UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"SVILUPPO DI UN WEB SERVER BASATO SU MECCANISMI PEER-TO-PEER"

Relatore: Prof. Mauro MIGLIARDI

Laureando/a: CAZZARO Mirco

ANNO ACCADEMICO 2021 – 2022

Data di laurea 17/03/2022

CONTENUTI

INTRODUZIONE

Negli ultimi 2 decenni l'accelerazione che ha avuto la digitalizzazione delle attività produttive, spinta dal dover stare al passo con i tempi, da una concorrenza spietata e da finanziamenti di tipo governativo, ha mosso ogni azienda, dalle grosse multinazionali ai liberi professionisti a spostare diverse attività (gestionali, logistiche e di vendita) su internet.

In particolare, spostare le attività di vendita e di marketing su internet oramai sono un obbligo implicito per qualunque attività economica.
Si è visto, in particolare negli ultimi 2 anni di pandemia, quanto sia diventato cruciale e preponderante il flusso economico generato da vendite online oppure tramite canale diretto ma derivate da advertisement in rete.

Analizzando il fenomeno e le modalità in cui esso si manifesta da un punto di vista informatico, si osserva che, oramai, lo "strumento" tramite il quale gli utenti della rete internet vengono a conoscenza, acquistano e si mettono in contatto con ogni sorta di attività economica è il web.

Tuttavia, esistono ancor'oggi aziende tipicamente molto piccole che per settorialità dei loro clienti e/o dei loro prodotti non si sono mai avvicinate al mondo digitale, probabilmente per paura che i costi d'ingresso sommati ai costi periodici di mantenimento di un portale web non siano al livello dei potenziali guadagni derivanti dall'investimento.

L'obbiettivo di questa tesi, dunque, è formulare un ipotetico sistema che annulli o contenga i costi di hosting di un server web, basando i contenuti del portale su una rete peer to peer piuttosto che in un unico server centralizzato che determina elevati costi di gestione da parte del committente.

Il progetto di questo sistema non è nuovo, ma si basa sul lavoro fatto dal collega Davide MARTINI, laureatosi 3 anni fa presso l'Università degli Studi di Padova.
Il focus del lavoro che andrò a presentare con questa tesi è una sostanziale modifica del lavoro fatto dal collega, che verrà spiegato nel Capitolo 1, sostituendo il sistema utilizzato da lui per la condivisione di risorse (IPFS – InterPlanetary File System), con un sistema più recente, chiamato ZeroNet, che verrà descritto nel Capitolo 2 e che a differenza di IPFS nasce specificamente per la condivisione di contenuti web, mentre la genericità di IPFS ad essere adattato a diversi protocolli ne mina le potenzialità.

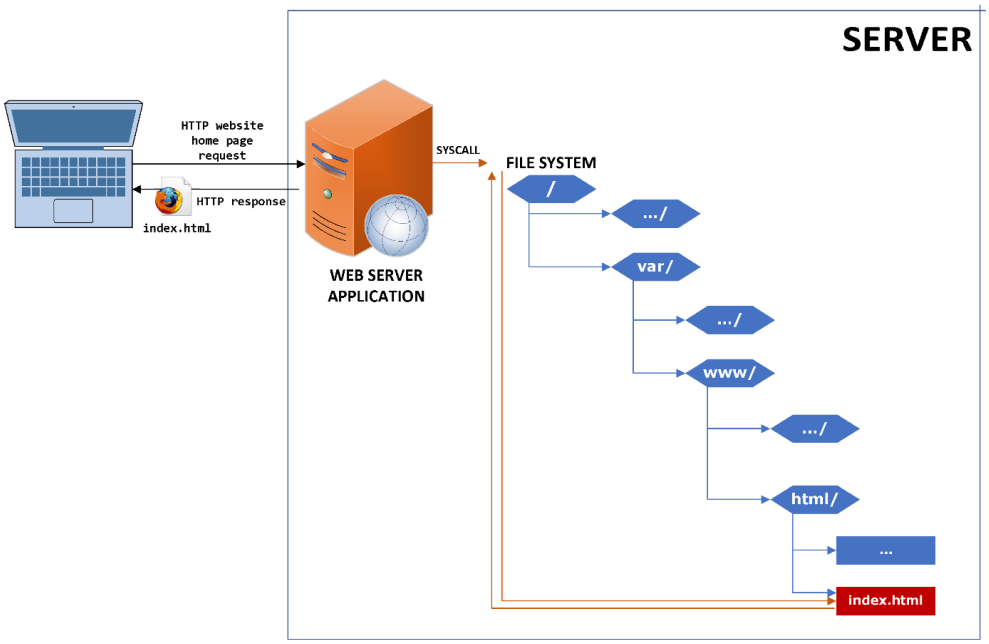
Il lavoro che verrà descritto in seguito è dunque orientato a spiegare come ho adattato ZeroNet al sistema di MARTINI, e inoltre, mostra come possa essere usato diversamente dallo scopo per cui è stato ideato e dunque in maniera del tutto trasparente per l'utente finale che intende accedere al portale web.

CAPITOLO 1

Il lavoro del collega MARTINI è stato in primo luogo quello di individuare un sistema strutturato che rispondesse al requisito principale, ovvero quello di distribuire i contenuti tra più peer appartenenti al progetto. In secondo luogo, l'impegno è stato quello di individuare un sistema che fornisse un metodo efficiente di validazione delle modifiche e che ne tenesse traccia, e che fornisse un sistema di voto per approvare, eliminare, o escludere contenuti ed utenti. [1]

