo cripterà, con specifiche che si vedranno in dettaglio nel prossimo capitolo. Dopo di che in base a quanti server il client avrà a disposizione frammenterà il file in tante parti quanti sono i server. Procederà poi a preparare tanti oggetti request quanto sono i frammenti, nei quali specificherà che si tratta di messaggi put, quale è il contenuto del file, il checksum per controlli di integrità e infine invierà la richiesta ai server con il metodo send. Accertandosi che i server rispondano in maniera affermativa di aver ricevuto il frammento e di averlo correttamente salvato.Per poter

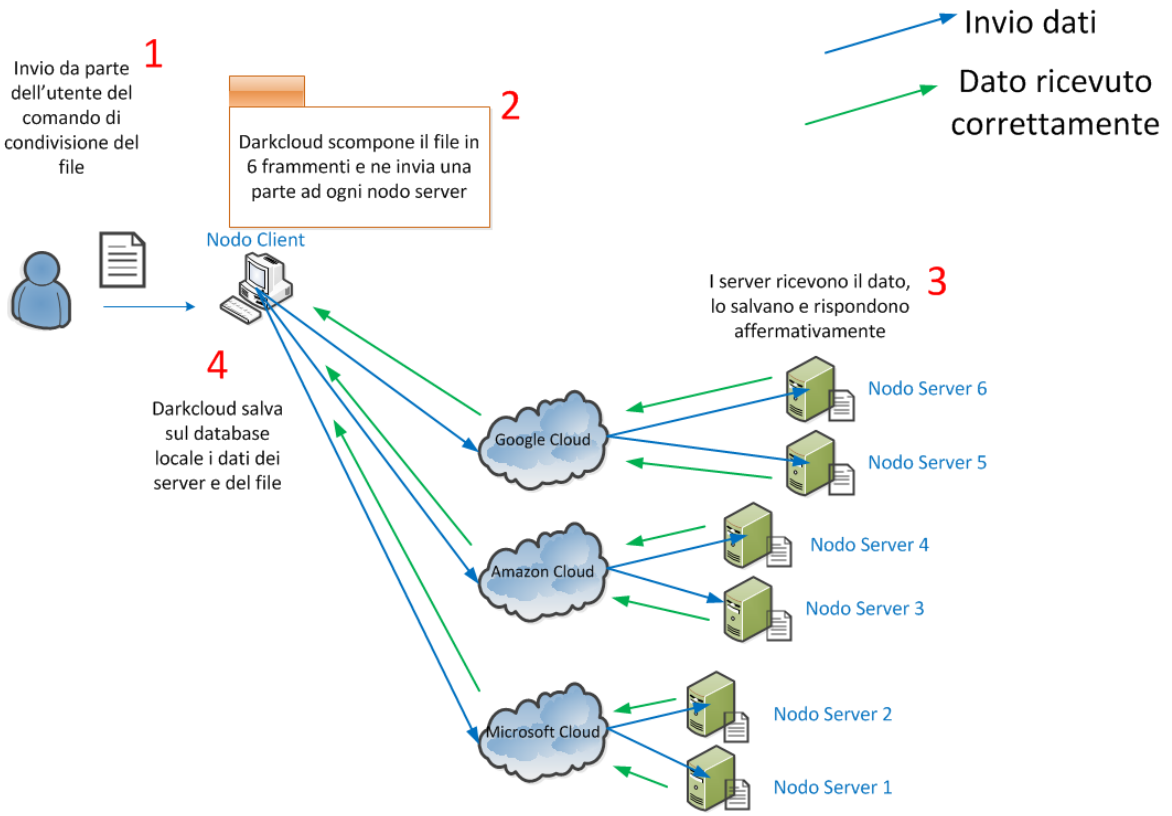

Figura 3.3: Invio di un file su Darkcloud

ricomporre il file il client salverà su un database interno i dettagli dei file e di ogni frammento, compreso il checksum di ognuno e il nodo server sul quale è stato salvato.

L'operazione di recupero viene effettuata dalle interfacce. Nel caso del client.py si deve specificare il client da usare, e il nome con il quale il file è salvato in remoto. Quindi l'interfaccia client.py invierà al client una richiesta di tipo GET specificando il nome del file. Il client a sua volta ricevuta la request recupera il campo con il nome e va a cercare nel sua database i dettagli di come e dove è stato salvato il file. Quindi su quali server, in quale ordine e quali sono i checksum delle singole parti. Una volta inviate le request ai server specificando i nomi dei frammenti, recupera le response mandate dai server. Ora può controllare l'integrità dei file calcolando il checksum dei frammenti ricevuti e confrontandolo con il checksum che aveva salvato. Se i dati sono corretti ricompone il file e lo decripta, inviandolo per concludere all'interfaccia che glielo aveva richiesto.

3.6 Condivisione dei file

Fino ad ora si è visto a grandi linee il funzionamento del salvataggio e del recupero del file, ma dato che questa rete deve permettere a persone di condividere tra di loro file è stato anche implementato un comando SHARE. Questo comando non va a toccare i server, in quanto non avrebbe avuto senso replicare i file sulla rete per ogni utente, infatti quello che si è permesso con il comando share è stata la condivisione della conoscenza di un comando con un altro nodo. Se un ipotetico utente1 volesse condividere un file con l'utente2 sarà sufficiente usare il comando share. Dato che la conoscenza e i dati sono salvati sul client e non sulle interfacce, sarà l'utente1 che chiederà tramite la

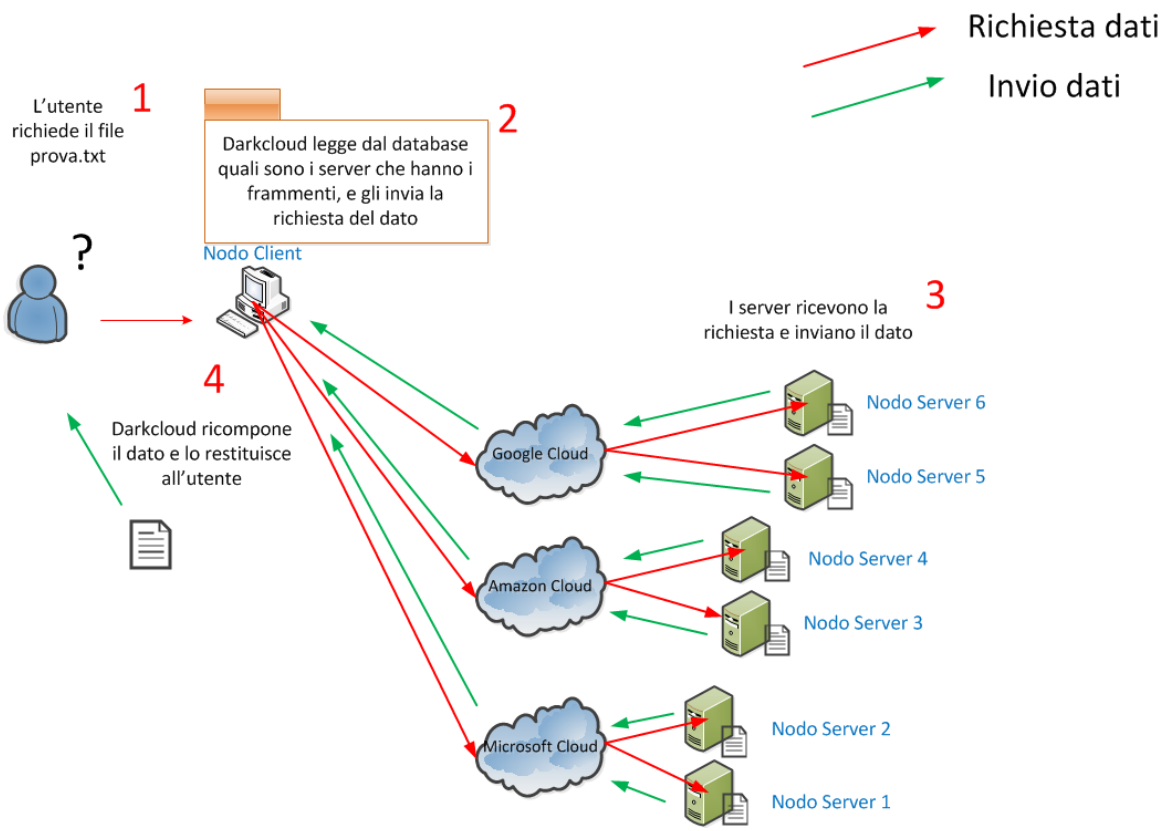

Figura 3.4: Recupero di un file da Darkcloud

sua interfaccia al suo client di inviare al client di utente2 i dati riguardanti un determinato file. In modo che in un secondo momento se l'utente2 vuole recuperare il file possa semplicemente chiederlo al suo client tramite la sua interfaccia. La procedura parte con l'utente1 che invia tramite la sua interfaccia un richiesta al suo client specificando il nome del file remoto, e i parametri del client di utente2, nel caso specifico di client.py per esempio si tratta dell'indirizzo ip e della porta di ascolto. Il client di utente1 andrà a recuperare dal suo database tutti i dati necessari a recuperare il file. Ora creerà un oggetto request di un tipo RECEIVE e gli allegherà tali dati. Il comando receive serve per salvare nel database locale i dati di un file, e dei suoi frammenti, che non è stato caricato nella rete dal nodo che esegue il comando. Quindi a seguito della ricezione dei dati e dell'esecuzione del comando receive il client dell'utente2 sarà in grado come l'utente1 di recuperare il file.

3.7 Sicurezza del nodo

E' vero che il mantenere l'indice di tutti i propri file in locale potrebbe essere un punto debole sul piano della sicurezza, ma proprio per questo si sono adottati una serie di accorgimenti per rendere difficile ad un attaccante, anche qual'ora avesse compromesso il nodo, reperire le informazioni. Questo è possibile grazie all'utilizzo combinato , di crittografica simmetrica, crittografia asimmetrica, invio dei dati ai server in modo casuale, utilizzo di comunicazioni criptate secondo il protocollo SSL, controllo in ogni operazione dell'integrità dei file tramite checksum e aggiunta di nodi in modo statico.

