UNIVERSITÀ DEGLI STUDI ROMA TRE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Tesi Di Laurea

Analisi, progettazione e prove sperimentali di un FulgurHub in TypeScript

Laureando

Federico Ginosa
Matricola 457026

Relatore
Alberto Paoluzzi

Correlatore
Federico Spini

Anno Accademico 2017-2018

Ad Ada Lovelace

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

2008 whitepaper Bitcoin Nel 2008 viene pubblicato il whitepaper di Bitcoin, una proposta di soluzione al problema della doppia spesa basata sull'utilizzo di una rete peer-to-peer [9].

2009 pubblicazione protocollo Il 2009 vede la nascita della prima cryptovaluta, bitcoin, con la pubblicazione dell'implementazione del protocollo Bitcoin. Gli standard variano, ma sembra essersi formato un consenso nel riferirsi con Bitcoin maiuscolo al protocollo e con bitcoin minuscolo alla moneta in se [12].

Bitcoin 10 anni dopo Negli anni successivi decine di protocolli alternativi a Bitcoin sono fioriti. Le cryptovalute da argomento di nicchia hanno visto una costante crescita di adozione. Dal 2009 fino a oggi il numero di transazioni quotidiane è cresciuto più che linearmente, raggiungendo il suo attuale picco storico nel dicembre del 2017, con più di 450K transazioni in un giorno [13].

Limiti architetturali Questo crescente interesse nei confronti delle cryptovalute si scontra però con i limiti architetturali relativi alla scalabilità della blockchain. Con gli attuali parametri di blocksize e blockinterval (1 megabyte

e 10 minuti), il throughput massimo è compreso tra le 3 e le 7 tps (transazioni per secondo). Portando questi due parametri all'estremo, 1 megabyte e 1 minuto, si riuscirebbe a decuplicare il throughput, senza sacrificare il protocollo in termini di sicurezza [4]. Sebbene 30 - 70 tps rappresenterebbero un fondamentale miglioramento tecnologico di Bitcoin, il throughput raggiunto non sarebbe comunque confrontabile con quello di sistemi centralizzati analoghi come il circuito VISA, con le sue 56K tps [14].

Soluzioni Diverse sono le soluzioni proposte per risolvere i problemi di scalabilità della blockchain e possono essere suddivise in tre categorie:

- **Algoritmo di consenso** Alla base del whitepaper di Satoshi Nakamoto c'è il Proof Of Work. Modificando il meccanismo alla base della ricerca del consenso è possibile migliorare la scalabilità della blockchain. Ad oggi diverse sono le alternative proposte [6], [1], [7].
- **Sharding** Questo concetto non è nuovo nel mondo dei database. L'idea è quella di suddividere la blockchain in più parti. La ricerca del consenso avviene in ciascuno di queste parti. Anche da questo punto di vista diversi sono i lavori e le proposte [8], [15].
- **Off-chain** Layers è un famoso pattern architetturale. Un forte impiego di questo pattern è stato fatto nell'ambito del networking, vedi pila ISO/OSI. La scalabilità off-chain si realizza costruendo un secondo layer sopra alla blockchain, che permetta di ereditare le sue caratteristiche (sicurezza e distribuzione), aggiungendone delle altre, come la scalabilità.

Questi tre diversi approcci alla scalabilità della blockchain non sono in contrasto l'uno con l'altro, ma anzi possono essere applicati assieme in maniera sinergica. Nel lavoro di questa tesi ho approfondito l'ultima categoria, la scalabilità off-chain. In particolare mi sono occupato delle seguenti attività:

- Analisi dello stato dell'arte relativa a soluzioni di scalabilità off-chain
- Realizzazione di un canale di pagamento inestinguibile (IPC)

- Analisi, progettazione e sviluppo di FulgurHub
- Prove sperimentali di FulgurHub

In questa tesi il capitolo due tratta il background necessario, in particolare si approfondisce il design di un payment channel e si introduce l'architettura di FulgurHub. Nel terzo capitolo si effettua l'analisi nel dettaglio di FulgurHub. Nel quarto capitolo si descrivono le fasi di progettazione e sviluppo di FulgurHub. Nel quinto capitolo si mostrano le prove sperimentali relative a quanto è stato implementato e si discutono i risultati in termini di performance e scalabilità.

Capitolo 2

Background

In questo capitolo si discute il background necessario. In particolare in sezione 2.1 si discute la blockchain: come funziona, i suoi casi d'uso e i suoi limiti. In sezione 2.2 si descrivono gli state channel, facendo un affondo sui payment channel e inextinguishable payment channel. Infine in sezione 2.3 si introduce il protocollo FulgurHub: le sue motivazione, le caratteristiche e i lavori correlati.

2.1 Blockchain

Il problema La blockchain risolve il problema del double spending in un sistema peer-to-peer completamente decentralizzato [9]. Questo permette di memorizzare in maniera immutabile dei pagamenti in un registro pubblico, avendo la certezza che nessuno possa spendere più volte lo stesso token.

Il caso d'uso Il caso d'uso tipico della blockchain è l'invio e la ricezione di pagamenti. La transazione rappresenta un pagamento. Essa può essere immaginata come un arco che unisce due nodi. Il nodo iniziale rappresenta il

pagante, il nodo finale il pagato. Tutte queste transazioni vengono memorizzate su un registro pubblico detto ledger. Sebbene le transazioni siano pubbliche, esse vengono descritte anche come pseudoanonime, in quanto pubblico è solo l'indirizzo di paganti e pagati ma non il loro nominativo.

Cos'è la blockchain La blockchain è una lista concatenata di blocchi. Ciascun blocco contiene: l'hash del precedente blocco, il merkle root relativo alla lista di transazioni associate al blocco corrente e un nonce. In Bitcoin un nuovo blocco viene aggiunto ogni dieci minuti e il merkle root rappresenta una prova succinta di una lista di transazioni di dimensione minore o uguale a 1 megabyte.

Come funziona la PoW I blocchi vengono aggiunti dai miner. I miner sono dei nodi della rete che si occupano di trovare un nonce cha faccia si che l'hash del blocco corrente abbia un numero di zeri iniziali pari a D. Questo valore D rappresenta la difficoltà corrente di mining della rete. La difficoltà è autoregolata dal protocollo e aumenta o diminuisce a seconda del tempo necessario per minare i precedenti blocchi. Un miner che riesce a presentare un nonce e un blocco valido ottiene in cambio le fee delle singole transazioni e una coinbase.

Cos'è uno smart contract Inviare un pagamento in Bitcoin significa sbloccare uno o più UTXO (Unspent Transaction Output). Sbloccare un UTXO significa presentare una prova crittografica della proprietà di un certo token. La verifica della prova crittografica viene effettuata da tutti i nodi della rete eseguendo un ASFND (automa a stati finiti non deterministico). Il protocollo Bitcoin permette di implementare e deployare sulla rete degli automi anche più complessi. Script è il linguaggio di programmazione stack-based non Turing-completo che permette di descrivere questi automi in Bitcoin. Quando la complessità degli automi aumenta, si parla di smart contract, ovvero di contratti che permettono lo sblocco di fondi previa verifica di un insieme complesso di regole.