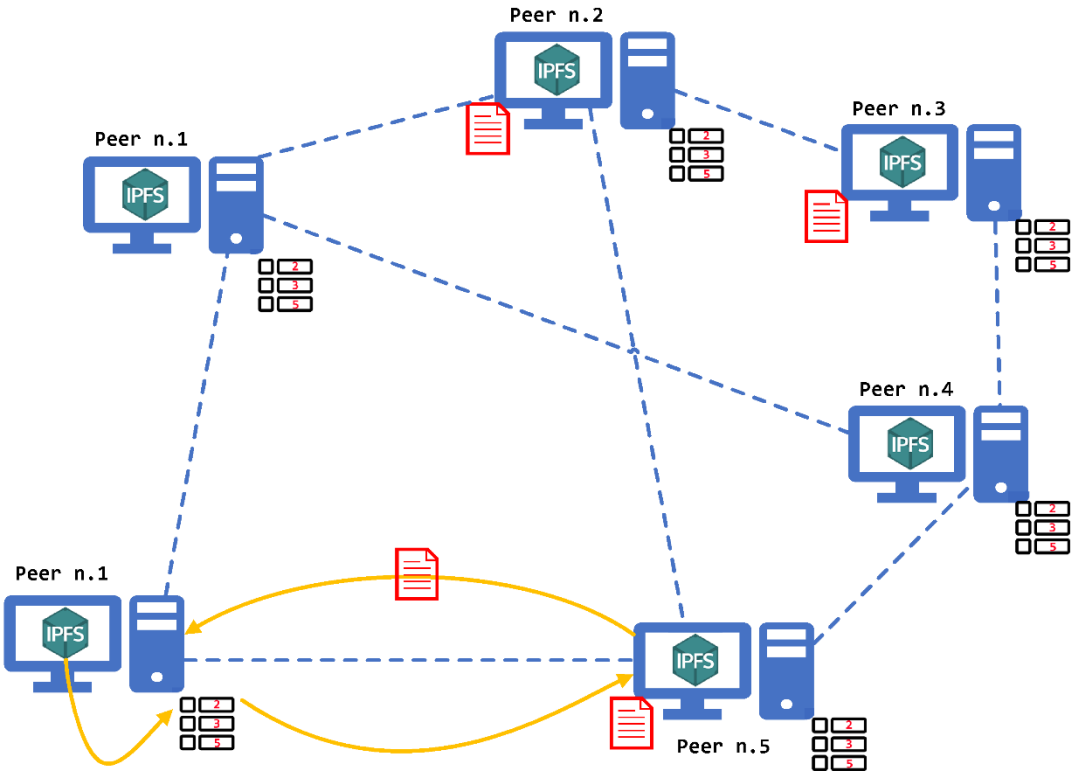
1.1 IPFS

InterPlanetary File System è un protocollo che astrae, ad un livello più alto, un file system. Molti dei protocolli più diffusi per la condivisione di file basata sul modello client server (es. HTTP, FTP, SMB, ecc.…) usano un sistema di recupero delle risorse definito "location-based": tipicamente, un client richiede una risorsa tramite un URL (Uniform Resource Location) che indica dove cercare all'interno dell'albero del file system uno specifico file. Per esempio, nel protocollo HTTP, l'apertura della home page di un sito richiede al web server di rispondere con il file index.*, che si trova all'interno della cartella htdocs. Quindi, il web server accederà, ad esempio, al ramo /var/www/html/index.html (su sistemi linux) dell'albero delle directories tramite una system call, e risponderà sulla connessione con il client con il file recuperato in seguito alla richiesta.



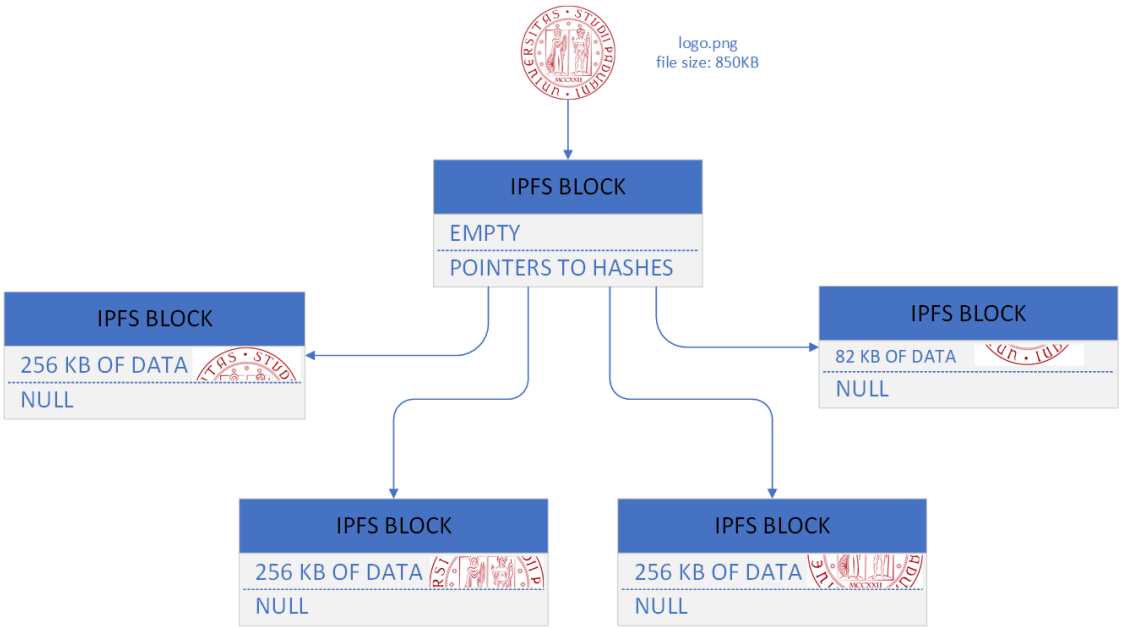
IPFS, invece, mira ad eliminare i server come li conosciamo oggi, ovvero macchine posizionate tipicamente in server farm controllate da grandi provider, che comportano costi, con scarsa

fault tolerance, per quanto si tratti di strumentazione all'avanguardia, e soggetta anche a politiche territoriali quali censure distribuendo e replicando i contenuti tra più pari.
Ogni host sulla quale è installato il protocollo IPFS richiede risorse non più con un meccanismo "location based", ma bensì usando un meccanismo "content based": ogni peer consulta una cosiddetta DHT (distributed hash table, verrà esplicata meglio in 3.1) dalla quale possono ricavare informazioni su quali altri peer dispongono della risorsa richiesta. La hash table associa hash che si riferiscono alla singola risorsa e gli host che hanno salvato tale risorsa. A quel punto, il richiedente inizia ad inoltrare richieste per tale risorsa agli altri peer (seguendo politiche per scegliere in ordine i peer "più vicini"), e il primo che risponde invierà la risorsa richiesta. A quel punto, l'host che ha fatto la richiesta diventa un nuovo peer in grado di fornire tale risorsa, aggiungendo alla DHT una voce relativa alla disponibilità anche da parte sua di essa.



Non necessariamente le richieste per un singolo file devono essere gestite da un unico host all'interno della rete.
I file, infatti, non vengono salvati interamente, ma suddivisi in blocchi, ognuno dei quali è associato ad un hash. Ogni blocco contiene dati relativi al file e puntatori agli hash dei blocchi successivi.
La dimensione massima di un blocco all'interno del protocollo IPFS è di 256KB: per file di dimensione inferiore a tale limite, un singolo blocco è sufficiente per contenerli. In tal caso, i puntatori ai blocchi successivi saranno nulli.
Nel momento in cui un singolo file supera tale dimensione, IPFS salva tale file in una struttura

ad albero avente radice senza contenuto di dati, con altezza unitaria ed arietà pari a $\left\lceil \frac{\text{fileDimension}}{256\text{KB}} \right\rceil$.
Nel caso in cui invece ad essere salvata sia una cartella di dimensione superiore a 256KB, la forma è quella di un albero dove la radice non contiene dati, i file sono suddivisi in blocchi figli della radice con la regola vista in precedenza e, in caso vi siano sottocartelle, esse verranno rappresentate con dei blocchi vuoti che saranno, ricorsivamente, radici dei sottoalberi che conterranno altri file e cartelle
Nello schema sottostante viene rappresentato il primo caso:



Pertanto, nel momento in cui viene richiesta una risorsa che risiede tra più host, per come è strutturato il protocollo, diversi blocchi della stessa risorsa possono essere richiesti simultaneamente aumentando il throughput totale.