Infatti quando un file è stato spezzettato, ogni parte viene cifrata e poi inviata ai server. Se l'invio ai server avvenisse in modo sequenziale e ordinato, un utente che stesse

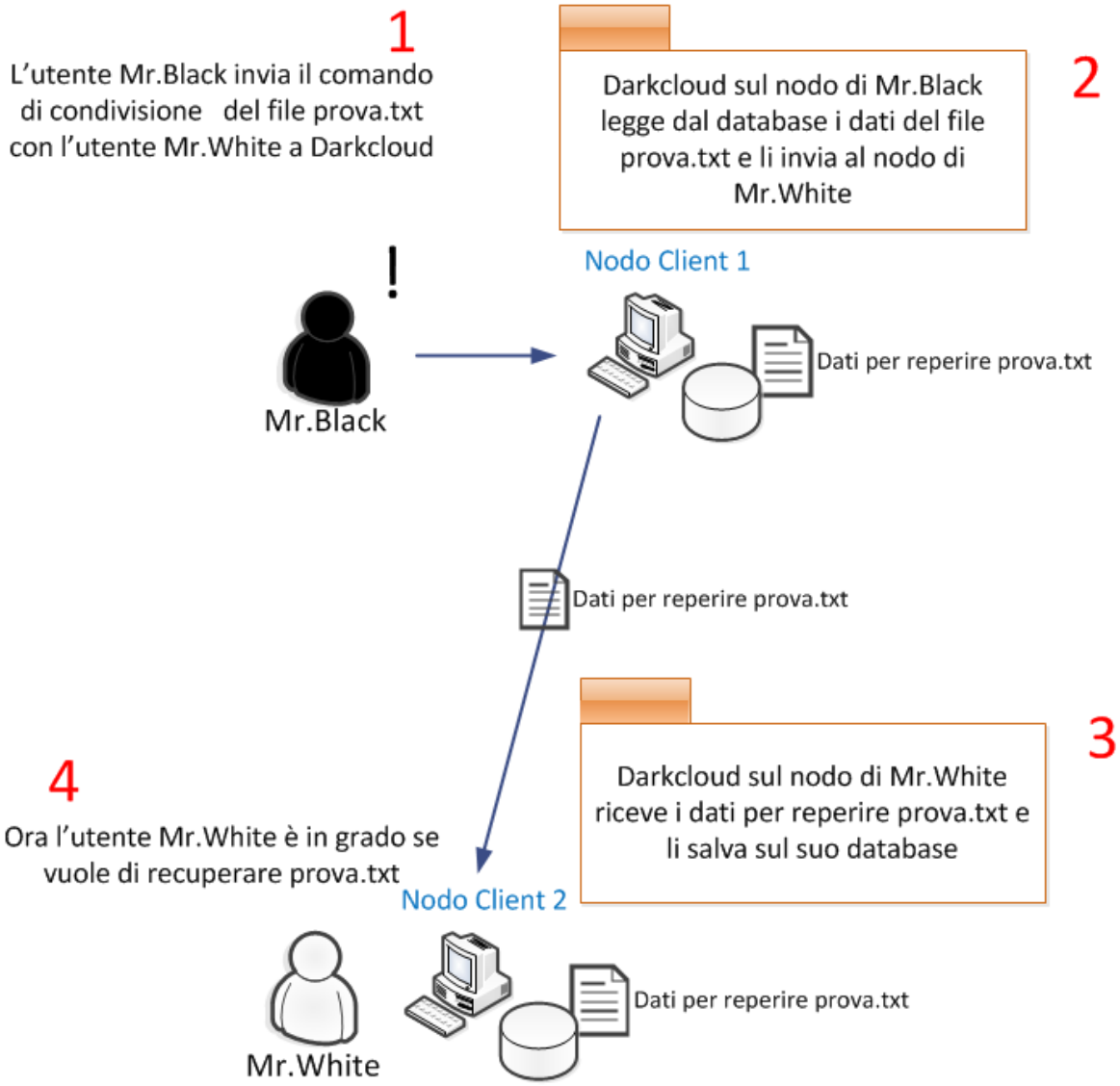

Figura 3.5: Condivisione di un file con un altro utente

sniffando la connessione riceverebbe i frammenti in ordine facilitandogli notevolmente il lavoro di ricostruzione. Ancora se inviassero i file in modo ordinato in base ai server chi avesse accesso a questi server potrebbe sapere in che ordine andrebbero riordinati. Per evitare ciò, dopo aver spezzettato il file e salvato sul database la sequenza esatta il programma mescola i frammenti tra di loro, solo dopo questo li invia ai server.

Per verificare che i dati che vengono scambiati tra i nodi non vengano contraffatti o subiscano modifiche a causa di errori di connessione si è fatto un forte uso dei controlli del checksum. Il checksum è una sequenza di bit che viene calcolata partendo da un file. Il checksum di un file è univoco, quindi se il file viene modificato il checksum non corrisponde più. Prima di inviare un frammento di file ad un server, ad esempio, un client ne calcola il checksum e lo salva sul database. Oltre a ciò salva anche il checksum dell'intero file. Quando in un secondo momento andrà a ricomporre il file, dopo aver recuperato i frammenti andrà a calcolare i checksum di tali frammenti e li confronterà con i checksum salvati. Stesso procedimento dopo aver ricomposto il file, ne calcolerà il checksum e lo confronterà con quello salvato. Questo aiuta anche ad evitare errori nelle procedure di ricomposizione dei file.

3.7.1 Crittografia

La crittografia asimmetrica, conosciuta anche come crittografia a coppia di chiavi, è un tipo di crittografia usata per condividere file tra due o più utenti. Questo tipo di crittografia permette agli utenti di scambiarsi informazioni in modo che un terzo utente non autorizzato non possa leggere il messaggio. Ogni utente in questo sistema possiede due chiavi di cifratura, una pubblica e una privata. La chiave pubblica viene data a chiunque voglia comunicare con lui, mentre quella privata rimane di esclusiva conoscenza del proprietario. Un messaggio cifrato con la chiave pubblica dell'utente1 può essere decifrato solo tramite la chiave privata dell'utente1. In questo modo se l'utente2 vuole mandare un messaggio cifrato all'utente1, lo cifrerà con la sua chiave pubblica, e solo l'utente1 potrà leggerlo. In Darkcloud si è usato questo metodo per cifrate le parti di file che vengono scambiate tra i nodi. Ogni nodo ha la sua chiave privata e quella pubblica, e inoltre tutti i nodi conoscono le chiavi pubbliche degli altri. In questo modo quando si condividono parti di file tra nodi client e server un male intenzionato che intercettasse i pacchetti non riuscirebbe a carpirne l'informazione. I file invece prima di essere spezzettati vengono criptati tramite una chiave simmetrica creata dal nodo stesso. La crittografia simmetrica permette di criptare un file con una chiave, e per decriptare quel file quella chiave è l'unica utilizzabile. In modo che anche se un male intenzionato riuscisse a ricomporre il file infrangendo la crittografia asimmetrica, dovrebbe avere un ulteriore chiave per decifrare il file intero. Dato che ogni nodo mantiene l'indice dei file in locale, prima di salvare le chiavi di cifratura simmetrica sul database, queste chiavi vengono ulteriormente cifrate tramite la chiave pubblica del nodo locale. In questo modo se qualcuno riuscisse a il database, senza la chiave privata non riuscirebbe a conoscere le chiavi dei file.

Capitolo 4

Realizzazione

In questo capitolo si procede ad una disamina delle scelte tecniche fatte nel progetto darkcloud, e si illustra il funzionamento dettagliato delle procedure.

4.1 Linguaggi di programmazione e ambiente di sviluppo

Il linguaggio con il quale è stato scritto darkcloud è Java[16]. La scelta è stata orientata verso questo linguaggio dopo considerazioni fatte sulla complessità del codice di cui si necessitava. Infatti dato che si è optato per un software dove le chiavi di decriptazione, le informazioni su tutti i file, i dati degli altri nodi e molti altri dati sensibili sono salvati in locale si rendeva necessario adottare nel software forti mezzi per mantenere in sicurezza queste informazioni. In java gli strumenti per criptare sono di facile utilizzo. Inoltre anche le connessioni e la formattazione dei messaggi scambiati tra nodi risultava molto semplificata grazie alle classi già disponibili. Dato che si doveva costruire da zero una piattaforma per gestire una rete e questa avrebbe avuto bisogno di un codice molto articolato e con elementi con metodi simili ma caratterizzazioni diverse, ha reso java una scelta preferibile. Il software doveva poi essere in grado di essere eseguito su piattaforme di diverso tipo, in quanto non solo i professionisti che lo usano potrebbero eseguirlo su sistemi operativi diversi, ma visto che ne saranno usate istanze su cloud computing e facile che lo si eseguirà in queste ultime su sistemi linux. Oltre al linguaggio java nel software si usa l'sql per salvare i dati. In particolare è stato utilizzato il client sqlite3, dato che il software deve salvare informazioni su molti altri nodi, su ogni file , e per ogni file sui frammenti è indispensabile un linguaggio di gestione database multi piattaforma e stabile come l'sql. Una scelta ormai assodata del gruppo di lavoro del professor Colajanni è l'utilizzo del sistema operativo Debian per la fase di sviluppo del

software. Tale sistema operativo infatti garantisce nelle release stable la sicurezza di lavorare su un sistema molto stabile e nella comunità degli sviluppatori molto sfruttato. Essendo usato molto dai programmatori gli strumenti di compilazione, debug, e gli ambienti di sviluppo per programmare sono stati testati a lungo e quindi arrivati ad un buon livello di stabilità. Essendo sicuri che non darà spiacevoli sorprese in corso di programmazione. L'ambiente di sviluppo java utilizzato è stato Eclipse. Gli strumenti che offre per il lavoro di programmazione sincronizzato lo rendono perfetto per lo sviluppo in team. Per sincronizzare il codice si è utilizzato usato Google Code. Si tratta di un servizio di casa Google gratuito che permette di creare progetti software e consente a diverse persone di lavorarci. Si è infatti installato il plugin Subversion per Eclipse, questo ha permesso di impostare il repository per il progetto. In questo modo chi lavora ad una classe può bloccarla direttamente da Eclipse e quando ha finito sbloccarla e caricare i risultati. Quindi i requisiti di sistema sono una installazione di Java Runtime Environment versione 1.6.x o maggiore e il client di sqlite3.

4.2 Analisi di un nodo

Per funzionare un nodo, che sia client o server, ha bisogno di alcuni file. Il file eseguibile darkcloud.jar, che è lo stesso sia per i client che per i server. Questo eseguibile deve essere avviato sia sui client che sui server. Una volta avviati almeno un client e un server sarà possibile iniziare ad usare il sistema tramite un terminale, potendo salvare file e recuperarli. Il database darkcloud.db, che contiene tutte le informazioni del nodo. Nel database vengono salvati i dati di tutti i file, i dati dei frammenti dei file e le caratteristiche degli altri nodi presenti in darkcloud. Il file di configurazione darklcoud.conf, che contiene tutti i setting del nodo. Vi si trovano specificati l'indirizzo ip e la porta di ascolto del nodo locale, il tipo di nodo locale, l'indirizzo ip e le porte degli altri nodi presenti sulla rete con specificato se si tratta di client o server. Il file darkcloud.keystore, che contiene le chiavi pubbliche degli altri nodi e i certificati ssl. Dato che nella rete si usano connessioni protette SSL ogni nodo deve conoscere la chiave pubblica di ogni altro nodo, oltre a ciò le chiavi pubbliche degli altri nodi sono usate per cifrare i file che gli si mandano. Infine i file private.key e public.key che contengono le chiavi RSA pubblica e privata del nodo usate per criptare e decriptare le informazioni da e per il nodo. Se si vuole spostare un nodo su un altro computer sarà sufficiente spostare i file sopra menzionati. Dato che la mappatura dei nodi è contenuta nel file darkcloud.conf dovremmo anche modificare i valori se cambierà l'ip del computer in uso, sia nelle impostazioni locali che nelle liste ip degli altri nodi che devono comunicare con il nodo locale.