Smart contract Turing-completi Sebbene abbia senso parlare di smart contract in Bitcoin, l'uso del termine in questo contesto è stato introdotto solo nel 2014, con la pubblicazione del whitepaper di Ethereum [3]. Ethereum è un protocollo che eredita gran parte delle caratteristiche di Bitcoin e in più introduce la EVM (Ethereum Virtual Machine) la macchina virtuale che esegue gli smart contract. Gli smart contract in Ethereum vengono descritti in Solidity, un linguaggio di programmazione C-like Turing-completo. La turing completezza permette di descrivere un più ampio spettro di regole.

Scalabilità off-chain Nel Capitolo 1 sono stati introdotti i limiti architetturali della blockchain e le tre categorie di approcci risolutivi. La scalabilità off-chain è una delle tipologie di soluzioni atta a superare i limiti di scalabilità della blockchain. Questo approccio riduce sensibilmente le interazioni necessarie sulla blockchain, spostandole fuori di essa, senza compromettere le proprietà di sicurezza.

2.2 State channel

Gli state channel permettono a due parti di modificare in maniera sicura porzioni della blockchain, dette depositi di stato. Questi depositi di stato sono memorizzati all'interno di indirizzi multisignature. Le parti modificano lo stato dello state channel scambiando messaggi off-chain. Questi messaggi descrivono un aggiornamento dello stato, per esempio la prossima mossa in una partita di tris [16].

2.2.1 Payment channel

Un payment channel è una particolare tipologia di state channel. I messaggi scambiati off-chain rappresentano dei pagamenti, ovvero l'aggiornamento

del bilancio delle parti. Instaurare un payment channel richiede una sola operazione on-chain da ciascuna parte. L'operazione on-chain viene eseguita su uno smart contract dedicato al singolo payment channel. Questa unica operazione on-chain abilita un numero illimitato di pagamenti off-chain, nei limiti del balance iniziale delle parti. I messaggi off-chain possono essere scambiati mediante qualunque mezzo, comunemente una connessione http. Un payment channel permette dunque di spostare i problemi di scalabilità dalla blockchain a un server http, ma la letteratura riguardo a come far scalare quest'ultimo è consolidata.

Architettura Un payment channel permette di effettuare un numero illimitato di transazioni off-chain tra due parti. Ciascuna parte deve mettere a disposizione un server http che permetta l'invio e la ricezione di pagamenti. Una delle due parti deploya lo smart contract associato e apre il canale. In un secondo momento la controparte effettua il join del canale, stabilendone la definitiva apertura. In questa progettazione si è presa come riferimento la blockchain di Ethereum.

Deploy Il deploy è la prima fase di inizializzazione. Alice deploya lo smart contract del relativo canale. L'operazione di deployment è richiesta per ciascun singolo payment channel. Questa fase permette di ottenere l'indirizzo di un contratto, che nelle successive fasi verrà adottato per richiamare le operazioni on-chain che si intende richiamare. In questa fase lo stato del payment channel è detta $INIT$.

Figura 1: Deploy on-chain dello smart contract di un payment channel.

Apertura Alice apre il canale e blocca un quantitativo arbitrario di fondi

all'interno dello smart contract. Questi fondi rappresentano il bilancio iniziale di Alice. Si fa notare come la fase di deploy e di apertura possano essere svolte con un'unica operazione on-chain. Oltre a depositare i fondi, Alice con questa operazione porta in catena il suo indirizzo ip e l'indirizzo ethereum di Berto. Terminata la procedura lo stato del canale diventa $OPENED$.

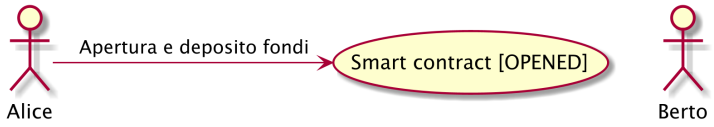

Figura 2: Apertura e deposito fondi on-chain in un payment channel.

Join In un secondo momento Berto effettua il join del canale di pagamento aperto da Alice. Anche questa operazione viene effettuata on-chain. Berto deposita i fondi che corrisponderanno al suo bilancio iniziale e porta in catena il proprio indirizzo ip. Con questa operazione il canale è definitivamente stabilito e lo stato passa da $OPENED$ a $ESTABLISHED$.

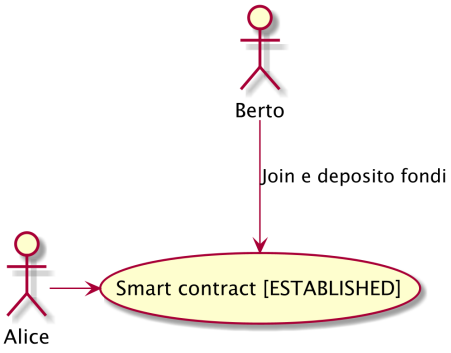

Figura 3: Join e deposito fondi on-chain in un payment channel.

Schema propose/accept I pagamenti off-chain avvengono mediante lo schema propose/accept. Alice (o Berto) propone un aggiornamento dello stato del canale firmando questa proposta con la propria chiave privata. Berto riceve la proposta, ne verifica la validità ed eventualmente l'accetta inviando la

proposta controfirmata ad Alice. Il pagamento è avvenuto, senza la necessità di ulteriori tempi di attesa o conferme.

Gli endpoint pubblici Ciascuna controparte di un payment channel mette a disposizione un server http. Gli endpoint pubblici sono detti /propose e /accept. L'endpoint /propose permette di ricevere una proposta di aggiornamento di bilancio. L'endpoint /accept permette di ricevere una proposta precedentemente inviata, controfirmata in Tabella 1.

Tabella 1: Struttura di una propose

Campo	Descrizione
seq	Il numero di sequenza
balance_a	Il balance di chi ha aperto il canale
balance_b	Il balance di chi ha effettuato il join del canale
sign	La firma della propose

Richiesta di chiusura Chiudere un canale significa aggiornare il balance on-chain delle parti in modo tale che corrisponda a quello dell'ultima propose comunemente accordata. La prima fase di questo processo è detta richiesta di chiusura. In particolare si porta in catena l'ultima propose comunemente firmata. In questo modo lo stato del canale passa da $ESTABLISHED$ a $CLOSED$. La richiesta di chiusura può essere effettuata da Alice o da Berto.

Finalizzazione della chiusura L'operazione di finalizzazione della chiusura viene effettuata da tutte e due le parti. Essa corrisponde al ritiro on-chain dei rispettivi fondi. Questa operazione può essere effettuata solo quando è passato un certo tempo dalla richiesta di chiusura. Il tempo che occorre attendere per finalizzare la chiusura è detto grace period.

Discutere una propose Alice (o Berto) potrebbe non comportarsi corret-

tamente, portando in chiusura una propose diversa dalla più recente. In questo caso Berto può discutere la propose durante il grace period. Discutere una propose significa portare in catena una propose firmata da Alice con numero di sequenza maggiore rispetto a quella presentata. Nel caso in cui la discussione abbia successo, Alice viene punita; la punizione consiste nel trasferimento di tutti i suoi fondi a Berto.