1.2 ETHEREUM

Ethereum è un sistema di blockchain che oltre a fungere da libro mastro di transazioni in Ether, la criptovaluta associata, permette di stipulare degli “smart contract”, ovvero accordi digitali, scritti in un linguaggio di programmazione associato alla blockchain stessa chiamato “Solidity”, tramite il quali si possono sottoscrivere delle condizioni tra due o più parti (utenti della blockchain Ethereum).
Ethereum permette inoltre di registrare nei vari blocchi della lista, oltre a semplici transazioni, anche il possesso di risorse (files) che prendono il nome di NFT (non-fungible tokens).
Tutto ciò che è possibile gestire tramite blocchi della blockchain di Ethereum può essere oggetto di contratto intelligente realizzato in Solidity. Nella realtà di riferimento, infatti, la possibilità di autenticare chi ha creato e quindi possiede un dato file, permette di imporre condizioni su quali file è consentito pubblicare su una certa piattaforma. In caso tali condizioni non vengano rispettate (analogamente a quanto avviene in fase di accettazione delle condizioni di utilizzo di una classica piattaforma di tipo Client-Server) l’utente trasgressore è

passibile di un trattamento imposto dal contratto smart che lui stesso ha sottoscritto.

1.3 INTEGRAZIONE DELLE DUE TECNOLOGIE

Il collega Martini, nella sua soluzione, è riuscito a creare una netta separazione tra le due problematiche, individuando poi la soluzione per unire le due infrastrutture e garantire un corretto funzionamento del sistema.
Prima di evidenziare il funzionamento del sistema di gestione dei contenuti, è necessario sottolineare che qualsiasi applicazione web porta i suoi vantaggi quando non è richiesto alcun intervento (configurazione del client, installazione di software, ecc.) da parte dell'utente finale. Pertanto un qualsiasi servizio web e cloud ha motivo d'essere solo quando l'utente può utilizzare il sistema aprendo un qualsiasi browser.

La soluzione presentata dal collega, infatti, prevede in primo luogo il recupero dei contenuti attraverso un gateway, il quale interfaccia un web server e il protocollo IPFS: ogni richiesta HTTP che viene ricevuta dal server viene tradotta in una interrogazione IPFS, diffusa tra i peer della rete e inoltrata attraverso il web server.
Tra i contenuti che vengono recuperati dal gateway (HTML, CSS, JS, JSON) necessari alla corretta rappresentazione del portale web, vengono recuperati gli smart contract che contengono le regole per un utilizzo condiviso del sistema.

Infine, ogni qualvolta l'utente desidera eseguire delle operazioni che richiedono la verifica del rispetto delle policies del sistema condivise dal network di partecipanti, o eventualmente una votazione per l'approvazione per l'inserimento di nuovi contenuti, definite nello smart contract, vengono eseguite le procedure e/o transazioni sulla blockchain Ethereum, permettendo o eliminando richieste di pubblicazione, accesso o modifica dei contenuti.

L'interazione tra blockchain e sistema web è resa possibile da una libreria javascript (Web3.js), la quale però, essendo una semplice interfaccia, richiede che l'utente finale faccia parte del network Ethereum, e ciò non è assolutamente garantito. Tale problematica contrasta con quanto sottolineato in precedenza, obbligando l'utente finale a installare del software aggiuntivo.

Una soluzione efficiente è quella di utilizzare come browser web MetaMask, un browser che integra all'interno la gestione dei portafogli Ethereum, e che permette di eseguire script injection sulle pagine che si visitano, permettendo di iniettare direttamente la libreria Web3.js.

CAPITOLO 2

L'obbiettivo di questa tesi è quello di fornire una valida alternativa al sistema di condivisione di risorse peer-to-peer utilizzato con un sistema che nasca specificamente per il web, e non come generico sistema di condivisione di risorse. Tra i vari sistemi esistenti, quello che risponde alla maggior parte dei requisiti è *ZeroNet*, un sistema per la condivisione di contenuti web tra pari che integra una web application, che fornisce i servizi web di base, ad un vero e proprio protocollo per lo scambio di risorse peer-to-peer.

2.1 INTRODUZIONE A ZERONET

ZeroNet è un sistema open-source sviluppato in Python che funge sia da client che da server web in una rete di tipo peer-to-peer, dove diversi utenti che appartengono alla rete possono scambiarsi contenuti web statici e dinamici. Similmente ad IPFS e a tutti i sistemi di condivisione di risorse P2P, anche questo sistema è "content-based", ovvero le risorse (web in questo caso) vengono indicizzate e recuperate direttamente, senza riferimenti espliciti alla loro posizione: questo perché in ZeroNet ogni sito che viene visitato è poi replicato nel client che effettua la richiesta, e su base volontaria del client esso stesso può diventare hoster di tali risorse.

2.2 INSTALLAZIONE DEL SOFTWARE E CREAZIONE DI UN SITO WEB

Per quanto concerne Linux, ZeroNet può essere installato utilizzando il gestore di pacchetti *Snappy*, il quale permette di risolvere in maniera molto agevole la dipendenza da moduli e librerie esterne, senza causare conflitti con librerie già in uso da altri software.

Una volta installato il sistema, sarà possibile avviare il software da riga di comando tramite l'alias `zeronet`.

Per creare, successivamente, un proprio portale web hostato nella rete è sufficiente eseguire il comando
`zeronet siteCreate`.
In questo istante ZeroNet ci restituirà la chiave pubblica (che corrisponde all'indirizzo del sito stesso) e la chiave privata, necessaria in fase di pubblicazione e aggiornamento dei contenuti del sito.

Ogni modifica ad ogni file del sito, infatti, dovrà necessariamente essere validata tramite tale chiave usando il comando `zeronet siteSign <chiave_pubblica>`.

In seguito a questo comando, ZeroNet richiederà la chiave privata, ed in seguito ricalcolerà gli hash di ogni singolo file all'interno del file content.json.

2.3 BITCOIN E FIRMA DEI CONTENUTI

Il problema della gestione delle risorse replicate in un sistema distribuito è quello di prevenire che un peer modifichi deliberatamente una risorsa di cui non è il proprietario, poiché nel momento in cui un peer si trovi a dover richiedere tale risorsa, potrebbe non ottenere ciò che sta cercando, oppure in uno scenario perfino peggiore potrebbe subire l'attacco di malware.
Questo requisito verrà poi allargato, nella nostra realtà, per prevenire che persone aderenti al progetto modifichino deliberatamente il portale web in autonomia senza autorizzazione degli altri partecipanti.

Alla base del sistema di protezione dei contenuti di ZeroNet vi è il concetto di firma digitale utilizzando una coppia di chiavi, pubblica e privata. Tipicamente quando si parla di chiavi pubbliche e private è quando si parla di un sistema di crittografia asimmetrica: un utente che intende ricevere messaggi da un altro utente genera una coppia di chiavi matematicamente collegate tramite una funzione, una pubblica ed una privata. A quel punto, l'utente che ha generato le chiavi invia la chiave pubblica al mittente del messaggio, il quale codificherà il suo messaggio (variabili, file, ecc.… che vengono scritti su una connessione) con la chiave pubblica ed invierà il messaggio. Al momento della ricezione il messaggio, incomprensibile da chiunque, verrà decodificato dal destinatario con la chiave privata generata precedentemente, e non condivisa con nessuno. In questo modo nessuno che sia in possesso della chiave privata potrà leggere il contenuto di tale messaggio.