4.3 Gestione dei nodi

Per gestire i nodi sono stati creati una serie di script che permettono di eseguire le funzioni fondamentali per il funzionamento della rete. Sono script da usare solo per modifiche da parte di amministratori della rete, gli utilizzatori finali non si troveranno mai a doverli usare dovranno solo interagire con i terminali. Questi script si possono trovare nella sotto directory 'net' e sono:

- createNode.sh
- removeNode.sh

- startNet.sh
- stopNet.sh
- startNode.sh
- stopNode.sh

Lo script createNode.sh permette di creare dei nodi, i parametri richiesti quando viene avviato sono: nome del nodo, tipo di nodo che può essere client o server, porta di ascolto, keystore password, nome e cognome, nome dell'unità aziendale, nome dell'azienda, località, provincia, codice a due lettere del paese in cui si trova l'unità. Al temine della configurazione lo script controlla se sono già stati creati altri nodi. Successivamente viene chiesto se si vuole che i certificati degli altri nodi che sono stati trovati siano marcare come fidati, questo è necessario per essere in grado di comunicare con essi. Al termine di questa operazione della sotto directory 'net' avremo una cartella che si chiama come il nodo contenente i file necessari al suo funzionamento. Le impostazioni precedentemente inserite vengono salvate nel file di configurazione del nodo chiamato darkcloud.conf . Se in seguito alla creazione si vuole apportare modifiche al nodo basterà andare nella directory 'net', nella sotto directory che si chiama come il nodo e modificare tale file. In particolare di default tutti i nodi conosciuti e certificati della rete hanno 127.0.0.1 come indirizzo ip. Se si vuole spostare questi nodi su altre macchine collegate ad internet, basta poi modificare i file darkcloud.conf di tutti i nodi della rete. Infatti la conoscenza che un nodo ha degli altri è specificata in tale file, quindi se la geografia della rete viene cambiata deve essere aggiornata in ogni nodo della rete. Lo script removeNode.sh permette di disinstallare il nodo dalla macchina locale, e si occupa anche di rimuovere da eventuali altri nodi presenti nella cartella le specifiche del nodo rimosso. Con lo script startNet.sh si può avviare tutti i nodi presenti nella directory 'net'. Se invece si vogliono avviare dei nodi singolarmente si utilizzerà lo script startNode.sh. Quando un nodo viene avviato in una cartella nascosta che si chiama .run vengono creati dei file nel formato 'nome nodo'.pid nei quali vengono salvati i pid dei nodi avviati. Questo serve per evitare di aprire più istanze dello stesso nodo, azione che porterebbe a problemi sul database e alla gestione delle risorse. Al loro avvio gli script di avvio della rete o dei nodi controllano che nella cartella .run non vi siano già i pid dei nodi che si tenta di avviare. Inoltre gli script di avvio controllano che all'interno della cartella del nodo siano presenti l'eseguibile e il file di configurazione per evitare errori. Analogo discorso per lo script stopNet.sh tramite il quale si può spegnere tutti i nodi presenti nella directory 'net'. Mentre se si vuole spegnere un singolo nodo si utilizzerà lo script stopNode.sh. Oltre a inviare il segnale di kill del processo gli script di arresto cancellano dalla cartella .run i pid dei processi.

4.4 Gli oggetti Darkcloud e NetNode

Quando si avvia l'eseguibile di darkcloud, il main crea una istanza della classe Darkcloud e una della classe NetService. Darkcloud è la classe principale di ogni nodo, che contiene le variabili con le chiavi per la criptazione, le liste con i dati degli altri nodi, il collegamento con il database e altro ancora. NetService invece è la classe che prende i dati di ip e porta di ascolto del nodo locale e crea il socket ssl che viene usato in ascolto. A seconda che un nodo sia client o server poi viene creata una istanza di ServerService o ClientService, che rimangono in ascolto di connessioni in entrata e in caso arrivino richieste istanziano a loro volta le classi Client o Server che si occupano

di leggere il messaggio in arrivo e decifrare di che tipo di richiesta si tratta, invocando poi il giusto metodo di risposta delle classi ServerResponseMethod o ClientResponseMethod. Queste ultime due classi contengono il codice che un nodo esegue in risposta ad una richiesta. I dettagli del codice eseguito nei diversi comandi verrà analizzato nel paragrafo dedicato ai comandi. Le diverse richieste sono elencate all'interno della classe Request.java, che modella le richieste che i nodi possono scambiarsi. Attraverso il tipo di Java definito Enum, Enumerazione, si può nella classi Request.java definire i diversi tipi di richieste semplicemente specificandone il nome, in modo che nelle classi ServerResponseMethods.java e ClientResponseMethods.java si vada a indicare il codice da eseguire a seconda della richiesta ricevuta. Se si vuole aggiungere un comando basta aggiungerlo alla lista lista in Request.java e scriverne il codice nella classe di risposta del client o del server.

La classe Darkcloud contiene due paricolari hashmap, chiamati clientNodes e serverNodes. Una hashmap è una collezione di oggetti, che memorizza coppie chiave-valore, dove la chiave serve come indice. clientNodes e serverNodes infatti sono composti da oggetti di tipo NetNode e chiavi di tipo stringa. Gli oggetti NetNode sono istanze della classe omonima e vengono create come immagini degli altri nodi della rete, utilizzate quindi dalla classe Darkcloud come interfacce per comunicare con gli altri nodi. La classe clientService oltre ad aspettare le connessioni in arrivo crea all'avvio un istanza della classe NetPoll. Questa classe viene fatta partire all'inizio del programma e rimane attiva sempre fino a quando non viene terminato il programma, il suo compito è quello di controllare periodicamente quali nodi sono vivi e quali no. Fa questo inviandogli dei messaggi di tipo ping, e in base alla risposta segna gli oggetti NetNode come online o offline all'interno della struttura clientNodes e serverNodes. La classe NetNode possiede un metodo che si chiama send. Questo metodo viene invocato dal programma quando ha bisogno di inviare informazioni a un altro nodo. Il metodo send accetta in ingresso un oggetto di tipo Request e fornisce in uscita un oggetto di tipo Response. Dato che il nodo locale costituito dalla classe Darkcloud, ha per ogni altro nodo un oggetto NetNode, all'interno del quale sono contenute le informazioni dell'altro nodo, se si vuole inviare un messaggio all'altro nodo sarà sufficiente usare il comando NetNode.send. Ora il NetNode si attiverà e il suo metodo send leggerà i suoi dati, creerà una connessione con il nodo remoto che lui rappresenta, gli invierà l'oggetto request che Darkcloud cioè il nodo locale gli vuole mandare, aspetterà la risposta rappresentata da un oggetto response e la restituirà a chi lo ha invocato cioè Darkcloud.

4.5 Connessioni

In questo paragrafo parleremo delle connessioni che si instaurano tra due nodi, tramite i quali avviene lo scambio di informazioni. Nel caso specifico di Darkcloud si è deciso di utilizzare le connessioni SSL. Le connessioni SSL sono connessioni criptate tramite una chiave pubblica. Quando si creano i nodi, lo script di creazione crea per ogni nodo una coppia di chiavi e un KeyStore. Il KeyStore è un file che contiene tutte le password pubbliche degli altri nodi che appartengono alla rete. Questo KeyStore per sicurezza è a sua volta protetto da una password. Quando il programma viene avviato la prima classe, quella principale, che viene istanziata è Darkcloud. All'interno di questa classe nei comandi di inizializzazione vengono impostate due variabili di sistema di java, javax.net.ssl.keyStore e javax.net.ssl.trustStore. In queste due variabili si vanno a porre la posizione del KeyStore, in modo che quando nel corso del programma si procede a creare un socket SSL java sappia dove può reperire la chiave del nodo che vuole contattare e in questo modo creare la connessione protetta.

L'avere una connessione però non è sufficiente a trasferire gli oggetti ideati per passare i dati tra una classe e l'altra. Infatti il protocollo SSL trasporta stringhe mentre in Darkcloud è necessario passare tra nodi diversi molti tipi di informazioni diverse, come numeri interi, byte di dati, date e altro. Per questosi ricorre ad una particolare formattazione delle stringhe che poi verranno scritte nei socket. In modo che quando il dato inviato viene ricevuto da un altro nodo lui sappia come interpretarlo e quindi possa estrarne i contenuti informativi. Per formattare i messaggi si ricorre all'XML. XML è un linguaggio di markup, ovvero un linguaggio che definisce un meccanismo sintattico che permette di definire e riconoscere il significato degli elementi contenuti in un testo.

Quando due classi vogliono scambiarsi informazioni lo fanno essenzialmente tramite l'uso di due oggetto, la request e la response. Request e response sono estensioni della classe message. La differenza tra di loro è che una request deve richiamare un comando, mentre la response deve comunicare se l'operazione è andata bene o male. Dato che i comandi dei nodi sono cinque anche le tipologie di request sono cinque, mentre i tipi di response sono due, risposta positiva o errore. Il codice degli oggetti request e response ha dei metodi che permettono di inserire al loro interno tutte le informazioni necessarie, e anche di organizzarle. Tramite il metodo appendField si può creare un nuovo campo, in questo campo è possibile inserire direttamente dei dati, oppure creare al suo interno degli attributi, tramite il metodo setContent. Quando si è finito di inserire i campi e gli attributi le informazioni all'interno degli oggetti sono pronte per essere spedite. Per fare questo utilizzeremo il metodo delle request e delle response che si chiama toString. Questo metodo prende tutti i dati riposti in un oggetto, organizzato in campi e attributi, e lo converte in messaggi XML dovutamente formattati, sotto questa forma i dati possono viaggiare sul socket SSL. Una volta ricevuto questo messaggio XML potrà facilmente essere trasformato nell'oggetto originario, request o response che sia. Questo avviene creando un nuovo oggetto request o response e fornendogli il messaggio tramite il metodo fromString. Dato che l'oggetto sa come sono formattai i dati in XML può agilmente convertirli nei campi e negli attributi originali. In questi messaggi XML prima di essere spediti viene inserita la lunghezza del messaggio e la firma del mittente. Questo per facilitare la decodifica al metodo ricevente. Più precisamente è un messaggi formattato in 3 righe, nella prima riga cè la lunghezza del messaggio, nella seconda riga il contenuto informativo e nella terza riga la firma del mittente.

I metodi che gestiscono la scrittura di questo messaggi sui socket sono il send di NetNode e run di CLient. Quando un nodo vuole mandare un messaggio ad un altro nodo utilizza il metodo send all'interno dell'istanza NetNode che rappresento il nodo ricevente. Mentre il meccanismo di ricezione è basato sulla classe ClientService o ServerService. Queste classi sono continuamente in ascolto e se rilevano un messaggio SSL istanziano Client e gli passano il socket. Sia nel caso del send che del Client per prima cosa si creano due buffer uno per la lettura e uno per la scrittura. Nel send poi si allega alla request il campo nodeid, che comprende il proprio ip e la propria porta di ascolto, in modo che il ricevente abbia sempre i dati per identificare il mittente. Si procede poi a scrivere sul buffer uscente il messaggio XML tramite il comando request.toString() di cui si è parlato prima che traduce il contenuto della request in codice XML. A questo punto il clientService o il serverService rilevano il messaggio, istanziano Client o Server e gli passano il riferimento al socket. Il Client per prima cosa legge la prima riga passata nel messaggio, che contiene la lunghezza, questo valore gli serve per decifrare il resto del messaggio. Ora legge dal buffer una riga lunga quanto era specificato nella prima riga. Fornisce in ingresso al metodo fromString, di un nuovo oggetto request, la stringa appena letta, i cui valori vengono cosi inseriti

nella request stessa. Il Client ora salva in un array tutti i metodi che conosce il client, e poi li confronta con quello della request. Se trova una corrispondenza invoca tale metodo e gli passa l'oggetto request. Come risultato del codice del comando gli viene restituito un oggetto response, che procede a spedire sullo stesso socket dal quale ha letto tramite però un buffer di scrittura. Chiaramente per scriverlo usa il metodo toString dell'oggetto response ricevuto che traduce il contenuto in una stringa. Infine Client chiude i due buffer e il socket. Il send riceve il messaggio tramite il socket, chiude il socket, traduce il response con fromString, chiude i buffer e il socket. Procede al controllo dell'esistenza del campo nodeid per verificare l'identità di chi gli ha risposto. Dopo di che controlla anche il campo che contiene la chiave pubblica, e controlla se corrisponde con quella che il nodo ha salvato in corrispondenza di quel nodeid. Se così non è sostituisce la chiave con quella ricevuta. Per concludere restituisce l'oggetto response alla classe che lo aveva invocato.