Il problema della free-option Quando Alice invia una propose a Berto senza ricevere la controfirma, Berto ha il vantaggio di poter scegliere di chiudere il canale con due propose, la penultima o l'ultima. Inviare una propose coincide con inviare un pagamento, quindi sebbene Berto possa decidere di presentare in catena la penultima propose, questa descriverà uno stato per lui più svantaggioso.

2.2.2 Inextinguishable payment channel

I payment channel permettono di trasferire un volume di coin limitato. Il valore trasferibile è fissato alla somma del balance di Alice e di Berto. Spesso questi canali sono sbilanciati, ovvero una delle due controparti effettua più pagamenti dell'altra. Un canale sbilanciato nel tempo prosciuga il balance di una delle due parti, rendendo il payment channel inutilizzabile. L'unica soluzione consiste nel chiudere il payment channel corrente e aprirne un nuovo, caricando nuovi fondi. Questa soluzione richiede delle operazioni on-chain onerose (deploy, apertura e join). Gli IPC (inextinguishable payment channel) superano questo problema, proponendo dei canali di pagamento che permettono di caricare e scaricare a caldo i balance.

Schema detach/attach Questo schema rappresenta un'estensione dello schema propose/accept. Esso permette di staccare un token off-chain e di attaccarlo on-chain. Un token rappresenta un certo quantitativo del bilancio. La struttura di un token è illustrato in Tabella 2.

Tabella 2: Struttura di un token

Campo	Descrizione
seq	Numero di sequenza del token
value	Valore del token
sign	Firma del token

Anche la struttura dati relativa a una propose viene estesa. I campi aggiunti sono illustrati in Tabella 3.

Tabella 3: Campi propose aggiuntivi in un IPC

Campo	Descrizione
hash token	L'hash relativo al token
type of propose	attach/detach

Ritiro a caldo Alice vuole ritirare a caldo 0.5 eth; effettua il detach off-chain di un token; invia a Berto una propose contenente un token di 0.5 eth che scala da $\text{balance}_{\text{a}}$. Berto risponde con propose e token firmati. Il token firmato rappresenta la PoD (Proof of Detachment). Alice effettua l'attach in catena della PoD e ritira a caldo 0.5 eth.

Ricarica a caldo Alice vuole ricaricare a caldo il canale di 0.5 eth; effettua l'attacch on-chain di un token depositando nello smart contract 0.5 eth. Questa operazione on-chain viene notificata a Berto; tale notifica rappresenta la PoA (Proof of Attachment). A questo punto Alice invia a Berto una propose in cui effettua l'attach di un token di pari valore e incrementa di 0.5 eth il proprio balance. Berto risponde con la propose firmata, confermando la ricarica a caldo.

Double spending di un token Quando Alice ritira a caldo presentando un token, lo smart contract associa una PoA (Proof of Attachment) relativa al numero di sequenza del token corrente. Questo permette allo smart contract di non accettare token già spesi.

2.3 Fulgur Hub

2.3.1 Motivazioni

Sebbene i canali di pagamento e gli IPC rappresentino un punto di svolta per la scalabilità off-chain, essi sono degli strumenti rudimentali e con una esperienza utente limitata. Non è infatti pensabile dover inizializzare un canale di pagamento con ciascun individuo con cui si voglia instaurare un rapporto economico. Fulgur Hub nasce dalla necessità di migliorare l'esperienza utente degli IPC e di potenziare alcune delle loro caratteristiche.

2.3.2 Caratteristiche

Transazioni istantanee ed economiche In Bitcoin la conferma di una transazione richiede 60 minuti. In un IPC basta lo scambio di due messaggi su protocollo http per effettuare e confermare un pagamento. Questo apre diverse e interessanti prospettive economiche, ad esempio una macchina in cloud potrebbe essere pagata dopo ogni secondo di utilizzo o potremmo vedere il nostro stipendio accreditato dopo ogni minuto di lavoro; FulgurHub abilita questi casi d'uso.

Transazioni tra più di due entità In un IPC i pagamenti possono essere effettuati tra due partecipanti. FulgurHub consente di effettuare pagamenti tra gli N utenti registrati ad un FulgurHub.

Pagamenti ibridi FulgurHub permette di effettuare dei pagamenti ibridi. Ciascun utente infatti possiede due balance, uno on-chain e uno off-chain e può decidere di spostare dei fondi da uno stato off-chain a uno stato on-chain e viceversa. Inoltre abilita i pagamenti tra utenti di due FulgurHub diversi.

Autogestito In un IPC l'utente deve costantemente verificare e accettare la validità di un pagamento, oltre a contestare eventuali comportamentei scorretti della controparte. In FulgurHub i server degli utenti e dell'hub si occupano di gestire autonomamente diversi scenari, limitando allo stretto necessario l'intervento manuale.

Pagamenti trustless Caratteristica essenziale è che un utente onesto abbia la certezza di non perdere i propri fondi. In sistemi centralizzati questa garanzia esiste perché ci si fida di un'entità centrale, come una banca o un servizio di e-payment. In un FulgurHub questa garanzia è data dal protocollo stesso, in questo senso i pagamenti sono trustless.

Passività e anonimato FulgurHub è un sistema passivo; questo significa che l'hub non contatta mai gli utenti, ma solo quest'ultimi contattano l'hub. Questo permette agli utenti di non dover fornire il loro indirizzo ip reale e quindi di poter effettuare pagamenti anche dietro una rete come Tor.

2.3.3 Lavori correlati

Tumblebit Si tratta di un hub di pagamenti anonimo basato su Bitcoin. L'approccio di centralizzazione garantisce anonimato e pagamenti trustless. Sfortunatamente il particolare payment channel adottato è unidirezionale e ha un tempo di vita limitato [5].

CoinBlesk Un bitcoin wallet che usa un server centrale che permette di eseguire dei pagamenti virtuali. Supporta micropagamenti istantanei, ma l'approccio non è considerabile trustless [2].

Lightning e Raiden Network Entrambi i network si basano su un grafo di payment channel bidirezionali. Un pagamento avviene in maniera analoga all'instradamento di un pacchetto su internet. Una volta trovato il percorso ottimo esso deve essere completato con successo in ciascun hop intermedio. Se un solo hop fallisce il pagamento fallisce. Questo garantisce l'atomicità dei pagamenti [10] [17]. Sebbene Lightning Network e Raiden Network siano progettati per essere decentralizzati, la realtà economica fa tendere la topologia di rete alla centralizzazione; maggiore è il numero di hop, maggiori sono le commissioni e le probabilità di insuccesso. FulgurHub è stato disegnato con questo in mente e propone una topologia hub and spoke.

Capitolo 3

Analisi

In questo capitolo si descrive il processo di analisi svolto. In particolare in sezione 3.1 si discutono gli obiettivi dell'analisi. In sezione 3.2 si descrive l'architettura generale di FulgurHub. Infine in sezione 3.3 si descrivono i principali casi d'uso e la gestione di eventuali eccezioni.