Il concetto di firma digitale funziona in maniera speculare al sistema appena descritto: sfruttando sempre una coppia di chiavi pubbliche e private, l'utente prima di inviare un messaggio (che può o meno essere cifrato, ai fini della firma digitale è assolutamente irrilevante), applica al messaggio (tipicamente non direttamente su di esso ma su una sua sintesi dopo aver applicato una funzione di hash) la chiave privata, ottenendo così una stringa chiamata firma. Insieme al messaggio (in chiaro o cifrato) viene inviata tale firma al destinatario che possiede la chiave pubblica. Il destinatario, dunque, applicherà la chiave pubblica sulla firma e se il valore restituito coincide con la sintesi del messaggio, il destinatario può essere sicuro della provenienza.

Come anticipato, alla creazione di un sito, ZeroNet tramite l'algoritmo BIP32 ci restituisce l'indirizzo di un wallet BitCoin con relativa password. Questa coppia di dati si adatta perfettamente alla necessità di firmare digitalmente i contenuti utilizzando l'indirizzo del wallet come chiave pubblica e la password come chiave privata per firmare le modifiche, in quanto esiste una relazione matematica tra le due che permette di codificare file con una e decodificarne il contenuto con l'altra.
Alla creazione di un sito (ovviamente senza alcun contenuto) ZeroNet inserisce nella nuova directory un file chiamato content.json: i file json sono file di parametri tramite i quali diverse applicazioni possono comunicare usando un formato standardizzato. Tale file contiene diverse voci, come la versione di ZeroNet utilizzata per creare il sito oppure il timestamp del momento in cui il sito è stato pubblicato.
I campi più importanti, necessari per il funzionamento del sistema di firma digitali, sono però altri:

- per ogni file è salvato nome, hash generato tramite algoritmo sha512, e lunghezza espressa in byte;

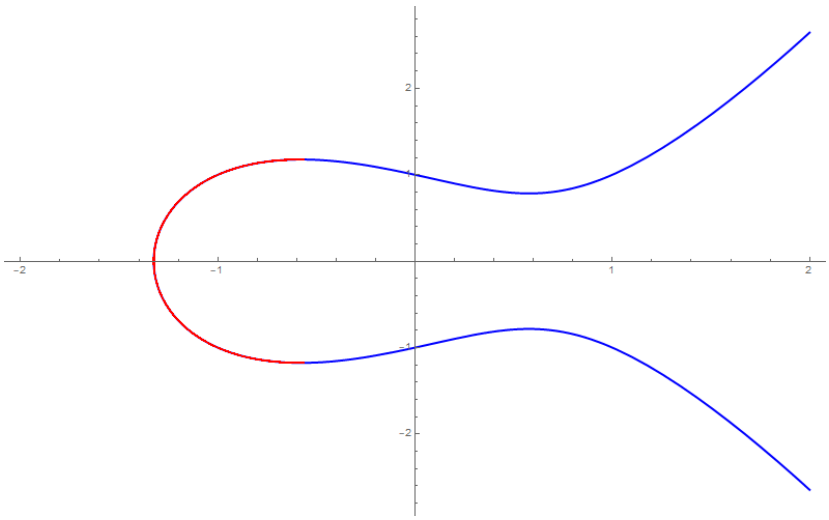
- la firma digitale.

Il meccanismo descritto in precedenza si applica molto bene nel caso in cui ogni messaggio da firmare sia composto da un singolo file: è infatti molto semplice, in molti linguaggi, applicare qualsivoglia algoritmo di hashing su un file.
Nel nostro caso, però, ad ogni caricamento di un sito dobbiamo trasferire diverse decine di file diversi:
una strategia ammissibile sarebbe quelle di firmare singolarmente ogni file, ma facendo in tal modo si rischierebbe di aver un overhead eccessivo.
Perciò, la strategia adottata è stata quella di firmare unicamente il contenuto del file content.json, inserendo la firma nello stesso al termine del file, il quale è composto da informazioni varie (versione, timestamp, ecc.…) e gli hash sha512 di ogni singolo file.

Il client che intenderà visitare il sito, dunque, riceverà i file tra cui il file content.json: questo verrà letto, verrà recuperata la firma la quale verrà decodificata con la chiave pubblica.
In seguito verranno calcolati gli hash sha512 di ogni singolo file e verranno comparati con il contenuto appena decodificato.

Al termine di questi controlli, se non sono stati rilevati errori, sarà possibile confermare che il contenuto del sito ricevuto sia effettivamente ciò che il proprietario intendeva trasmettere, e i contenuti mostrati all'utente.

L'algoritmo utilizzato da ZeroNet per codificare nella firma il contenuto del file content.json con la chiave pubblica, e in seguito decodificarla è l'algoritmo ECDSA (= elliptic curve digital signature algorithm) , ovvero lo stesso algoritmo usato dal sistema BitCoin per validare le transazioni tra utenti della blockchain omonima. La scelta è stata dettata dal fatto che le chiavi sono generate tramite algoritmo BIP32 (BitCoin), e dunque sviluppate "su misura" per ECDSA.
Rispetto al più diffuso RSA, questo algoritmo risulta più efficiente in quanto garantisce pari livello di protezione usando però molti meno bit per la chiave.
Inoltre, uno dei grossi vantaggi di questo algoritmo è che nell'ipotesi in cui qualcuno riesca ad avere una versione di un messaggio in chiaro e la sua versione cifrata tramite ECDSA con chiave privata, risulta estremamente difficile risalire alla chiave privata.
Nello specifico, tale algoritmo definisce una curva, tipicamente definita come $y^2 = x^3 + ax + b$, dove a e b sono parametri che possono essere impostati da implementazione a implementazione.
La rappresentazione della curva è la seguente:



(Elliptic Curve con parametri a = -1, b = 1)

La funzione, così com'è definita gode di 2 proprietà:

- è simmetrica rispetto all'asse x;
- se seleziono 2 punti, uno nella parte evidenziata in rosso ed uno nella parte evidenziata in blu, la retta passante intersecherà la curva in almeno un altro punto.

Così facendo, selezionando 2 punti A e B, definiti sulla base del contenuto da cifrare, viene individuato un terzo punto C sulla curva.
Poiché la funzione è pari, tale punto viene proiettato nella parte di curva che si trova nel semipiano opposto.
Iterativamente viene ripetuta tale operazione tra il punto A e la proiezione del punto prodotto nell'iterazione precedente, per un numero n di volte definito dalla chiave privata.
Il punto Z ottenuto dopo n iterazioni sarà la versione cifrata del messaggio tramite ECDSA.