4.6 Database

In Darkcloud si fa largo uso di database[17]. I dati mantenuti sul nodo sono molti. Infatti le informazioni sui dati e sugli altri nodi non sono centralizzate o mantenute in remoto, ma sono invece tutte salvate in locale. Ogni nodo, sia client che server, possiede nella cartella principale del programma un database che si chiama darkcloud.db . Le tabelle principali che contiene sono :

- FILE
- FILEFRAGMENT
- FILEPERMISSION
- FILEUPDATE
- NODE
- NODETYPE
- PERMISSIONTYPE
- UPDATETYPE

Lo schema relaziona del database è riportato di seguito:

FILE(NAME,CONTENT,KEY,CHECKSUM,UPLOADER,CREATIONTIME,
MODIFYTIME,MODIFYBY)
NODETYPE(NODETYPEID,NODETYPESTR)
UPDATETYPE(UPDATETYPEID,UPDATETYPESTR)
FILEFRAGMENT(NAME,FRAGMENTID,CHECKSUM,NODEID)
FK: NODEID **REFERENCES** NODE(NODEID)
FK: UPDTYPE **REFERENCES** UPDATETYPE(UPDATETYPEID)
FILEUPDATE(UPDID,FILENAME,NODEID,UPDTYPE,UPDTIME)
FK: FILENAME **REFERENCES** FILE(NAME)
FK: UPDTYPE **REFERENCES** UPDATETYPE(UPDATETYPEID)
NODE(NODEID,PUBKEY,TYPE,ADDR,PORT)

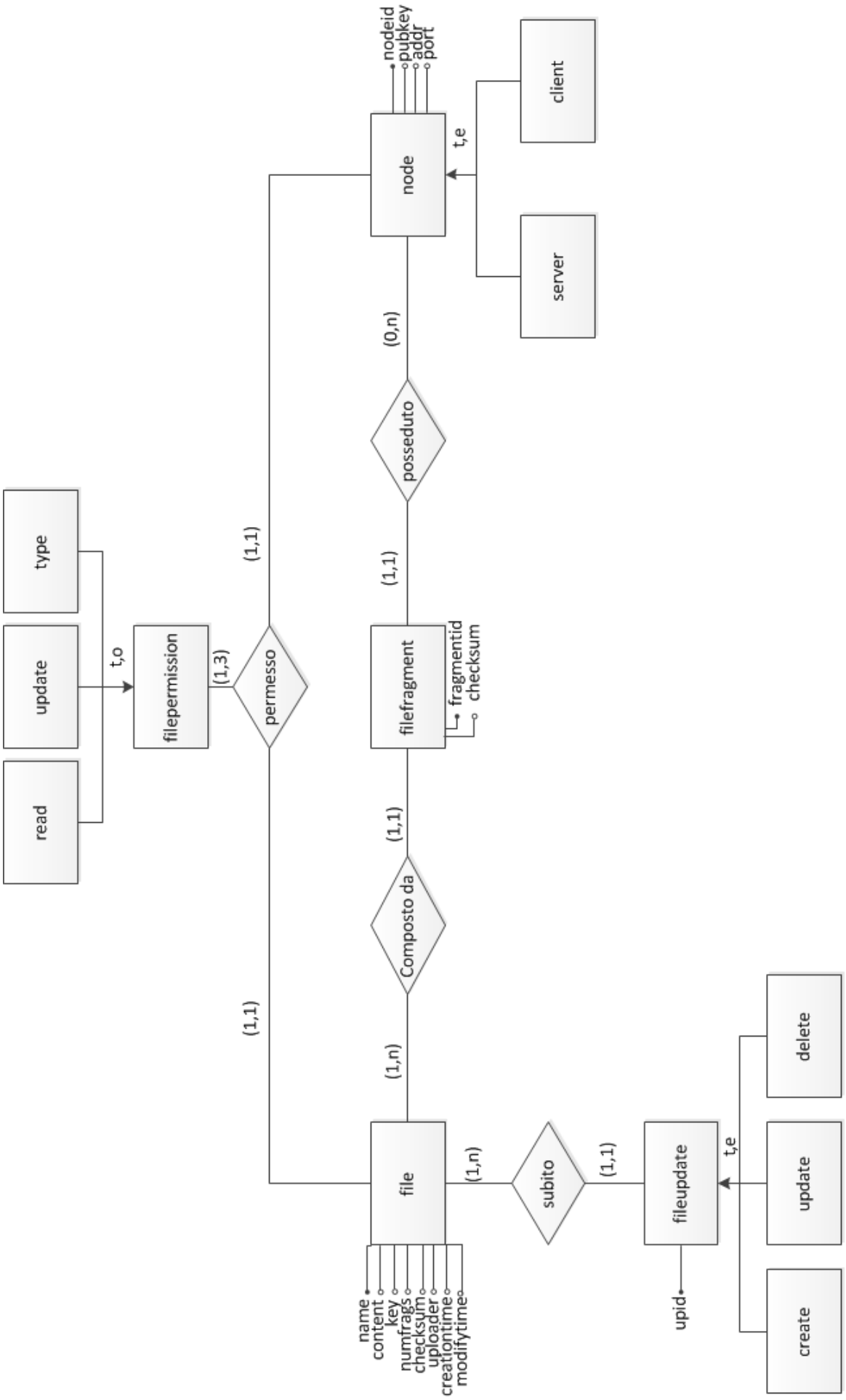

Figura 4.1: Class Diagram

FK: TYPE **REFERENCES** NODETYPE(NODETYPEID)
PERMISSIONTYPE(PERMTYPEID,PERMTYPESTR)
FILEPERMISSION(NODEID,FILENAME,PERM)
FK: NODEID **REFERENCES** NODE(NODEID)
FK: FILENAME **REFERENCES** FILE(NAME)
FK: PERM **REFERENCES** PERMISSION(PERMID)

All'interno del codice gli inserimenti e le letture di dati sono frequenti, invece che elencarle qui si preferisce inserirle nelle descrizioni dei comandi riportati di seguito.

4.7 Crittografia

Questo programma fa della crittografia dei dati uno dei suoi punti di forza. Dato che informazioni preziose, che potrebbero essere utilizzare per risalire ai dati, sono salvate in locale si fa un largo uso di tecniche di cifratura. In questo progetto si usano due tipi di cifratura la AES e la RSA. Sono due tipi di cifratura diversi, il primo è simmetrico, il secondo asimmetrico. L'AES[8] (Advanced Encryption Standard) conosciuto anche come Rijndael, è l'algoritmo di cifratura a blocchi utilizzata come standard dal governo degli Stati Uniti d'America[11]. E' una tecnica di cifratura che perfino l'NSA utilizza per cifrare i documenti TOP SECRET. Questa tecnica di cifratura può utilizzare chiavi con lunghezza variabile di 128,192 o 256 bit e lavora con blocchi di dimensione prefissata di 128 bit. La dimensione maggiore della chiave crittografica garantisce una maggior sicurezza. Sono stati fatti dei test nel 2011 per vedere in quanto tempo con due gpu e un metodo di forza bruta si riuscisse a decifrare un file criptato in AES a 256 bit. Il risultato è stato che ci sarebbero voluti circa 13 anni di calcoli. All'interno del programma per una maggior sicurezza infatti si utilizzano chiavi da 256bit.

Della cifratura asimmetrica si è già accennato nel capitolo due, il meccanismo è semplice, ogni nodo ha una chiave pubblica e una privata, quella pubblica è scambiata con tutti i nodi e quella privata è mantenuta segreta. Se si cripta un file usando quella pubblica solo quella privata può decifrarlo , in questo modo tutti possono creare messaggi decifrabili solo da chi detiene la chiave privata. Lo standard de facto oggi si basa sulla cifratura RSA[9], un cifrario a chiave pubblica che permette di cifrare un messaggio sfruttando alcune proprietà elementari dei numeri primi. Questo cifrario prende il nome dalle iniziali dei matematici che nel 1976 lo crearono: Rivest, Shamir e Adleman. La loro idea fu quella di sfruttare la difficoltà di fattorizzare un numero intero. Di fatti la chiave pubblica è un numero N ottenuto moltiplicando due numeri primi molto grandi che restano segreti. Come nell'AES anche l'RSA può usare chiavi di grandezza variabile, e anche in questo caso più grandi sono, maggiore è la difficoltà in tentativi di attacchi di forza bruta. Le chiavi possono andare da una dimensione di 1024 ad un massimo di 4096 tipicamente. Spendendo un milione di euro per dotarsi di un cluster potente ci vorrebbero circa sei mesi per decriptare un messaggio criptato con RSA a 768 bit. Alcuni studiosi affermano che le chiavi a 1024 bit saranno scardinabili nel prossimo futuro[10]. Infatti oggi la maggior parte dei sistemi usa chiavi a 1024 o 2048. Si è deciso per tutela di usare chiavi a 4096 bit, che gli studiosi di crittografia credono sarà difficile poter scardinare in base alle previsioni sul futuro dell'informatica. La classe CryptUtil presente nel pacchetto Util di Darkcloud è quella nella quale si trovano i metodi che gestiscono la cifratura. Infatti vi sono il metodo generateKeyPair che permette di generare una coppia di chiavi RSA a 4096 bit, il metodo generateSymmetricKey che genera chiavi AES simmetriche da 256 bit, i metodi storePublicKey e storePrivateKey che permettono di memorizzare nei file usati come

keystore le chiavi, i metodi getPrivateKey e getPublicKey che specifiando un file restituiscono la chiave contenuta. Sono presenti alcuni metodi che permettono di convertire le chiavi in oggetti diversi, per esempio le chiavi generalmente in java sono utilizzate come oggetti Key o Secretkey però quando è necessario inserire chiavi in un messaggio non si possono lasciarle in questa forma, bisogna convertirle in stringhe. Questi metodi di conversione sono privKeyToString e pubKeyToString che convertono oggetti Key in stringhe, privKeyFromString e pubKeyFromString che traducono le stringhe in oggetti Key, secretKeyFromString che traduce le stringhe in oggetti SecretKey. I metodi sign e verifySign rispettivamente servono per generare e verificare le firme da allegare ai messaggi scambiati tra nodi, firme scritte con le chiavi private e che quindi possono essere decifrate tramite le chiavi pubbliche per verificare l'identità del mittente. Gli ultimi due metodi sono encrypt e decrypt che permettono di criptare o decriptare un array di byte specificando la chiave e il metodo di cifratura. I file situati nella cartella principale del nodo che riguardano la cifratura sono fondamentalmente quattro. Il certificato del nodo ad esempio

```
client_1.crt
```

4.8 Comandi

Andremo ora ad analizzare nello specifico quali passaggi avvengono e come vengono elaborati i dati nelle esecuzione dei comandi. I comandi principali sono PING, PUT, GET e SHARE.