3.1 Obiettivi

Dimostrazione di fattibilità Un obiettivo di questa tesi è stato dimostrare la fattibilità del protocollo FulgurHub. In particolare ci si è concentrati sulle feature principali: apertura di un canale, pagamenti OnChain-OnChain, pagamenti OffChain-OffChain, pagamenti OffChain-OnChain, pagamenti OnChain-OffChain, prelievi a caldo, ricariche a caldo, chiusura di un canale e riscossione di pending token.

Dimostrare la scalabilità architetturale Come detto in capitolo 2, le motivazioni che hanno mosso la progettazione di FulgurHub riguardano i

limiti architetturali di scalabilità della blockchain. Obiettivo di questa tesi è stato anche dimostrare la scalabilità architetturale di FulgurHub.

3.2 Descrizione generale dell'architettura

Il sistema si basa su uno smart contract. Lo smart contract gestisce il balance on-chain e il payment channel di ciascun utente. In figura 4 si mostra l'architettura hub-and-spoke in cui 4 utenti (Alice, Berto, Cecilia e Dario) operano su FulgurHub [11].

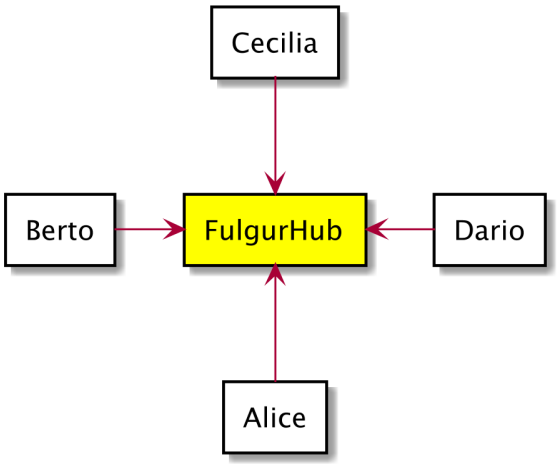

Figura 4: Architettura hub-and-spoke di FulgurHub

Hub L'hub è supportato da un modulo software che interagisce con lo smart contract. Il modulo è stateless, favorendo disponibilità e scalabilità. L'hub è passivo, questo significa che non contatta mai direttamente i client; solo i client possono contattare l'hub. I client possono contattare l'hub mediante richieste http a uno dei suoi endpoint pubblici.

Client La relazione tra client e hub può essere descritta come una "registrazione trustless" del client al servizio di intermediazione offerto dall'hub

[11]. Il client è supportato da un modulo software che interagisce con lo smart contract e l'hub. La registrazione del client coincide con l'instaurare un IPC esteso che permetta dei pagamenti ibridi, come descritto in 3.3. Un client può chiudere la registrazione dall'hub in ogni momento; in particolare deve chiudere la propria registrazione appena si verifica un comportamento anomalo dell'hub.

Smart contract Lo smart contract garantisce la relazione trustless tra i client e l'hub. In particolare lo smart contract deve essere utilizzato quando una delle parti non si comporta correttamente. Oltre a questo lo smart contract si occupa di aprire/chiudere la sottoscrizione del client e di gestire pagamenti ibridi che coinvolgano endpoint on-chain.

3.3 Casi d'uso

Strutture dati e simbolismo FulgurHub si fonda su due tipi di strutture dati, le propose e i token. Una propose ϕ_i descrive il balance off-chain di client (β_i^C) e hub (β_i^H). Le propose sono ordinate totalmente sulla base del numero di sequenza i. Un token τ_j può essere staccato ($\mathbb{D}$) o attaccato ($\mathbb{A}$) ad una propose. Inoltre una propose può essere firmata dal client ($\phi_i^{\sigma_C}$), dall'hub ($\phi_i^{\sigma_H}$) o da entrambi ($\phi_i^{\sigma_C,\sigma_H}$).

$$\phi_i^{\sigma_C,\sigma_H} =< \beta_i^C, \beta_i^H, \tau_j, \mathbb{D}||\mathbb{A} > \tag{1}$$

Un token è identificato in maniera univoca dalla tupla (j, α_P), dove j identifica il numero di sequenza del token e α_P l'indirizzo ethereum del pagato. Il client staccando un token può sottrarre una porzione ν_j del proprio bilancio. Un token può essere staccato dal bilancio on-chain od off-chain. Un token può essere recapitato al pagato. Il pagato per riscuotere un token deve attaccarlo

off-chain (mediante una propose) od on-chain (mediante lo smart contract). Esistono due tipi di token; quelli riscuotibili on-chain ($\mathbb{ON}$) e quelli riscuotibili off-chain ($\mathbb{OFF}$). Inoltre un token può essere firmato dal client (τ^{σ_C}), dall'hub (τ^{σ_H}) o da entrambi (τ^{σ_C,σ_H}). Un token può essere riscosso entro un tempo di scadenza exp.

$$\tau^{\sigma_C,\sigma_H}_{y,ID(P)} =< \nu_y, exp, \mathbb{ON}||\mathbb{OFF} > \tag{2}$$

Una propose $\phi_i^{\sigma_C}$ con un token τ_y detached ($\mathbb{D}$) firmato rappresenta una ricevuta di pagamento. La ricevuta di pagamento è una prova incontrovertibile della riscossione di un token.

Per indicare il balance off-chain di un'entità k ad una propose con numero di sequenza pari a i si usa il simbolo β_i^k, mentre per indicare il balance on-chain $\overline{\beta^k}$.

L'indirizzo ethereum di un'entità k è indicato dal simbolo α_k. L'insieme di indirizzi ethereum che hanno una sottoscrizione attiva con il FulgurHub associato ad H è detto Π^H.

3.3.1 Sottoscrizione di un FulgurHub

Alice vuole sottoscrivere una registrazione su un FulgurHub. Questa attività coincide con l'apertura di un payment channel.

Precondizioni
a) $\{\alpha^A\} \not\subset \Pi^H$
b) L'hub ha deployato lo smart contract
c) Il server dell'hub è in ascolto

Descrizione delle interazioni Un client per sottoscrivere un FulgurHub deve eseguire la funzione subscribe dello smart contract fornendo il proprio indirizzo ethereum α_C, il bilancio iniziale off-chain β_0^C e on-chain $\overline{\beta^C}$. Inoltre il client deve indicare il bilancio iniziale off-chain dell'hub β^H. Una volta eseguita la transazioni on-chain viene recapitata una notifica all'hub $< \beta_0^C, \overline{\beta^C}, \beta^H, \alpha_C >$. In figura 5 viene fornito un diagramma di sequenza del caso d'uso.