Tipicamente in sistemi a chiave asimmetrica è necessario stabilire un metodo di condivisione delle chiavi pubbliche, tra chi la genera e chi dovrà usarla per verificare, in questo caso, l'autorevolezza dei contenuti
Tale problema non sussiste in ZeroNet in quanto la chiave pubblica coincide con l'indirizzo usato per richiedere il sito: dunque, un utente che richiede il sito è già in possesso della chiave necessaria a verificare la firma. **[2]**

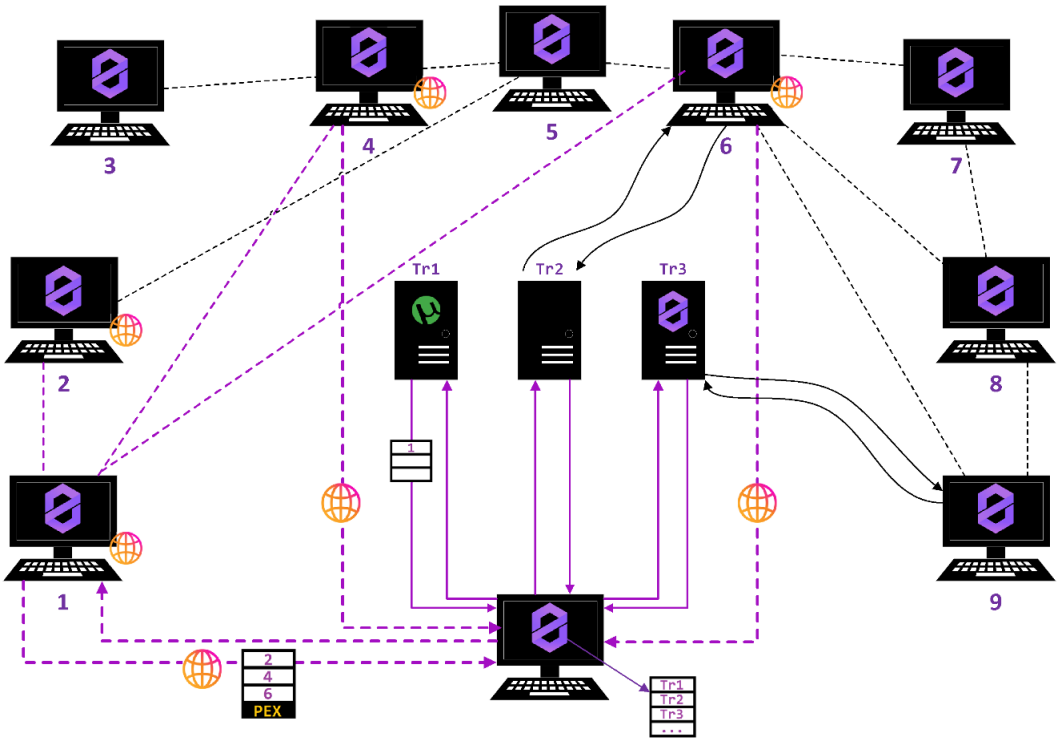
2.4 NEGOZIAZIONE DELLE CONNESSIONI

Ogni sistema di condivisione di risorse di tipo peer to peer deve prevedere un sistema per permettere ai vari peer di scoprire quali di essi possiedono le risorse (o parti di esse) richieste, e quali di essi sono disponibili al momento della richiesta
Il protocollo IPFS utilizza delle DHT (distributed hash tables), come spiegato al capitolo 1.2, ovvero una struttura dati distribuita e aggiornata che tiene traccia di quali peer posseggano quali risorse associate ad un certo hash.
ZeroNet, viceversa, usa due modalità distinte:

-Tracker: i tracker sono server atti a negoziare le connessioni tra i vari peer della rete. I tracker, dunque, tengono traccia di quali peer posseggono quali risorse e quali di questi peer sono disponibili nell'istante della richiesta. Poiché una stessa risorsa in una rete peer to peer viene scaricata da più punti diversi (se sono disponibili diversi peer), i tracker aiutano i peer a riassemblare efficientemente le risorse una volta ricevute.
Infine, trackers e peer comunicano periodicamente durante la trasmissione per informare il peer di nuovi peer eventualmente più veloci e, in tal caso, riorganizzare le connessioni.

Questo modello viene informalmente definito "Hybrid p2p" ed è la versione tradizionale del protocollo bittorrent, tanto è vero che ZeroNet stesso usufruisce di alcuni server tracker del protocollo bittorrent.
Sebbene efficace ed efficiente, questo modello non raggiunge la completa indipendenza da nodi centralizzati della rete. Pertanto, qualora venga preso il controllo di tali server, o alcuni di essi risultino non disponibili, l'intera rete ne risentirebbe in termini di sicurezza e di prestazioni.

- Peer eXchange (PEX): il protocollo peer exchange viene usato a supporto del tradizionale sistema ibrido p2p tramite tracker, riducendo il carico del lavoro di essi, e nel caso di una rete abbastanza estesa, impedire a terzi che prendano il controllo malintenzionato del server tracker di gestire le connessioni tra i vari peer. Il protocollo PEX permette ai peer appartenenti alla rete di scambiarsi direttamente ed autonomamente liste di peer attivi. L'utilizzo di questo protocollo permette inoltre di mantenere attivo il sistema anche in caso di brevi interruzioni del servizio dei tracker. [3]



Solo alcuni dei server tracker da cui i peer recuperano informazioni inizialmente circa le connessioni da instaurare sono dedicati esclusivamente al servizio ZeroNet: infatti, una parte

di essi servono anche il protocollo bittorrent per la trasmissione e la condivisione di file. Altri ancora invece non appartengono ad alcun servizio specifico, ed hanno carattere del tutto generico.

2.5 GESTIONE DELLE RISORSE

Una prima differenza con il protocollo IPFS si evidenzia in come le due tecnologie gestiscono i file. Sebbene in entrambi i protocolli, per avere un corretto funzionamento del sistema le risorse debbano essere replicate tra i vari peer della rete, il modo in cui le risorse vengono salvate, recuperate e protette varia nettamente. [4]

-Salvataggio delle risorse:

Per quanto riguarda il modo in cui le risorse vengono salvate nei vari peer, mentre il protocollo IPFS utilizza dei Merkle DAG (1.1.2), Zeronet si basa semplicemente sull'utilizzo del file system offerto dal sistema ospitante: il contenuto di un sito web che viene visitato viene collezionato all'interno di una cartella con nome corrispondente alla chiave pubblica del sito visitato; ognuna di queste cartelle si trova all'interno della cartella "PARENT_DIRECTORY/zeronet/common/data/". Ciò implica che i contenuti (HTML, CSS, JS) del sito visitato sono direttamente accessibili a qualunque utente, e senza un sistema di protezione adeguato (come visto al punto 2.2), chiunque potrebbe modificare tali file e, in caso di richiesta, fornire contenuti fraudolenti.