4.8.1 Ping

Il metodo ping viene utilizzato principalmente nei terminali per verificare se un nodo client è vivo e nella classe NetPoll che lo usa per tenere aggiornato l'elenco dei server e dei client vivi sulla rete. La sintassi del comando ping è:

python client.py -r PING -h 'indirizzo ip del client' -p 'porta di ascolto del client'

Il nodo che riceve un messaggio ha sempre in ascolto sul socket SSL la classe ClientService. Questa classe controlla continuamente se sul socket arrivano richieste. Quando ne rileva una crea una istanza della classe Client e gli passa il riferimento al socket. L'istanza di Client legge dal socket il messaggio, legge di che tipo di request si tratta e in questo caso invoca il metodo ping della classe ResponseMethods. ResponseMethods è una generalizzazione che poi viene estesa per creare ClientResponseMethods e ServerResponseMethods. L'istanza di Client passa al metodo ping la request contenuta nel messaggio. Per prima cosa il metodo ping recupera dal messaggio il campo pubkey e il campo nodeid, che contengono rispettivamente la chiave pubblica e il codice identificativo del nodo che gli ha mandato la richiesta . In seguito confrontando il nodeid con quelli salvati nelle strutture clientNodes e serverNodes controlla se è un nodo con il quale può dialogare e se si tratta di un server o di un client. In caso si tratti di un nodo conosciuto confronta la chiave pubblica del messaggio con quella che ha salvato nell'oggetto NetNode corrispondente, in caso sia uguale crea un nuovo oggetto response, gli da l'attributo di tipo ACK cioè di risposta positiva, e lo restituisce all'istanza di Client. L'istanza di Client ricevuto l'oggetto response lo scrive nel socket SSL, lo invia, chiude la connessione e muore. In questa spiegazione si è parlato solo della richiesta ping nel caso sia ricevuta da un nodo di tipo client, ma a differenza degli altri comandi il ping funziona nello stesso modo sia per i server che per i client. Con l'unica differenza che il client risponde anche se non viene specificata la pubkey e il nodeid, in

act.png

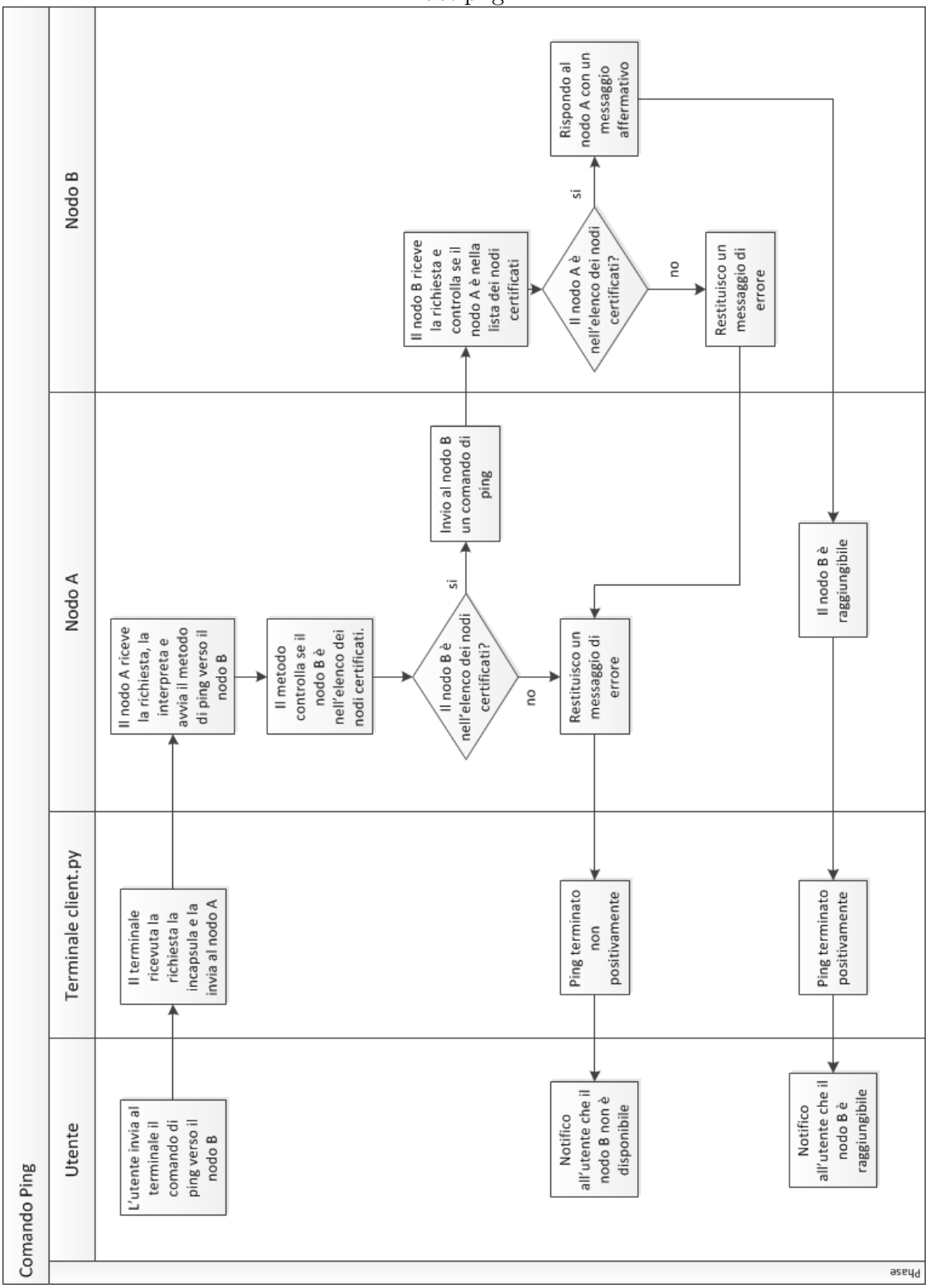

Figura 4.2: Activity diagram del comando ping

act.png

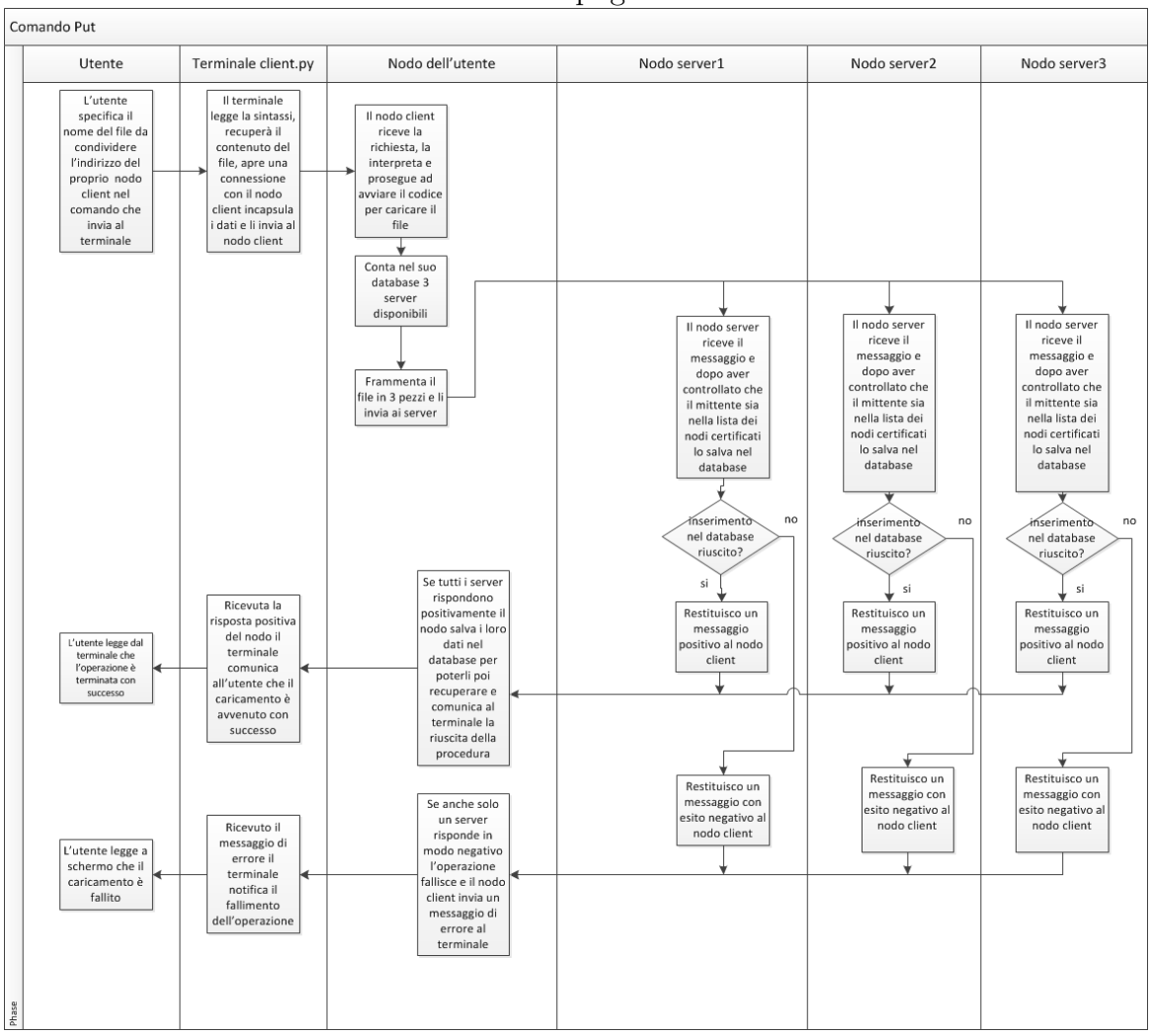

Figura 4.3: Activity diagram del comando put

quanto il client può ricevere richieste anche da terminali che di fatto non possiedono tali caratteristiche.

4.8.2 Put

La ricezione del comando put da parte di un client invece innesca una sequenza di azioni che comprendono anche l'interazione con i server. Quindi analizzeremo sia la risposta al comando put di un client che di un server. La sintassi del comando put è:

python client.py -r PUT -h 'indirizzo ip del client' -p 'porta di ascolto del client' -f 'nome del file locale' -F 'nome da assegnare al file in remoto'

Un utente dalla cartella dove si trova lo script che nei test fungeva da terminale, client.py, digita il comando. In questo specifica l'ip e la porta di ascolto del client a lui dedicato, il nome di un file che si trova nella stessa cartella, o specifica prima del nome la posizione, infine il nome con il quale sarà salvato il file dal client. Premuto invio lo script cioè il terminale effettua i controlli necessari sui campi inseriti, controlla se il file è codificato, in caso contrario ne legge il contenuto, lo salva e calcola il checksum del file. Crea un messaggio xml, vi aggiunge i campi per il contenuto del file, il nome del file in remoto, il checksum e il tipo di codifica con cui potrebbe essere salvato il file. Dopo di ché crea una connessione in base all'ip e alla porta di ascolto specificati e

apre un socket SSL tramite il quale invia il messaggio al client locale. Il client che ha ClientService in ascolto rileva il messaggio, istanzia Client e gli passa il socket. Client legge il messaggio legge il tipo di richiesta e invoca il metodo put della classe ClientResponseMethods passandogli l'oggetto request contenuto nel messaggio. Il metodo put di ClientResponseMethod legge dall'oggetto request il contenuto del campo file, controllando che non sia vuoto. Recupera anche il nome del file e il tipo di codifica, in caso il file risulti codificato il metodo procede a decifrarlo, dopo di che legge il campo checksum e calcola il checksum del file ricevuto. Paragona il checksum calcolato con quello del messaggio per verificare errori di trasmissione, in caso combacino procede. Ora tramite il metodo getAliveServerNodes della classe Darkcloud recupera una copia dei NetNode che sono vivi e li pone in un hashmap. Ora si rende necessario mescolare l'hashmap in modo che i server a cui si inviano i frammenti non siano tutti consecutivi. Per fare questo si crea una seconda hashmap con la stessa struttura di quella con i server. Si crea un array di stringhe e vi si salvano tutte le chiavi della hashmap dei server. La funzione Collections.shuffle(keys) procede a mescolarne le stringhe e con un ciclo for inserisce nella seconda hashmap i NetNode usando come di selezione le stringhe mescolate. A questo punto avremo la seconda hashmap che contiene i NetNode mescolati a dovere. Si procede con la fase di cifratura del file. Si genera una chiave simmetrica tramite il metodo generateSymmetricKey della classe CryptUtil. Il contenuto del file viene criptato sempre tramite un metodo di CryptUtil che si chiama encrypt. Particolare attenzione va fatta in questi passaggi in quanto nelle comunicazioni tra metodi i byte vanno passati codificati in base 64, inoltre i metodi che criptano vogliono in ingresso byte e non stringhe, maggiori dettagli vengono forniti nei prossimi paragrafi riguardanti la crittografia. Giunti a questo punto il contenuto del file è criptato e contenuto in un array di byte. Per sapere quanto grandi devono essere i frammenti si recupera la dimensione del file criptato e la si divide per il numero di server. In questo modo però se la divisione ha del resto andrebbero persi dei byte, quindi si aggiunge 1 alla dimensione dei frammenti. Tramite un ciclo for si legge le porzioni del file criptato che vengono poi salvate in un array di stringhe, la lettura parziale avviene tramite la funzione substring che permette di leggere da una stringa un numero dato di caratteri a partire da una posizione a scelta. Ora le parti sono contenute nell'array di stringhe. In un ciclo for che si ripete per il numero di server si inviano i frammenti. Creato un oggetto NetNode gliene si uno preso dalla hashmap mescolata, in modo da poter dopo usare il metodo send e mandargli un messaggio. Ora calcolato il checksum del frammento, si prende la request che ha mandato il terminale e gli si aggiunge i campi con il contenuto criptato, il checksum, dato che vi sono ancora i campi creati dal terminale la request contiene anche il nome del file e il nodeid del client. Completato l'oggetto request questo viene mandato tramite il metodo send al NetNode preso dalla hashmap mescolata.