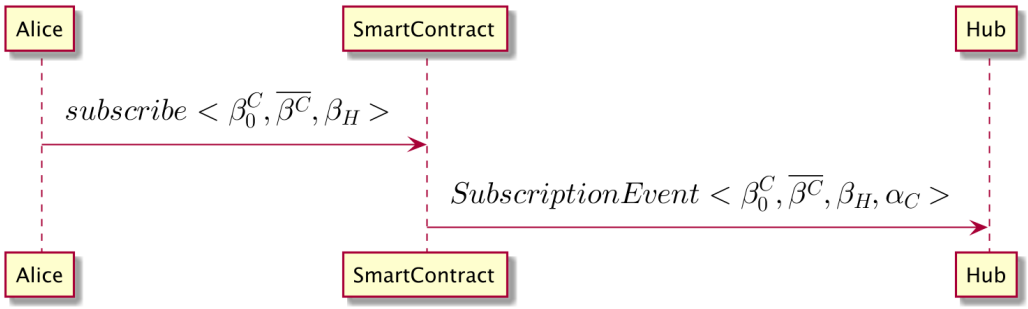

Figura 5: Sottoscrizione di un FulgurHub

3.3.2 Pagamento OnChain-OnChain

Un pagamento OnChain-OnChain sposta ν fondi dal balance on-chain di Alice $\overline{\beta^A}$ al balance on-chain di Berto $\overline{\beta^B}$. Questo pagamento viene totalmente gestito dallo smart contract e non richiede alcuna interazione con i server dei client o dell'hub.

Precondizioni
a) $\{\alpha^A, \alpha^B\} \subseteq \Pi^H$
b) Il balance on-chain di Alice e Berto è rispettivamente pari $\overline{\beta^A}$ e $\overline{\beta^B}$

Descrizione delle interazioni Alice esegue il metodo transfer dello smart contract. L'esecuzione del metodo richiede il quantitativo ν di fondi che si intende spostare e l'indirizzo ethereum α^B di Berto. Terminata l'esecuzione

del metodo lo smart contract aggiorna il balance on-chain di Alice in $\overline{\beta^A} - \nu$ e quello di Berto in $\overline{\beta^B} + \nu$. Un diagramma di sequenza è disponibile in figura 6.

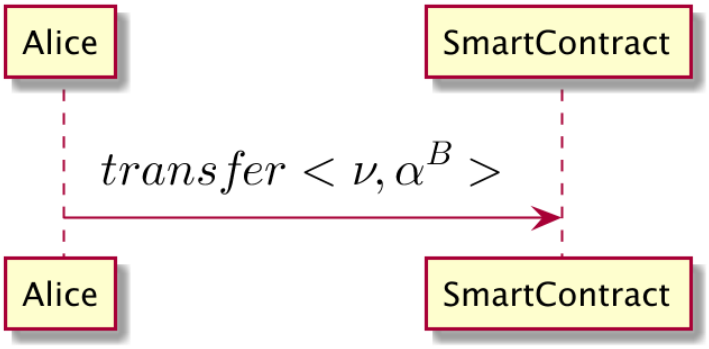

Figura 6: Pagamento OnChain-OnChain in FulgurHub.

3.3.3 Pagamento OffChain-OffChain

Un pagamento OffChain-OffChain sposta fondi dal balance off-chain di Alice β_i^A a quello di Berto β_i^B. Questo tipo di pagamento non richiede interazioni con la catena, il che lo rende economico e istantaneo.

Precondizioni
a) $\{\alpha^A, \alpha^B\} \subseteq \Pi^H$
b) Le ultime propose confermate nei canali di Alice e Berto sono ϕ_i^A e ϕ_j^B.

Descrizione delle interazioni Alice costruisce, firma e invia $\phi_{i+1}^{\sigma_A}$ all'hub. L'hub risponde con la propose $\phi_{i+1}^{\sigma_A,\sigma_H}$ e il token $\tau_{y,\alpha_B}^{\sigma_A,\sigma_H}$ controfirmati.

$$\begin{aligned} \tau_{y,\alpha_B}^{\sigma_A} &=< \nu_y, exp, \mathbb{OFF} > \\ \phi_{i+1}^{\sigma_A} &=< \beta_i^A - \nu_y, \beta_i^H, \tau_{y,\alpha_B}^{\sigma_A}, \mathbb{D} > \end{aligned} \tag{3}$$

$\tau_{y,\alpha_B}^{\sigma_A,\sigma_H}$ rappresenta una PoD (Proof of Detachment). Alice invia la PoD a Berto. Berto costruisce $\phi_{j+1}^{\sigma_B}$ effettuando l'attach della PoD.

$$\phi_{j+1}^{\sigma_B} =< \beta_i^B + \nu_y, \beta_i^H - \nu_y, \tau_{y,\alpha_B}^{\sigma_A}, \mathbb{A} > \tag{4}$$

Berto invia la ricevuta di pagamento $\phi_{j+1}^{\sigma_B}$ ad Alice. Alice ora ha in mano una prova incontrovertibile del fatto che il suo token sia stato riscosso. In questa fase l'hub si è esposto di ν_i fondi sul canale di Berto; Alice deve ribilanciare questa situazione e lo fa costruendo $\phi_{i+2}^{\sigma_A}$, una nuova propose in cui attacca la PoD ricevuta da Berto.

$$\phi_{i+2}^{\sigma_A} =< \beta_i^B + \nu_y, \beta_i^H - \nu_y, \tau_y^{\sigma_B}, \mathbb{A} >_(\ \sigma_B) \tag{5}$$

Il pagamento OffChain-OffChain è considerato concluso. In figura 7 viene fornito uno diagramma di sequenza delle interazioni.

B non invia la ricevuta di pagamento ad A Il collegamento tra Alice e Berto è opzionale. Alice infatti può contattare l'hub e richiedere la ricevuta di pagamento.

L'hub non permette di staccare un token Se l'hub non è collaborativo, Alice chiude il canale.

L'hub non permette di attaccare un token Se l'hub non è collaborativo, Berto ha la facoltà di chiudere il canale e successivamente riscuotere il pending token on-chain.

Mancanza di cooperazione nel ricevere un pagamento Il client può cancellare il pagamento al termine della sua scadenza, ritirandolo off-chain.