-Recupero delle risorse:

Al momento della creazione di un nuovo sito web, una nuova chiave viene generata e calcolata come visto in precedenza sulla base dei contenuti del sito.
Poiché ZeroNet è un sistema di tipo "content-based", un peer che intenda recuperare un sito web deve conoscere tale chiave per richiedere i contenuti del sito.
Ciò implica che ad ogni aggiornamento dei contenuti del sito web tale chiave venga ricalcolata e cambiata (per prevenire modifiche fraudolente da parte dei peer), e che sia quindi necessario comunicare ai vari utenti che desiderano visitare un sito la chiave aggiornata.
Per quanto riguarda in nostro sistema, come analizzeremo in seguito, questa è una delle problematiche cruciali: per fornire un sistema user-friendly, infatti, è necessario che tutte queste dinamiche siano trasparenti e che inoltre, per l'utente medio esterno, le modalità di accesso al portale web distribuito siano le medesime per un qualsiasi altro portale web, ovvero raggiungibile tramite un URL all'interno di un browser.
Per la risoluzione di tale problema, come vedremo, sarà necessario implementare un server proxy, che si occupi di aggiornamento in aggiornamento, di mantenere attiva la funzionalità del sistema ottenendo automaticamente la chiave più aggiornata del portale web.

2.6 ZEROHELLO

Ciò che rende “applicazione” oltre che a semplice protocollo ZeroNet è la web application ZeroHello che viene offerta all’atto dell’installazione.
Essa viene eseguita in una sandbox che ospita un web server ed è raggiungibile via browser all’indirizzo locale attraverso la porta 27001. Tale applicazione offre servizi di IM, forum, posta elettronica, ecc. che si basano sulla peer-to-peer-ness del protocollo sottostante.
I vari servizi offerti da questa applicazione che richiedono una forma di accounting basano il sistema di utenze sull’uso dei wallet attraverso la blockchain bitcoin: ogni nuovo utente (che si può creare direttamente dall’applicazione) richiede l’inserimento semplicemente dell’user id: verranno generate delle chiavi (pubblica e privata) senza le quali nessuno potrà autenticarsi come utente.

Per quanto riguarda l’obbiettivo di questa tesi, comunque, ignoreremo tale applicazione, sfruttando direttamente le funzionalità offerte dal protocollo sottostante. [5]

2.7 CONSIDERAZIONI FINALI

Infine, è importante sottolineare che ZeroNet, sebbene open-source, non nasce direttamente per offrire opportunità di sviluppo di sistemi più articolati come nel nostro caso, ma per tentare con l’insieme di protocollo e web application di rivoluzionare il web per come lo conosciamo, decentralizzando il web in toto.
É evidente che la strada che sta intraprendendo il web per come lo conosciamo oggi è totalmente opposta a questa filosofia, ma nonostante ciò le libertà d’uso che fornisce tale sistema ci permettono comunque di raggiungere il nostro scopo.
La chiusura definitiva di questo progetto, al di là di eventuali falle di sicurezza che ne potrebbero derivare in seguito all’emersione di nuove vulnerabilità, non comprometterebbero il corretto funzionamento del sistema, in quanto non esiste alcun “nodo” centrale della rete che decreterebbe la fine del funzionamento del sistema.
L’unico problema che potrebbe emergere è relativo a quanto visto al paragrafo 2.4, ovvero la perdita di alcuni tracker specifici di ZeroNet: ciò, in ogni caso, non comporterebbe malfunzionamenti persistenti in quanto i vari peer possono recuperare le informazioni da tracker torrent e tramite il protocollo PEX.

CAPITOLO 3

Il terzo e ultimo capitolo è volto ad analizzare le operazioni implementative effettuate per rendere operativo il portale web basato su meccanismi peer to peer utilizzando il sistema Zeronet.

La tesi implementata dal collega Martini usava come gestore delle risorse, ovvero file e cartelle della web application distribuita, il protocollo IPFS, e validava le modifiche apportate da un peer (mettendo eventualmente ai voti) tramite smart contract realizzati dal sistema Ethereum, e unendo le due tecnologie tramite una libreria (Web3.js) che implementava i servizi necessari di Ethereum direttamente nel browser web dal quale si effettuavano le modifiche.

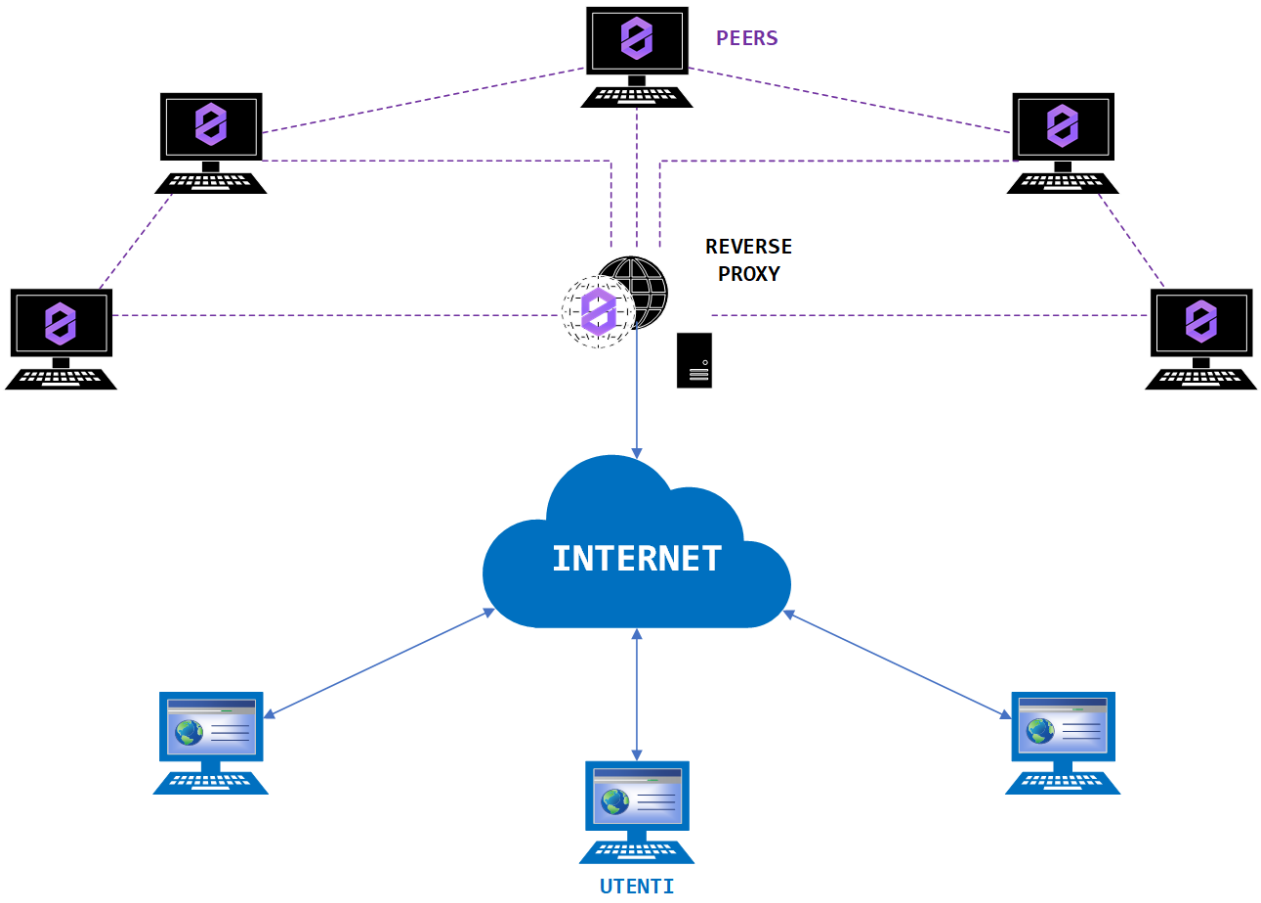
Nella soluzione che riporto in questa tesi, il focus sarà incentrato sull'uso come sistema distribuito di Zeronet, anziché IPFS, sfruttandone le peculiarità protocollari viste nel capitolo 2.