Ora il server riceve tramite la consueta procedura l'oggetto request di tipo put quindi viene invocato il metodo put del ServerResponseMethod. Il metodo put del server legge il contenuto del file e fa il controllo del checksum. Se il checksum corrisponde procede a salvare nel database i dettagli del frammento. I campi che vengono salvati sono nome del file, contenuto del frammento, checksum, nodeid del client, data di creazione e data di modifica. Inoltre dato che viene tenuta traccia anche delle modifiche fatte ai file viene anche inserito un nuovo record nella tabella FILEUPDATE con i dati nome del file, nodeid, tipo di update, e data della modifica. Al termine del metodo se tutto va bene si restituisce un oggetto response di tipo ACK che viene mandato al client.

Quando il client riceve il response di tipo ACK passa al salvataggio nel database dei dettagli di questo frammento nella tabbela FILEFRAGMENT, che comprendono nome

del file, un intero che indica il numero del frammento, il checksum e il nodeid del server dove il frammento è salvato. Alla fine di questo ciclo for se tutti i frammento sono stati correttamente inviati il metodo prende la chiave simmetrica usata per criptare il file e la cripta usando la sua chiave pubblica. Poi registra sul database nella tabella FILE i dettagli del file appena splittato sui server, salvandone nome, chiave criptata, checksum, data di creazione e data di modifica.

4.8.3 Get

Il comando get usato tramite un terminale permette di recuperare un file precedentemente salvato. Come per il comando put questo comando fa eseguire codice diverso a seconda se sia inviato ad un client o ad un server. Nel seguito analizzeremo entrambi. La sintassi del comando get è:

python client.py -r GET -h 'indirizzo ip del client' -p 'porta di ascolto del client' -F 'nome del file in remoto'

Quando un utente vuole recuperare un file che ha salvato in darkcloud sarà sufficiente che usi il comando get, il codice che sta dietro recupererà i frammenti, decripterà il contenuto e darà in uscita il contenuto del file. Il terminale come nel caso del ping e del put creerà un messaggio xml e al suo interno metterà il nome del file da recuperare, i dati del nodo client che intende usare per recuperarlo. Dopo di che creerà un socket SSL e invierà al nodo client che voi avete specificato, e che è l'unico che un utente deve usare per recuperare i suoi file, il messaggio. Il ClientService sempre in ascolto rileverà il messaggio, creerà in istanza di Client e gli passerà il socket. Client leggerà il messaggio, recupererà il tipo di request contenuta al suo interno e invocherà il metodo corrispondente del ClientResponseMethods, nel caso attuale il metodo get. Il metodo prende in ingresso un oggetto request, e inizia leggendo il campo file e verificando che non sia vuoto. Poi passa a leggere all'interno del campo file l'attributo name,cioè il nome del file, e controlla che anche esso non sia nullo. Fatti tali controlli il metodo fa un join sulle tabelle FILE e FILEFRAGMENT sulla colonna name, dove il valore name è uguale al nome passato nella request. Leggendo poi in tale ordine la chiave di decriptazione, il fragmentid cioè il numero del frammento, il checksum del frammento e il nodeid dove il frammento è salvato. Ordinando i risultati per fragmentid. Se la query restituisce risultati il passo successivo è decriptare la chiave. Tramite il metodo decrypt della classe Cryptutil specificando la chiave da decriptare, la propria chiave privata recuperata tramite il metodo getPrivateKey di Darkcloud e infine il tipo di cifratura nel caso RSA, otterremo la chiave simmetrica del file in chiaro sotto forma di stringa. Visto che però per decifrare il file tramite il metodo decrypt dopo averlo ricomposto dovremo specificare la chiave sotto forma di oggetto Secretkey, si converte subito da stringa a Secretkey tramite il metodo secretKeyFromString della classe CryptUtil. A questo punto tramite un ciclo for si ripetono le prossime operazioni per il numero di record restituiti dal join. Preso il primo nodeid restituito dal join e si copia l'oggetto NetNode che gli corrisponde in una istanza locale tramite il metodo getAliveServerNodes().get(nodeid). Ora il NetNode pronto per inviare una request al server. Creato un oggetto request, si imposta l'attributo tipo come get e gli si aggiunge un nuovo campo nominandolo file, al quale poi si aggiunge l'attributo name. Attribuito a name il nome del file ed si invia al server questa request tramite il metodo send dell'oggetto NetNode prima recuperato.

Ora il server riceverà la richiesta, leggerà dall'attributo name del campo field dell'oggetto request il nome del file. Tramite una select sulla sua tabella FILE, in base al nome recupererà i seguenti campi: contenuto, checksum, data di creazione, data di

modifica, il nodeid del nodo uploader e la modificabilità. Avendo tutti i dati necessari ora il metodo get creerà un nuovo oggetto response, impostanto l'attributo tipo su ACK. A questo response aggiunge un campo file e a questo campo aggiunge gli attributi checksum, data creazione, data modifica, uploader, modificabilità e contenuto del file. Assegna a questi attributi i valori letti nel database e restituisce questo oggetto response al client.

Il client se la response è di tipo ACK procede alla lettura del campo file. Come prima cosa legge l'attributo checksum e lo paragona con quello che calcola dal frammento contenuto nell'oggetto response. Se combaciano aggiunge il frammento del file alla stringa fileContent. La stringa fileContent alla fine del ciclo for contiene tutto il file cifrato ricomposto. Quindi è pronta per essere decriptata tramite la chiave che all'inizio del metodo si era recuperato dal database. Decifrato il file sarà sufficiente inserirlo nell'oggetto response e restituirlo al terminale. Ora il terminale ha il file richiesto dall'utente. Lo script usato per i test stampa direttamente a schermo il contenuto, se si vuole ottenere il file basterà indirizzare il risultato del comando in un file con quel nome.

4.8.4 Share

Il comando share permette di condividere la conoscenza di un determinato file con altri utenti. Come visto nel comando put, quando un client salva in darkcloud un file questo viene spezzettato ed inviato ai server, dopo di che il client salva i dati necessari a ricomporlo in sicurezza. Per far si che anche un altro client possa recuperare quel file è necessario prendere i dati contenuti nel database del client che condivide e copiarli nel database del client con cui si vuole condividere il file. Quindi a differenza degli altri comandi che vedevano coinvolti un client e diversi server questa volta nel codice del comando share devono interagire due client. nel caso del put e del get erano diverse le versioni del comando per il client e quella per il server, questa volta invece il codice del comando share è contenuta solo nei metodi di risposta del client, cioè nella classe ClientResponseMethod. Però visto che la procedura di salvataggio deve essere fatta direttamente dal nodo ricevente, si è creato un altro metodo, che si chiama receive. Un nodo non può salvare informazioni direttamente sul database di un altro nodo, sarebbe pessimo per la sicurezza. Quindi quando si invoca il comando share dal nodo A verso il nodo B per condividere un file, il nodo A reperisce nel suo database le informazioni del file, e le invia al metodo receive del nodo B, che procede con il controllo dei dati e il salvataggio nel suo database. La sintassi del comando è :

python client.py -r SHARE -h 'ip del client che condivide' -p 'porta del client che condivide' -F 'nome del file remoto' -H 'ip del ricevente' -P 'porta del ricevente'

Quando un utente vuole condividere con un altro la conoscenze di un file utilizza il suo terminale e il comando sopra scritto. Quando il terminale invia il messaggio al client, questo lo riceve tramite la classe ClientService e crea un istanza della classe CLient, la quale tradurrà il messaggio in un oggetto request. Riconosciuto che l'oggetto request è di tipo share procederà ad invocare tale metodo passandogli i campi contenuti nel messaggio. Il metodo share procede al controllo dell'esistenza del campo sharing. Nel campo sharing sono contenuti tre attributi, secondhost, secondport e remotefile. Questi attributi sono stati attribuiti dal terminale con i campi specificati nella sintassi del comando share. Il codice prosegue controllando che a tutti gli attributi sia stato assegnato un valore. Fatto ciò si passa a leggere dal database i dati che riguardano il file da due tabelle, FILE e FILEFRAGMENT. Per fare questo si ricorre al un join sul'attributo name che deve essere uguale al nome del file remoto specificato

nel comando. I risultati verranno ordinati per fragmentid e i campi selezionati sono in ordine la chiave del file, il checksum del file intero, la data di creazione del file, il numero che identifica il frammento, il checksum del frammento e nodeid del server che conserva il frammento. Si passa ora a recuperare il NetNode corrispondente al client a cui si deve inviare i dati. I NetNode sono identificati da un codice alfanumerico ottenibile dalla funzione getNodeKey della classe Darkcloud dando in ingresso l'indirizzo ip e la porta di ascolto del nodo. Ottenuto il nodeid si procede a copiare in locale l'oggetto NetNode del client a cui si inviano i dati. Per ottenere questo oggetto si ricorre il metodo Darkcloud.getIstance().getAliveClientNodes().get(nodeid) . Come si nota si prende il NetNode dalla struttura dati dove sono salvati quelli segnati come vivi. Infatti anche i client proprio come i nodi server vengono contattati periodicamente per verificare che siano online. Il prossimo passo consiste nel preparare la chiave del file per il trasferimento. Quando si è salvata la chiave simmetrica del file, nel metodo put, si cripta con la chiave pubblica locale. Ora bisogna decriptarla, con la propria chiave privata e criptarla con la chiave pubblica del nodo a cui si desidera mandarla. Per decriptarla si usa il metodo decrypt della classe Cryptutil, dandogli in ingresso la stringa presa nel database, la chiave pubblica locale ottenibile con il metodo Darkcloud.getIstance().getPrivateKey(), e infine specificando il metodo di decriptazione cioè RSA. Il metodo decrypt restituisce un array di byte, che si procede a convertire in un oggetto SecretKey tramite il metodo della Cryptutil denominato secretKeyFromString. Questo passaggio è necessario in quanto il metodo per criptare accetta in ingresso come chiave solo un oggetto SecretKey. La chiave pubblica del nodo ricevente è salvata nel NetNode corrispondente recuperato, e per usarla basterà il metodo NetNode.getPubKey(). Fornendo la chiave simmetrica decriptata e la chiave pubblica del nodo ricevente al metodo encrypt si ottiene la chiave simmetrica criptata pronta per essere spedita. Ricordo che le stringhe passate tra i metodi e inserite nei messaggi devono essere codificate in base64 per evitare problemi di trasferimento. Ora si prepara un oggetto request e gli si inseriscono i valori da mandare al client ricevente. L'oggetto request è di tipo receive. Agli oggetti request o response si può aggiungere tutti i campi desiderati e nei campi tutti gli attributi che si vogliono. In questo caso si deve trasferire un insieme di informazioni che riguardano il file intero e un numero variabile di insiemi di informazioni che riguardano i frammenti. Per questo prima si crea l'oggetto request, si setta il tipo come receive, si inserisce un nuovo campo che chiameremo file nel quale si inseriscono gli attributi che riguardano il file intero, che sono nome, chiave simmetrica, checksum, data di creazione e numero di frammenti. Ora, tramite un ciclo for che si ripete tante volte quanti sono i frammenti, si creano tanti campi quanti frammenti. Ogni campo si chiamerà fragment'i', dove i è il numero che identifica il frammento. Tramite il ciclo in ognuno di questi campi inseriremo come attributi i dati dei frammenti, che sono numero del frammento, checksum e l'identificativo nodeid del server dove si trova. Terminata la preparazione della request si procede a mandarla al client ricevente tramite il metodo send dell'oggetto NetNode precedentemente recuperato. Il metodo send aprirà un socket SSL con l'altro client, e invierà il messaggio.