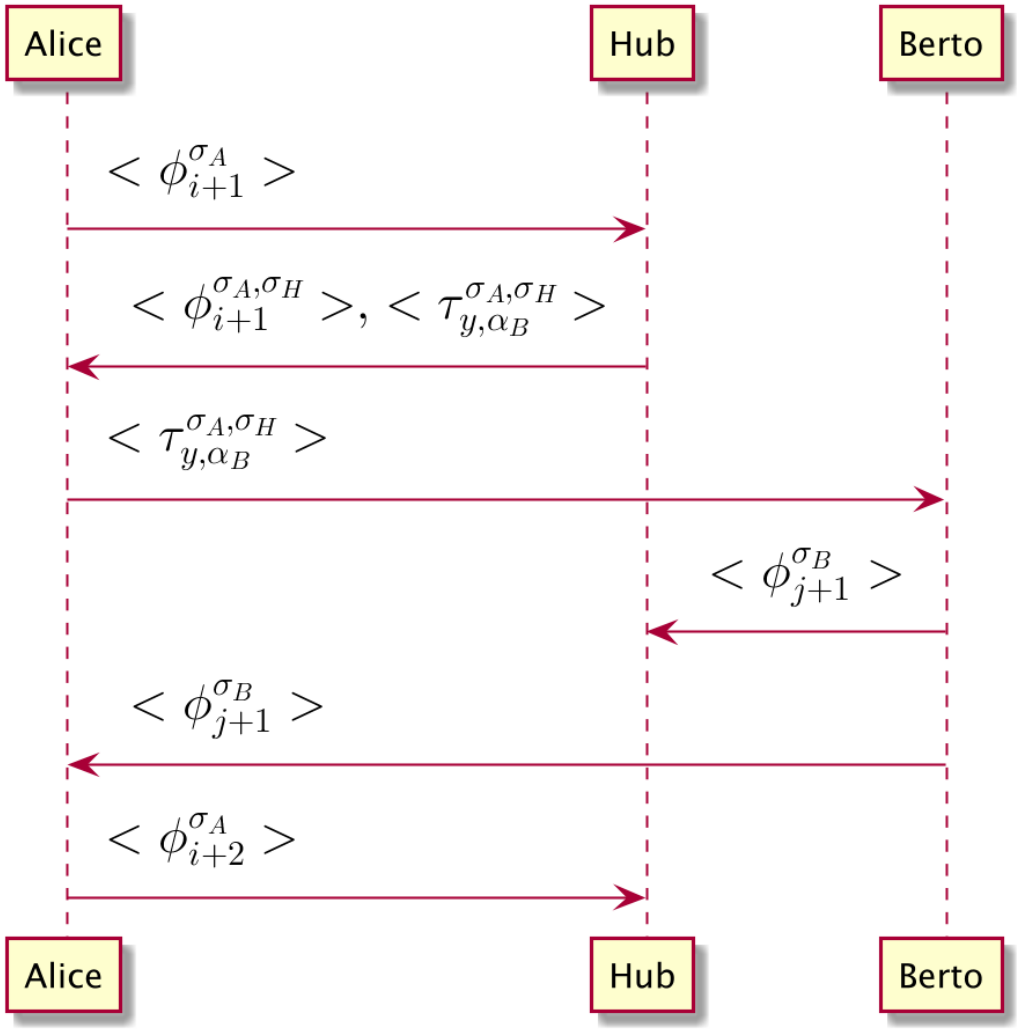

Figura 7: Pagamento OffChain-OffChain in FulgurHub.

3.3.4 Pagamento OffChain-OnChain

Un pagamento OffChain-OnChain consiste nel spostare fondi dal balance off-chain di Alice β_i^A al balance on-chain dxi Berto $\overline{\beta^B}$.

Precondizioni
a) $\{\alpha^A, \alpha^B\} \subseteq \Pi^H$
b) L'ultima propose confermata nel canale di Alice è ϕ_i^A.

Descrizione delle interazioni Alice costruisce, firma e invia $\phi_{i+1}^{\sigma_A}$ all'hub. L'hub risponde con la propose $\phi_{i+1}^{\sigma_A,\sigma_H}$ e il token $\tau_{y,\alpha_B}^{\sigma_A,\sigma_H}$ controfirmati.

$$\begin{aligned} \tau_{y,\alpha_B}^{\sigma_A} &=< \nu_y, exp, \mathbb{ON} > \\ \phi_{i+1}^{\sigma_A} &=< \beta_i^A - \nu_y, \beta_i^H, \tau_{y,\alpha_B}^{\sigma_A}, \mathbb{D} >_(\ \sigma_A) \end{aligned} \tag{6}$$

$\tau_{y,\alpha_B}^{\sigma_A,\sigma_H}$ rappresenta una PoD (Proof of Detachment). Alice invia la PoD a Berto. Berto effettua l'attach on-chain del token mediante la funzione attach dello smart contract. Lo smart contract aggiorna il balance on-chain di Berto in $\overline{\beta^B} + \nu_y$. Il pagamento è considerato concluso. In figura 8 viene fornito uno diagramma di sequenza delle interazioni.

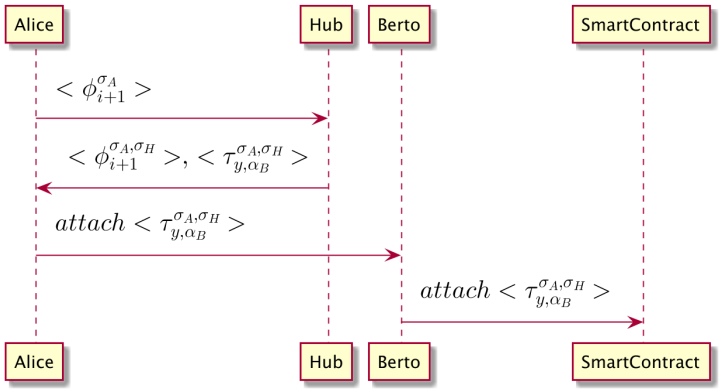

Figura 8: Pagamento OffChain-OnChain in FulgurHub.

3.3.5 Pagamento OnChain-OffChain

Un pagamento OnChain-OffChain consiste nel spostare fondi dal balance on-chain di Alice $\overline{\beta^A}$ al balance off-chain di Berto β_j^B.

Precondizioni
a) $\{\alpha^A, \alpha^B\} \subseteq \Pi^H$
b) L'ultima propose confermata nel canale di Berto è ϕ_j^B
c) Il balance on-chain di Alice è $\overline{\beta_A}$

Descrizione delle interazioni Alice esegue la funzione detach dello smart contract fornendo l'indirizzo di Berto (α_B) e il quantitativo ν che si vuole staccare. Lo smart contract aggiorna il balance on-chain di Alice in $\overline{\beta_A} + \nu$. Terminata l'esecuzione della funzione, lo smart contract invia la relativa PoD a Berto. Berto costruisce, firma e invia $\phi_{j+1}^{\sigma_B}$ all'hub, attaccando la PoD. L'hub risponde con la propose firmata $\phi_{j+1}^{\sigma_B,\sigma_H}$. In figura 9 viene fornito uno diagramma di sequenza delle interazioni.

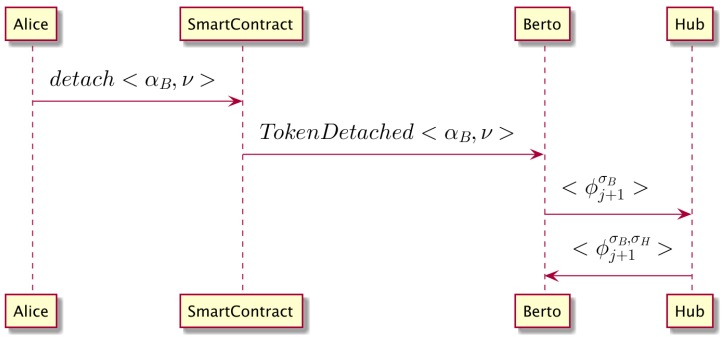

Figura 9: Pagamento OnChain-OffChain in FulgurHub.

$$\begin{aligned} \tau_y^{\sigma_B} &=< \nu_y, \perp, \mathbb{ON} > \\ \phi_{j+1}^{\sigma_B} &=< \beta_j^B - \nu_y, \beta_j^H, \tau_{y,\alpha_B}^{\sigma_B}, \mathbb{A} > \end{aligned} \tag{7}$$

3.3.6 Prelievo a caldo

Effettuare un prelievo a caldo significa spostare dei fondi dal balance off-chain di Alice β_i^A al balance on-chain di Alice $\overline{\beta^A}$.