3.1 DESCRIZIONE DELLO SCHEMA RISOLUTIVO PROPOSTO

A differenza della soluzione del collega MARTINI, il quale sfruttava proxy pubblici quali quelli offerti da ISP (Cloudfare nello specifico), nella nostra soluzione si mostra necessario agire autonomamente fornendosi di un server (eventualmente virtuale) comune al network di peer che intendono partecipare al progetto. Questo, oltre che essere dovuto alla necessità di disporre di un grado di versatilità molto elevato, come evidenziato nel paragrafo 3.3, mira anche ad una prospettiva di sviluppo per la quale il network richiederà una certa autonomia.
È necessario inoltre disporre di un nome di dominio che traduca un hostname in un indirizzo IP per raggiungere il proxy dall'esterno e quindi il portale web.
Questa premessa è doverosa, in quanto è necessario sottolineare fin da subito che sebbene tali risorse richiedano uno sforzo economico da parte della rete, tale sforzo è notevolmente ridotto rispetto all'architettura standard per la quale tali risorse sarebbero necessarie per ogni singola entità all'interno della rete (si pensi ad un insieme di realtà aziendali). Inoltre, al crescere del volume della rete di pari, i costi andando divisi per ogni entità andranno decrescendo, senza impattare sulle prestazioni complessive, ma anzi aumentando la fault tolerance generale del portale web.

Per la realizzazione del progetto è necessario affrontare una prima parte di corretta configurazione del sistema Zeronet, partendo dai vari peer fino ad arrivare alla corretta configurazione del reverse proxy nel server. In seguito, sarà necessario intervenire con delle soluzioni software necessarie per garantire la continuità del servizio in caso di modifica dei contenuti web del sito da parte di uno dei peer.

Uno schema grafico riassuntivo di quanto appena descritto è il seguente:



3.2 CONFIGURAZIONE DI ZERONET NEI PEER

La prima operazione da svolgere è sicuramente quella di installare Zeronet nei vari peer che faranno parte della rete.
Supponendo, per semplicità, di disporre in ogni macchina di Linux, sarà sufficiente utilizzare come visto al paragrafo 2.1 il gestore di pacchetti Snappy. Una volta conclusa la fase di installazione è possibile avviare, senza alcun particolare problema zeronet da CLI tramite il comando:
`[root@vps1 ~]# zeronet`

È inoltre possibile evitare che all'avvio Zeronet ci fornisca i parametri di inizializzazione del servizio, aggiungendo il parametro `--silent`, ed inoltre è possibile "daemonizzare" il processo per poter usare la stessa finestra del terminale per altri scopi o chiuderla senza interferire con il processo di Zeronet, tramite il comando:
`[root@vps1 ~]# zeronet --silent &`.

È importante sottolineare che se avviato senza ulteriori parametri, il web server di Zeronet si metterà in ascolto sulla porta 43110, ma è possibile sceglierne un'altra tramite il parametro `--ui_port <#0-65535>`.

Infine, è possibile ed altamente raccomandato, anche se non obbligatorio, permettere l'accesso dall'esterno della rete alla rete locale dei vari peer attraverso una porta (37002 di default), per permettere ai tracker ed agli altri peer di connettersi direttamente all'host in questione, senza

dover passare per un meccanismo di negoziazione delle connessioni, accelerando il processo di condivisione delle risorse tra i peer che usano Zeronet.
Il sistema Zeronet tenterà all'avvio dell'applicazione di aprire tale porta tramite protocollo uPnP, se il router della rete locale ne fornisce il supporto. [6]

3.3 CONFIGURAZIONE DI UN REVERSE PROXY

Come anticipato, nella strategia risolutiva individuata è necessario provvedere all'installazione di un proxy HTTP, facente parte della rete peer-to-peer (sulla quale è ovviamente necessario installare Zeronet), che transiti le richieste che arrivano dagli utenti esterni alla rete peer-to-peer alla rete stessa. L'utilizzo di un server dedicato è necessario principalmente per due motivi:

- Il proxy non dovrà semplicemente inoltrare le richieste da un nome di dominio in ascolto sulle porte 80/443 a Zeronet (in ascolto nella porta assegnata), poiché in tal caso Zeronet risponderebbe con la web application ZeroHello, e da lì manualmente l'utente dovrebbe raggiungere il sito web desiderato, comportamento decisamente poco user-friendly. Deve essere invece configurato in modo tale che le richieste da un nome di dominio specifico vengano trasformate in richieste sul *file index.html* della cartella che ha per nome la chiave pubblica del sito, in modo tale da risultare totalmente trasparente per l'utente finale.
  È evidente che ad ogni modifica dei contenuti web del sito, e quindi ad ogni riesecuzione del processo di digital signature dei contenuti, da cui consegue la creazione di una nuova coppia di chiavi pubblica e privata, sarà necessario aggiornare il proxy con la nuova configurazione. Questo sarà affrontato nel prossimo paragrafo.
- L'utilizzo di una macchina dedicata ci consente, come vedremo nel capitolo 3.5 di permettere ai vari peer di rilevare eventuali aggiornamenti al portale web da parte di un altro peer, e di automatizzare quindi un processo di "*Zeronet visiting*", per mantenere stabilmente diverse copie del portale web aggiornato fra tutti i componenti della rete.

Si evince come sia fine la grana di versatilità necessaria per raggiungere i due obbiettivi di cui sopra, che ci impone di disporre di un server comune alla rete di peer.

La soluzione più efficace per l'implementazione di un reverse proxy è quella di utilizzare uno tra i web server utilizzabili con licenze d'uso gratuite e libere.
Nella soluzione qui presentata, ipotizzando di utilizzare una macchina con sistema operativo CentOS, la scelta implementativa è ricaduta nel server web di Apache (chiamato in ambiente RHEL "*httpd*").
Per configurare un virtual host che inoltri le richieste ad una porta specifica, è necessario modificare il file
`/etc/httpd/conf/httpd.conf`
inserendo all'interno il blocco relativo al virtual host in questione con la seguente configurazione:

```
481 <VirtualHost *:80>
482     ServerName zeronet.mircocazzaro.com
483     ProxyPreserveHost On
484     ProxyPass /<cartella_sito>/ http://127.0.0.1:43110/
485     ProxyPassReverse /<cartella_sito>/ http://127.0.0.1:43110/
486     ProxyRequests Off
487     RewriteEngine on
488     RewriteCond %{HTTP:UPGRADE} ^WebSocket$ [NC]
489     RewriteCond %{HTTP:CONNECTION} Upgrade$ [NC]
490     RewriteRule .* ws://localhost:43110%{REQUEST_URI} [P]
491 </VirtualHost>
```

In questo modo tentando di raggiungere dall'esterno il nome di dominio `zeronet.mircocazzaro.com` (il quale record A dovrà necessariamente puntare all'indirizzo IP del proxy) si riceverà come risposta la home page del sito.