Il client ricevente tramite ClientService riceve il messaggio, crea un istanza di Client, la quale riconosce il tipo di request come receive. Invoca il metodo receive dalla classe ClientResponseMethod e gli passa la request. Il metodo receive legge per primo l'attributo del campo file che contiene il numero dei frammenti. Poi imposta un ciclo for che legge i campi che contengono i dati dei frammenti e li salva nella tabella FILEFRAGMENT. Se tutti i frammenti vengono salvati il metodo receive procede con la lettura del campo file e salva nella tabella FILE i dati in esso contenuto. Se questi inserimenti

avvengono con successo il metodo receive crea un oggetto response e gli attribuisce il tipo ACK che indica la corretta fine del metodo. Restituisce poi questa response al metodo Client che provvederà ad inviarla al client che ha fornito i dati.

Il client che ha inviato i dati riceve cosi la response positiva, e procede a sua volta a notificare al terminale che l'operazione è terminata con successo. Per farlo crea una response di tipo ACK e la restituisce al terminale tramite la classe Client. Il terminale da cui era partito il comando, in caso di ricezione del messaggio ACK stampa a schermo un messaggio che informa che il file è stato condiviso con il client specificato.

4.9 File di log

Una cosa essenziale in ogni software è il file di log, tramite il quale si può controllare il funzionamento di un nodo e la presenza di errori o anomalie. In caso di errori verificare dove si sono verificati questi errori all'interno del software e da cosa sono stati scatenati. Maggiore è l'accuratezza e maggiori sono i dettagli riportati in un file di log maggiore sarà la velocità con la quale si capirà il problema e minore il tempo impiegato per risolverlo. Proprio per questo si è utilizzato all'interno del software un sistema di log molto articolato. Per farlo si ricorre alla classe Logger che fa parte del package org.apache.log4j . Log4j è un progetto Open Source dell'Apache Software Foundation (ASF) che permette di monitorare un'applicazione Java durante il suo utilizzo qualora non fosse possibile utilizzare strumenti di debugging, si pensi ad esempio quando una applicazione è distribuita sul web o è di tipo multithreading. Il logging ha però una controindicazione: porta ad un decremento delle performance. Per alleviare questo difetto, log4j è stato sviluppato per essere veloce ed estendibile, permettendo vari livelli di granularità nel controllo del logging. Il vantaggio più importante che ha un sistema di logging ha rispetto a riempire il codice di System.out.println, è che è possibile controllare quando visualizzare gli statement di log. In altre parole è possibile dividere lo spazio dei log, che è lo spazio dove risiedono tutti gli statement di log, in categorie definite da criteri decisi dallo sviluppatore. La classe Logger rappresenta appunto uno spazio dei log che rispetta dei determinati criteri. Infatti nella cartella principale del nodo è situato il file darkcloud.log che contiene tutte le informazioni di cui nel codice si è deciso di tener traccia. Sono divise in categorie quali Message, Config, Info ed Error. Inoltre in tali file di log vengono indirizzati i messaggi di errore che il sofware genera. Per ogni errore o per capire meglio il funzionamento del programma si invita l'utilizzatore a leggere il file di log.

Capitolo 5

Risultati sperimentali

Al termine della fase di programmazione il programma Darkcloud risulta pienamente funzionante. Solitamente durante lo sviluppo di un software per verificarne il funzionamento e testare la funzionalità dei comandi si fanno continui test sulle macchine degli sviluppatori e su altre architetture. Anche con Darkcloud si sono svolti diversi test nella fase di development. Terminata questa ultima per testarne la qualità si sono effettuati dei test sulle prestazioni. I test di prestazioni sui software possono essere di diversi tipi, per Darkcloud si sono effettuati alcuni test sulla velocità e sul carico di lavoro. Sapere se la velocità di esecuzione di un comando è compatibile con i tempi di utilizzo normali è importante, in quanto se andassero a rallentare quello che è il lavoro dell'utilizzatore ultimo sarebbe controproducente. Oggi se un computer è troppo lento o un programma impiega troppo tempo, e l'utilizzatore è un professionista che ha bisogno di efficienza nel lavoro, questo ultimo non penserà due volte a cambiare prodotto. Un altro parametro importante è il carico di lavoro, cioè la fatica che il computer fa per per eseguire un certo comando. E' vero che oggi i computer sono molto potenti, ma l'aumento della portabilità dei dispositivi porterà anche a riduzioni della potenza di calcolo. Per quanto riguarda la compatibilità su diverse piattaforme e sistemi operativi Darkcloud è molto avvantaggiato, in quanto è scritto in java. Java è un linguaggio indipendente dalla piattaforma, infatti i programmi scritti in linguaggio Java sono destinati all'esecuzione sulla piattaforma Java, ovvero saranno lanciati su una Java Virtual Machine e, a tempo di esecuzione, avranno accesso alle API della libreria standard. Ciò fornisce un livello di astrazione che permette alle applicazioni di essere interamente indipendenti dal sistema su cui esse saranno eseguite.

5.1 Ambienti di test

I test sono stati effettuati principalmente su due sistemi. Il personal computer del tesista,che chiameremo calcolatore A, e il server universitario Capoeira. Le configurazioni hardware dei due sistemi sono molto differenti. Le componenti che faranno la differenza nelle prestazioni sono la cpu, la memoria e il disco fisso. Il calcolatore A ha una cpu Intel i5-750, un modulo di memoria ram da 4gb e disco rigido ssd. Il server Capoeira ha 2 cpu Intel E5540, 16gb di memoria ram e dischi scsi a 15000 rpm. La differenza che più sentiremo nei test riguarda i processori, infatti il server ha una capacità di calcolo superiore di quasi due volte a quella del calcolatore A. Il processore intel i5-750 infatti ha quattro core mentre ogni singolo E5540 ha quattro core, moltiplicato per due processori fa 8 core. Il calcolatore A ha però i core con frequenza fissa a 3.6 ghz mentre quelli di capoeira hanno una frequenza che varia in base alla richiesta. Il sistema operativo sul calcolatore A è Windows 7 64bit sul quale tramite VirtualBox

è stato usato in macchina virtuale il sistema operativo Debian 4.3.5 basato sul kernel linux 2.6.32. Si è optato per questa soluzione per semplificare l'utilizzo degli strumenti di sviluppo e test. Il server Capoeira invece ha installato Debian 4.5.3 basato sul kernel linux 3.0.0.

5.2 Risultati sperimentali

Per misurare i tempi impiegati dai comandi dall'avvio al termine dell'esecuzione si è impiegato il programma linux TIME. Questo comando anteposto ad un altro restituisce alla fine dell'esecuzione il tempo totale impiegato dal processo e la percentuale di cpu utilizzata per eseguirlo. I comandi utilizzabili al momento sono il ping, il put, il get e lo share. Il ping è un comando usato solo per verificare la disponibilità in rete di una macchina, quindi su di esso non sono stati fatti test prestazionali. Sono invece stati testati il put, il get e lo share. Un esempio di sintassi del test è il seguente

```
time python client.py -r PUT -h 127.0.0.1 -p 18100 -f prova.txt
-F prova.txt
```

I test sono stati fatti tutti creando i nodi in locale quindi non tengono conto di eventuali ritardi dovuti a latenze della rete. Il fattore delle latenze di rete non è quantificabile a priori, senza conoscere i casi di uso specifici, quindi per il momento è stato ignorato. Facendo test in locale si può capire come reagisce il software quando le istanze aumentano e il carico dei dati si incrementa. Si può dire che l'obbiettivo dei test è capire se il software gestisce bene l'aumento dei nodi, che comporta a seconda del comando un aumento della complessità dell'architettura e permette di valutare la scalabilità del programma.

Per testare il put si è creato in locale una architettura che comprendeva un nodo client e cinque nodi server. Si è iniziato con l'invio di un file ad un solo server, poi verso due server incrementando il numero fino all'invio a cinque server. Per ogni caso la misurazione è stata eseguita dieci volte e al termine raccolti i risultati si è calcolata la media. Il file inviato era sempre della stessa dimensione, cioè 1KB, in modo da uniformare i risultati. In questo test l'aumento del numero dei server comporta un aumento del codice che il client deve eseguire, osservare i tempi e i carichi dirà se il software è in grado di incrementare le proprie prestazioni o se vi sono colli di bottiglia che rallentano.

Ecco di seguito i test sul put.

n°server	tempo di esecuzione	uso cpu
1	1,2 sec	3%
2	1,9 sec	2%
3	2,6 sec	1%
4	3,3 sec	1%
5	4 sec	1%

Tabella 5.1: Test su Capoeira del comando PUT

I test eseguiti sul comando get vanno invece a testare il tempo di risposta e il carico di lavoro a cui si sottopone la cpu quando si devono recuperare dai server le informazioni per ricomporre il file. Anche in questo caso si sono utilizzati file di dimensione uguale, tutti da 1KB. Il programma utilizzato è stato time con una sintassi simile alla seguente:

n°server	tempo di esecuzione	uso cpu
1	1,8 sec	1%
2	2,8 sec	1%
3	4,1 sec	1%
4	5,7 sec	1%
5	6,1 sec	1%

Tabella 5.2: Test sul calcolatore A del comando PUT

```
time python client.py -r GET -h 127.0.0.1 -p 18100 -F prova.txt
```

Come nel get per testare la scalabilità del software nell'aumento della difficoltà computazionale si è incrementato gradualmente il numero di nodi dai quali si andava a prelevare l'informazione. In questo modo il programma deve eseguire più comandi. Prima si è recuperato dal client un file contenuto su un nodo, poi su due nodi incrementando fino a cinque nodi. Per ognuna di queste architetture si sono fatti dieci rilevamenti e i valori in tabella ne sono le medie. Si riportano di seguito le tabelle dei risultati dei test del get. L'ultimo comando da analizzare è lo share, a differenza del

n°server	tempo di esecuzione	uso cpu
1	1,4 sec	3%
2	2,1 sec	2%
3	2,3 sec	2%
4	2,8 sec	1%
5	3,6 sec	1%

Tabella 5.3: Test su Capoeira del comando GET

n°server	tempo di esecuzione	uso cpu
1	2,1 sec	1%
2	3,0 sec	1%
3	3,6 sec	1%
4	4,2 sec	1%
5	6,5 sec	2%

Tabella 5.4: Test sul calcolatore A del comando GET

put e del get questo comando non fa interagire nodi client con nodi server, ma solo nodi client tra di loro. Quindi nei test di questo comando non sarebbe utile ai fini dell'analisi della scalabilità l'aumento del numero dei server, ma piuttosto l'aumento del numero dei client con cui un nodo client voglia condividere un informazione. Per questo motivo in questi test si è utilizzato il comando share con un numero sempre maggiore di client come obbiettivo, prima con un solo client poi con due crescendo fino a cinque client. Per fare questo si è incontrato il problema che lo share accetta solo un altro client come destinatario dello share. Per ovviare questo si è realizzato uno script shell all'interno del quale sono stati riportati tanti comandi share concanetati dall'operatore & che nella console permette di eseguire i comandi in background. In questo modo si inviano contemporaneamente le richieste di share e le si fanno processare tutte

contemporaneamente. Usando poi il programma time seguito dallo script si ottiene il tempo totale dell'esecuzione dei comandi share. La sintassi del file script.sh per lo share delle informazioni di un file da un nodo verso altri due per esempio sarebbe

```
python client -r SHARE -h 127.0.0.1 -p 18100 -F prova.txt -H 127.0.0.1
-P 18101 & python client -r SHARE -h 127.0.0.1 -p 18100 -F prova.txt
-H 127.0.0.1 -P 18101
```