Precondizioni
a) $\{\alpha^A\} \subseteq \Pi^H$
b) L'ultima propose confermata nel canale di Alice è ϕ_i^A
c) Il balance on-chain di Alice è $\overline{\beta_A}$

Descrizione delle interazioni Alice costruisce, firma e invia $\phi_{i+1}^{\sigma_A}$ all'hub. L'hub risponde con la propose $\phi_{i+1}^{\sigma_A,\sigma_H}$ e il token $\tau_{y,\alpha_A}^{\sigma_A,\sigma_H}$ controfirmati.

$$\begin{aligned} \tau_{y,\alpha_B}^{\sigma_A} &=< \nu_y, exp, \mathbb{OFF} > \\ \phi_{i+1}^{\sigma_A} &=< \beta_i^A - \nu_y, \beta_i^H, \tau_{y,\alpha_A}^{\sigma_A}, \mathbb{D} > \end{aligned} \tag{8}$$

Alice presenta $\tau_{y,\alpha_A}^{\sigma_A,\sigma_H}$ in catena eseguendo la funzione attach dello smart contract. Lo smart contract aggiorna il balance on-chain di Alice in $\overline{\beta_A} + \nu$.

3.3.7 Ricarica a caldo

Effettuare una ricarica a caldo significa spostare ν fondi dal balance on-chain di Alice $\overline{\beta^A}$ a quello off-chain β_i^A.

Precondizioni
a) $\{\alpha^A\} \subseteq \Pi^H$
b) L'ultima propose confermata nel canale di Alice è ϕ_i^A
c) Il balance on-chain di Alice è $\overline{\beta_A}$

Descrizione delle interazioni Alice esegue la funzione detach dello smart contract passando come parametri α_A e ν. Lo smart contract aggiorna il

balance on-chain di Alice in $\overline{\beta^A} + \nu$. Una volta terminata l'esecuzione della funzione, lo smart contract invia all'hub e ad Alice la relativa PoD. Alice costruisce, firma e invia $\phi_{i+1}^{\sigma_A}$ all'hub. L'hub risponde con la propose $\phi_{i+1}^{\sigma_A,\sigma_H}$ e il token $\tau_{y,\alpha_A}^{\sigma_A,\sigma_H}$ controfirmati.

$$\begin{aligned} \tau_{y,\alpha_B}^{\sigma_A} &=< \nu_y, \perp, \mathbb{OFF} > \\ \phi_{i+1}^{\sigma_A} &=< \beta_i^A + \nu_y, \beta_i^H, \tau_{y,\alpha_A}^{\sigma_A}, \mathbb{A} >_(\ \sigma_A) \end{aligned} \tag{9}$$

3.3.8 Chiusura di un canale

Precondizioni
a) $\{\alpha^A\} \subseteq \Pi^H$
b) L'ultima propose confermata nel canale di Alice è ϕ_i^A

Descrizione delle interazioni Alice porta in catena l'ultima propose ϕ_i^A con la funzione close dello smart contract. Lo smart contract registra la richiesta di chiusura del canale e avvia un timer di durata pari a una costante G dello smart contract, detta grace period. Scaduto il timer, Alice può ritirare tutti i suoi fondi $\overline{\beta^A} + \beta_i^A$ eseguendo la funzione withdraw dello smart contract.

3.3.9 Riscossione di un pending token

Un client può riscuotere dei pending token, ovvero dei token non ancora scaduti o utilizzati, durante il grace period.

Precondizioni
a) Alice ha avviato la chiusura del canale.
b) Il timer G non è ancora scaduto.

Descrizione delle interazioni Alice presenta in catena un pending token utilizzando la funzione redeemToken dello smart contract. L'esecuzione di questa funzione non corrisponde con il prelievo immediato del token. Una notifica della presentazione del token corrente viene inviata all'hub. Una volta scaduto G, Alice può riscuotere il suo balance (incrementato del quantitativo del token).

Tentativo di ritirare un pending token già usato Alice presenta in catena un pending token già riscosso. Durante il grace period l'hub può portare in catena la relativa PoD del token utilizzando la funzione argueRedemptionToken. Alice viene punita per il suo comportamento malevolo; tutti i suoi fondi (on-chain e off-chain) vengono trasferiti all'hub.

Capitolo 4

Progettazione e sviluppo

In questo capitolo si descrive la fase di progettazione e di sviluppo. In particolare in sezione 4.1 si descrivono le tecnologie utilizzate e le motivazioni dietro la loro scelta. In sezione 4.2 si descrive lo smart contract che supporta FulgurHub. In sezione 4.3 si descrive il modulo software utilizzato dagli utenti di FulgurHub. In sezione 4.4 si descrive il modulo software utilizzato dal gestore di un FulgurHub.

4.1 Le motivazioni tecnologiche

4.1.1 La blockchain: Ethereum

Smart contract Lo smart contract che supporta le operazioni on-chain di questa implementazione di FulgurHub è compatibile con la blockchain di Ethereum.

Ambiente di sviluppo maturo Le motivazioni che hanno mosso la scelta di Ethereum rispetto altre blockchain riguardano il supporto di smart con-

tract e l'ambiente di sviluppo maturo. In particolare sono stati utilizzati i seguenti tool e linguaggi: Solidity, ganache, web3. Solidity è il linguaggio di programmazione C-like turing completo con il quale è stato sviluppato lo smart contract alla base di FulgurHub. Solidity mette a disposizione un compilatore e un debugger. Il compilatore trasforma il linguaggio in codice macchina compatibile con la EVM (Ethereum Virtual Machine). Il debugger di Solidity permette di conoscere lo stato intermedio di uno smart contract durante la sua esecuzione. Ganache è una blockchain di test deployabile in locale, che semplifica la fase di test di uno smart contract; Ganache permette di deployare ed eseguire uno smart contract gratuitamente e in maniera veloce, senza utilizzare la rete principale di Ethereum. Infine web3 è un'interfaccia in JavaScript che permette di eseguire le operazioni più comuni sulla blockchain di Ethereum, come il deployment di uno smart contract, l'esecuzione di una funzione o un pagamento.

Altre soluzioni Esistono altre interessanti soluzioni alternative a Ethereum. Una in particolare è Tezos. Tezos come Ethereum mette a disposizione la possibilità di deployare smart contract con un linguaggio di programmazione turing-completo. Il linguaggio di riferimento è Michelson, un subset di Ocaml che semplifica la verifica formale di correttezza di uno smart contract. Sebbene Tezos non sia stato utilizzato in fase di sviluppo, un suo futuro impiego potrebbe essere facilmente integrabile grazie alla definizione di un'interfaccia dello smart contract (vedi sezione 4.2).

4.1.2 Il linguaggio di programmazione: TypeScript

Supporto di web3 Il modulo software sviluppato esegue le funzioni dello smart contract mediante web3, un'interfaccia JavaScript che permette di eseguire operazioni sulla blockchain di Ethereum. TypeScript, il linguaggio di programmazione adottato è un meta-linguaggio di JavaScript, il che ha

permesso la definizione di un'interfaccia che wrappasse le funzionalità di web3 necessarie.