I parametri di avvio di Zeronet nel server proxy differiscono leggermente dai parametri utilizzati tra i vari peer della rete: infatti, per poter eventualmente disporre di un certificato SSL (requisito oramai fondamentale per molti utenti) è necessario inserire nella configurazione, come si evince sopra, il parametro `ProxyPreserveHost On`, il quale transita il nome host con il quale si ha raggiunto il proxy all'applicazione di destinazione (Zeronet). Zeronet però di default non riconosce il nome host se non viene specificato, non mostrando i contenuti e segnalando via CLI l'errore. Nel proxy, dunque, è necessario avviare il processo aggiungendo il parametro, nel caso specifico, `--ui_host zeronet.mircocazzaro.com`.

3.4 AUTO-RENEW PROXY CONFIGURATION

La configurazione del web server come reverse proxy contiene il nome della cartella, corrispondente alla chiave pubblica del sito, sul quale deve essere inoltrata la richiesta.
È necessario, però, in caso di aggiornamento dei contenuti del sito web, riaggiornare le chiavi tramite una nuova digital signature.
Tale problematica impone ad un peer che intende modificare i contenuti e che riesegue il comando `zeronet siteSign`, di comunicare al proxy tale modifica, in modo che possa inoltrare correttamente future richieste.

Una soluzione potrebbe essere quella di far eseguire uno script al peer che effettua la modifica che si colleghi direttamente al server proxy tramite SSH, per poi andare automaticamente a modificare il file di configurazione del web server; tuttavia, tale soluzione comporta due problematiche:

- Sebbene la comunicazione tramite Secure Shell sia protetta, non è di certo una buona norma di sicurezza permettere ad un host di modificare deliberatamente il contenuto di un file di configurazione del web server che se alterato potrebbe portare ad un non funzionamento del sistema o perfino ad un attacco al server;
- Scrivendo direttamente sul file di configurazione si perde l'opportunità, come verrà descritto al paragrafo 3.5, di utilizzare l'informazione appena ottenuta per informare gli altri peer dell'aggiornamento, informazione necessaria per implementare un meccanismo di *auto-visiting* volto a garantire la continuità del servizio del portale web.

Una soluzione potrebbe essere quella di permettere, sempre tramite uno script bash, eseguito manualmente dai peer in seguito all'aggiornamento del sito, dove un utente del server proxy, autorizzato esclusivamente ad eseguire tale operazione, si colleghi via SSH al proxy scrivendo in un file di testo la chiave pubblica del sito.
Il server, periodicamente (con periodo relativamente breve) verifica il contenuto del file tramite, ad esempio, un secondo script bash (il codice potrebbe essere eseguito periodicamente inserendo un record all'interno della *cron table*). Dopo aver verificato che il contenuto corrisponda, tramite pattern matching tra stringhe o tramite un'espressione regolare, che si tratta effettivamente di una chiave, la sostituisce con la precedente nel file di configurazione `/etc/httpd/conf/httpd.conf`. Lo script verificherà poi la correttezza del file di configurazione per riavviare infine il servizio relativo al web server.

3.5 AUTO-VISITING TRAMITE ZERONET

Per quanto riguarda la persistenza dei contenuti aggiornati tra tutti i peer, è necessario implementare un sistema di auto-visiting all'interno della rete Zeronet della versione del sito più recente.

Come appena descritto, ad ogni aggiornamento del portale web, il server proxy viene informato della modifica tramite un file di testo.

Tramite tale file è possibile implementare un meccanismo tramite il quale i vari peer della rete possano, periodicamente, andarne a leggere il contenuto e, qualora contenesse una chiave diversa da quella attualmente in uso, si recasse automaticamente via web all'indirizzo `localhost:43110/<new_key>/`: in tal modo i contenuti web del sito verrebbero scaricati e i tracker messi a conoscenza di un nuovo peer in grado di fornire le risorse di quel sito.
Rimanendo nell'ipotesi del paragrafo 3.2, dove tutti i peer dispongono di ambienti Linux, anche tale script può essere configurato periodicamente come *cron job*.

Sarebbe opportuno, infine, schedulare le richieste di lettura del file dal server tra tutti i peer in modo tale che le richieste avvengano in istanti diversi piuttosto che contemporaneamente.

CONCLUSIONI E FUTURI SVILUPPI

La tesi magistrale del collega Martini proponeva una soluzione che analizzava a 360 gradi i requisiti funzionali e non funzionali per la messa in opera di un sistema volto a garantire, oltre ad una continuità di fruizione dei contenuti, anche un solido sistema basato su Ethereum per garantire a tutti i peer che il portale condiviso venga sottoposto a modifiche solo approvate da un contratto smart sottoscritto da tutti.

Nella soluzione proposta, viene analizzato un modello funzionante e realmente implementato di condivisione di contenuti web peer-to-peer che però non fornisce alcun tipo di garanzia né ai peer, né all'utente finale, che i contenuti visitati non siano stati fraudolentemente modificati da un unico peer della rete senza approvazione da parte degli altri nodi.
È lasciata ad ulteriori sviluppi la possibilità di integrare il sistema qui descritto con un sistema distribuito che garantisca l'affidabilità del sistema e che impedisca al singolo di appropriarsi indebitamente di tutto il portale web distribuito.

Inoltre, il sistema di comunicazione tra peer e proxy per la gestione della configurazione e per il sistema di "auto-visiting" lasciano ampio spazio all'implementazione di misure di sicurezza per prevenire attacchi diretti al proxy o per la manomissione del codice degli script visti ai paragrafi 3.4 e 3.5, oppure ad ulteriori sviluppi per trovare nuove strategie implementative per adempiere tali requisiti.

Infine, essendo Zeronet un sistema open-source, lascia spazio ad eventuali ulteriori implementazioni e/o modifiche al codice che potrebbero fornire un miglior supporto a quanto visto al paragrafo 3.4, senza che sia necessaria un'operazione manuale da parte del peer per comunicare informazioni al proxy, ma facendo in modo che tale operazione sia automatizzata.
Un'altra modifica lato front-end invece potrebbe essere l'eliminazione dell'icona sovrapposta ad ogni sito che Zeronet inserisce come collegamento alla web application ZeroHello.

Questi ed altri elementi potrebbero essere ritoccati per fornire la massima esperienza d'uso possibile all'utente finale.

BIBLIOGRAFIA

[1] **A distributed CMS for small enterprises aggregation, D. Martini, 2019.**

[2] **A Secure Multiple Elliptic Curves Digital Signature Algorithm for Blockchain, Wei Bi , Xiaoyun Jia, Maolin Zheng.**

[3] **Understanding Peer Exchange in BitTorrent Systems, D. Wu; P. Dhungel; X. Hei; C. Zhang; K. W. Ross, 2010.**

[4] [5] [6] **https://zeronet.io/docs/, Documentazione ufficiale di ZeroNet, Tamas Kocsis.**