Mentre invece la sintassi del comando time con lo script è

```
time ./script.sh
```

Per ogni architettura di prova sono stati eseguiti dieci test e il valore riportato in tabella rappresenta la media. Il file condiviso era situato su un solo server in modo da poter focalizzare i test sulla scalabilità dello share e non della dimensione dei dati salvati, inoltre tale file era delle dimensioni di 1KB. Si riportano di seguito le tabelle dei test dello share.

n°client	tempo di esecuzione	uso cpu
1	1,4 sec	3%
2	1,5 sec	3%
3	1,6 sec	7%
4	1,7 sec	8%
5	2,2 sec	8%

Tabella 5.5: Test su Capoeira del comando SHARE

n°client	tempo di esecuzione	uso cpu
1	1,9 sec	1%
2	2,5 sec	1%
3	3,6 sec	1%
4	4,2 sec	1%
5	7 sec	1%

Tabella 5.6: Test sul calcolatore A del comando SHARE

Ad una prima analisi i risultati potrebbero sembrare strani visto che su Capoeira l'utilizzo della cpu aumenta mentre sul calcolatore A rimane basso. Questi valori sono dovuti al tipo di processori usati. Infatti il calcolatore A ha un processore da desktop con frequenza molto alta, risparmi energetici disattivati e alto consumo quindi ha il processore sempre ad un regime di lavoro alto. I processori dei server sono più dinamici dovendo pensare anche al consumo dato che sono in numero molto maggiore. La potenza data dall'utilizzo di 8 core nel server però si nota dai tempi molto più bassi. I risultati sono positivi anche per quanto riguarda la scalabilità in quanto l'aumento dei tempi è lineare e questo indica che il software gestisce bene l'incremento di complessità computazionale.

Capitolo 6

Conclusioni

Questa tesi ha voluto analizzare la nascita del progetto Darkcloud nelle fasi di studio, di analisi, di realizzazione della struttura di rete e implementazione delle funzioni avanzate.

La prima fase quella di analisi ha portato alla conoscenza di molti aspetti che sono tornati utili per il progetto. Ad esempio avere una rete decentralizzata è utile per la sicurezza e la stabilità, ma avere nodi che fungano sia da server che da client come nel p2p puro porta a rallentamenti delle prestazioni se si vuole un buon grado di anonimato ed espone a molte vulnerabilità. Ancora avendo compreso il funzionamento delle DHT e il meccanismo di indicizzazione di Freenet è parso chiaro che per un software con un bacino di utenza piccolo la difficoltà di programmazione sarebbe stata troppo elevata. Un importante decisione è stata quella di optare per una rete statica, dove i nodi vanno aggiunti manualmente, è vero che sarebbe più comodo un modello dinamico, ma come insegnano le reti friend to friend questo va a discapito della sicurezza. Capire come la topologia di una rete influisce sulla scalabilità delle prestazioni ha imposto l'adozione di due tipi di nodi con compiti ben distinti.

Per poter delineare la struttura ad alto livello del software si è dovuto valutare bene le caratteristiche che questo doveva avere. Ogni decisione in fase di progetto è un compromesso, bisogna valutare cosa è meglio in base alle esigenze dell'utente ultimo. La scelta del linguaggio Java ha il vantaggio della portabilità e della disponibilità di librerie molto fornite, però essendo un linguaggio indipendente dalla piattaforma ha prestazioni minori di C++. Darkcloud dovendo funzionare su personal computer non ha problemi di limitatezza di risorse, questo ha fatto scegliere Java. Decidere che tipo di interfaccia usare è un importante scelta, chiaramente se si fosse optato per un eseguibile con interfaccia integrata si sarebbe ottenuta una maggior velocità di sviluppo, ma un eseguibile che si comandi tramite terminali esterni sarà un grande vantaggio quando dovrà essere utilizzato da una utenza diversificata. Avere un terminale su pagina web, su tablet e altri terminali alternativi facilita ed estende l'uso di Darkcloud. In fase di progetto è anche stato scelto che tipo di comandi realizzare, l'uso di comandi semplici o articolati, favorendo comandi semplici si è data l'opportunità a futuri sviluppatori di interfacce di realizzare comandi più complessi e articolati.

Una volta che le specifiche del codice erano ormai nitide si è proceduto ad un lavoro in team. La prima parte della programmazione ha riguardato la struttura di base con la definizione dei diversi tipi di entità e dei meccanismi di comunicazione tra di essi, quindi la definizione di un protocollo. Una volta che la base è stata funzionante si è passato alla realizzazione delle funzioni tramite le quali i nodi eseguivano tra di loro procedimenti complessi. Non sono mancate le difficoltà, una delle maggiori riguarda la crittografia. Infatti si è presentato l'ostacolo del formato delle chiavi e dei dati criptati complicato

dal fatto che tutti i metodi dialogano tra di loro con dati in base 64. Ma a parte queste piccolezze l'aver progettato fin dall'inizio un buon protocollo e aver per prima cosa scritto una struttura del programma prestando attenzione all'incapsulamento dei dati e alla modularità ha semplificato notevolmente l'implementazione di funzioni complesse.

La fase di test dell'applicazione infine ha rivelato che il software reagisce bene ad un uso intenso. Anche se i test sono stati svolti su una sola macchina per volta, quindi la potenza disponibile era molto meno di quella che una rete metterà a disposizione, i risultati fanno capire che la scalabilità del carico di lavoro è ottima. I tempi brevi garantiscono che in un caso di utilizzo reale con molti nodi funzionanti i tempi di attesa riguarderanno le latenze di rete e non tempi di software. Le percentuali di utilizzo della cpu rivelano anche che il carico di lavoro è davvero minimo.

Alcuni sviluppi futuri tramite cui si potrebbe far crescere questo progetto sono lo studio di librerie di alcuni servizi di cloud computing per realizzare versioni ad hoc di Darkcloud da eseguire a livello SaaS, potrebbe essere utile aggiungere alla funzione share la possibilità di accettare condivisioni anche quando il destinatario è offline tramite una lista di condivisioni in attesa, lo sviluppo di terminali che permettano di usare Darkcloud su tablet o smartphone infine sarebbe utile implementare un codice di correzione degli errori che si occupi di ricostruire dati danneggiati nelle comunicazioni.

Il programma Darkcloud è un ottima base per reti di condivisione dati in modo sicuro, e senza dubbio sarà molto utile a chi svilupperà reti simili in futuro. Il codice è rilasciato in licenza GNU GPL 3 e quindi potrà essere riutilizzato tutto o in parte da altri utenti, non che migliorato o modificato dagli utilizzatori.

Rappresenta una novità nel settore usando la frammentazione delle informazioni su una rete anonima in abbinamento al cloud computing per il mantenimento delle stesse in sicurezza.

Spero che questo trattato vi abbia illuminato sul funzionamento del software e possa aiutarvi sia a comprenderne le meccaniche sia a guidarvi nell'utilizzo.

Gionatan Fortunato

Bibliografia

[1] Biddle Peter, England Paul, Peinado Marcus, Willman Bryan.
The Darknet and the Future of Content Distribution.
ACM Workshop on Digital Rights Management. Washington, D.C.: Microsoft Corporation.(2002)

[2] Ian Clarke, Oskar Sandberg, Brandon Wiley, Theodore W. Hong.
The Darknet and the Future of Content Distribution.
ACM Workshop on Digital Rights Management. Washington, D.C.: Microsoft Corporation.(2000)

[3] Nathan S. Evans, Christian Grothoff.
Randomized Recursive Routing for Restricted-Route Networks.
http://grothoff.org/christian/nss2011.pdf(2011)

[4] Nathan S. Evans, Christian Grothoff.
Beyond Simulation: Large-Scale Distributed Emulation of P2P Protocols.
http://grothoff.org/christian/cset2011.pdf(2010)

[5] Frank Dabek.
A Distributed Hash Table.
http://pdos.csail.mit.edu/papers/fdabek-phd-thesis.pdf(2005)

[6] Ren´e Brunner.
A performance evaluation of the Kad-protocol.
http://www.eurecom.fr/ btroup/BThesis/MasterThesisBrunner.pdf(2006)

[7] S. Adler.
The Slashdot effect: an analysis of three Internet publications.
http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html(2000)

[8] Tasini A.
Crittografia simemtrica: il sistema AES..
http://amslaurea.cib.unibo.it/2641/1/tasini_andrea_tesi.pdf(2011)

[9] Alex Biryukov and Dmitry Khovratovich.
Related-key Cryptanalysis of the Full AES-192 and AES-256.
http://impic.org/papers/Aes-192-256.pdf(2009)

[10] Andrea Pasquinucci.
La crittografia a chiave pubblica è a rischio?.
http://www.ucci.it/docs/ICTSecurity-2002-01.pdf(2002)

[11] USA Federal Information.
Announcing the Advanced Encryption Standard(AES).
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf(2001)

[12] Öznur Altintas, Niclas Axelsson.
Combining Bittorrent with Darknets for P2P privacy.
http://user.it.uu.se/ victor/SCS10/Oneswarm.pdf(2002)

[13] Marco Balduzzi, Jonas Zaddach, Davide Balzarotti, Engin Kirda, Sergio Loureiro.
A Security Analysis of Amazon's Elastic Compute Cloud Service.
http://www.iseclab.org/people/embyte/papers/securecloud.pdf(2011)

[14] Micheal Fenn, Jason Holmes, Jeffrey Nucciarone.
A Performance and Cost Analysis of the Amazon Elastic Compute Cloud (EC2) Cluster Compute Instance.
http://rcc.its.psu.edu/education/white_papers/cloud_report.pdf

[15] Juraj Somorovsky, Mario Heiderich, Meiko Jensen, Jörg Schwenk Chair.
All Your Clouds are Belong to us – Security Analysis of Cloud Management Interfaces.
http://www.nds.rub.de/media/nds/veroeffentlichungen/2011/10/22/ AmazonSignatureWrapping.pdf

[16] Cay S.Horstmann, Gary Cornell.
Java 2 Sesta Edizione.
McGrawHill.(2004)

[17] Domenico Beneventano, Sonia Bergamaschi, Francesco Guerra, Maurizio Vincini.
Progetto di basi di dati Relazionali.
Pitagora Editrice Bologna.(2007)

[18] `http://www.businessmagazine.it/articoli/3179/sicurezza-informatica-pochi-anni-all-armageddon_index.html`
Articolo sul rapporto clusit 201 (2012)

[19] `http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html`
Sito di riferimento comunity sviluppatori Gnutella

[20] `http://opennap.sourceforge.net/napster.txt`
Fonte materiale studio protocollo Napster

[21] `http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html`
Documenti protocollo Kademlia

[22] `https://freenetproject.org/`
Sito comunity Freenet Project

Elenco delle figure

Ringraziamenti

Per prima vorrei ringraziare la mia famiglia che mi ha dato la possibilità di continuare gli studi.
Poi vorrei ringraziare i miei compagni di università, quelli della BSI e quelli della BandaBassotti, con cui ho diviso difficoltà e gioie in questi anni. Infine tutti i miei amici che mi hanno sostenuto in questa impresa in particolare Simone Franchini e Simone Mancuso.
Ringrazio anche il professor Michele Colajanni per avermi permesso di fare una tesi di notevole importanza e molto interessante, e l'ing. Fabio Manganiello che mi ha assistito, consigliato e soprattutto sopportato in tutta l'attività di tesi.

Un sentito grazie a tutti !