Tipizzazione forte Alternativamente a TypeScript si sarebbe potuto utilizzare JavaScript. Rispetto a JavaScript è stato impiegato TypeScript dato il supporto della tipizzazione forte. Questo ha permesso di definire interfacce stabili e di intercettare eventuali bug già in fase di compilazione.

4.1.3 Il database lato hub: Redis

Throughput in scrittura Come descritto nel capitolo 3 l'hub riceve dei messaggi firmati dai client che deve memorizzare. Per la natura del protocollo di FulgurHub questi messaggi vengono frequentemente memorizzati e raramente letti. Il numero delle scritture può essere anche ingente. Per questi motivi si è deciso di utilizzare un database chiave valore; in particolare è stato utilizzato Redis, per il suo considerevole throughput in scrittura.

Customizzazione delle qualità nei limiti del teorema CAP Altro motivo per cui è stato adottato Redis rispetto a un altro database chiave-valore è rappresentato dalla possibilità di effettuare tuning delle sue qualità architetturali. In particolare il teorema CAP dice che un database può avere solo due tra queste caratteristiche contemporaneamente:

- Consistenza
- Disponibilità
- Sharding

Redis permette di scegliere quali di queste due caratteristiche avere. In una prima fase di un FulgurHub ha senso scegliere solamente la consistenza e la disponibilità. Sebbene un requisito essenziale dell'architettura sia la scalabilità, una singola istanza Redis su commodity hardware garantisce garantisce un throughput ampiamente sufficiente [18].

Nel caso in cui si debba aumentare il numero di transazioni al secondo si potrà scegliere tra scalare verticalmente l'hardware o abilitare lo sharding a sfavore della disponibilità.

4.1.4 Il database lato client: LevelDB

Anche il client come l'hub deve essere supportato da un database. In questo caso la scelta è ricaduta su LevelDB. LevelDB è un database chiave valore single process, multi thread basato sulle API linux POSIX. Anche qui la scelta è ricaduta su un database chiave valore date le ottime performance in scrittura [19].

4.2 Lo smart contract

Implementazione in Solidity Lo smart contract è stato implementato in Solidity. La gestione dello smart contract si basa su una mappa ti dipo $indirizzoEthereum \rightarrow Wallet$. Un Wallet è una struttura dati che gestisce tutte le informazioni on-chain di un utente iscritto all'hub: balance on-chain, PoDs, PoAs, timestamp di chiusura di un canale e token riscossi. Oltre a questo gli smart contract mettono a disposizione gli eventi. Gli eventi sono dei messaggi che possono essere pubblicati nel momento in cui una qualche funzionalità di uno smart contract viene eseguita. Questi eventi sono pubblici e chiunque può mettersi in ascolto di eventi sulla base di certi parametri. In questo smart contract sono stati utilizzati vari eventi. *Subscribed* è un evento che viene sollevato quando un nuovo utente si registra; serve a notificare che un utente si è registrato all'hub. *TokenDetached* notifica che un token è stato staccato. *TokenAttached* notifica che un token sia stato attaccato. *WalletClosed* notifica l'hub che un utente sta chiudendo un canale; questa

notifica serve all'hub per verificare che l'utente non abbia presentato una propose non valida.

Interfaccia in TypeScript Le interazioni con lo smart contract avvengono mediante un'interfaccia TypeScript. Sì è deciso di implementare un'interfaccia TypeScript per non legare il particolare tipo di blockchain adottata (Ethereum) con l'implementazione in se. Sebbene infatti la scelta progettuale sia ricaduta su Ethereum, FulgurHub è un protocollo che può essere esteso su diverse tipologie di blockchain.

4.3 Il client

4.3.1 RPC privata

Di seguito vengono descritti gli endpoint dell'RPC privata del client.

/join Permette di effettuare una sottoscrizione all'hub.

/detachOffChainOffChain Permette di effettuare il detach di un token off-chain. Questo token potrà poi riutilizzabile off-chain.

/detachOnChainOffChain Permette di effettuare il detach di un token on-chain. Questo token potrà poi essere attaccato off-chain.

- Join di un hub
- Trasferimento OnChain-OnChain
- Detach di un token OffChain-OffChain
- Detach di un token OnChain-OffChain
- Invio della PoD
- Redimere un pending token
- Attach di un token OnChain

- Regolazione di un pagamento OffChain
- Invio della ricevuta di pagamento

4.3.2 Endpoint pubblici

Ricezione di una PoD

Ricezione di una ricevuta di pagamento

4.3.3 Gestione degli eventi asincroni

Il Monitor

Detach di un token on-chain

Ricezione di una PoD

4.4 Hub

4.4.1 Endpoint pubblici

Ricezione di una propose

Ricezione di una ricevuta di pagamento

4.4.2 Gestione degli eventi asincroni

Il monitor

Join di un utente

Chiusura di un canale

**Ritiro di un pending token*

Capitolo 5

Prove sperimentali

5.1 Gli obiettivi

Verifica performance

Verifica scalabilità architetturale

5.2 L'approccio adottato

Benchmark server

Docker swarm

Transazioni seriali

Transazioni concorrenti

Simulazione della latenza di rete

5.3 Il throughput lato client

Al variare della RAM

Al variare della CPU

Al variare della latenza

5.4 Il throughput lato hub

Al variare della RAM

Al variare della CPU

Al variare della latenza

5.5 Considerazioni

Performance e rete

Scalabilità dell'hub

Capitolo 6

Conclusioni e sviluppi futuri

Autogestione finanziaria

Endpoint denominati in maniera diversa

[1] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. 2016. Cryptocurrencies without proof of work. In *International conference on financial cryptography and data security*, 142–157.

[2] Thomas Bocek, Sina Rafati, Bruno Rodrigues, and Burkhard Stiller. 2017. Coinblesk–a real-time, bitcoin-based payment approach and app. *Blockchain Engineering* (2017), 14.

[3] Vitalik Buterin and others. 2014. A next-generation smart contract and decentralized application platform. *white paper* (2014).

[4] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 acm sigsac conference on computer and communications security*, 3–16.

[5] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. 2017. TumbleBit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and distributed system security symposium*.

[6] Sunny King and Scott Nadal. 2012. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August* 19, (2012).

[7] Jae Kwon. 2014. Tendermint: Consensus without mining. *Draft v. 0.6, fall* (2014).

[8] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 acm sigsac conference on computer and communications security*, 17–30.

[9] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[10] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network: Scalable off-chain instant payments. *draft version 0.5* 9, (2016), 14.

[11] Federico Spini. 2018. Fulgur: Hybrid trustless wallet. (2018).

[12] The bitcoin boom | the new yorker.

[13] Average number of transactions per block.

[14] Visa inc. Is a global payments technology company that connects consumers, businesses, financial institutions and governments in more than 200 countries and territories, enabling them to use electronic payments instead of cash and checks.

[15] A proof of stake design philosophy | vitalik buterin.

[16] State channels - an explanation.

[17] Raiden network.

[18] How fast is redis? – redis.

[19] Google/leveldb: LevelDB is a fast key-value storage library written at google that provides an ordered mapping from string keys to string values.