UNIVERSITÀ DEGLI STUDI ROMA TRE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Tesi Di Laurea Magistrale

Implementazione in TypeScript e sperimentazione dell'architettura FulgurHub per la scalabilità blockchain

Laureando

Federico Ginosa

Matricola 457026

Relatore

Alberto Paoluzzi

Correlatore

Federico Spini

Anno Accademico 2017-2018

Ad Ada Lovelace

Ringraziamenti

Grazie a tutti i professori che ho avuto il piacere di conoscere in questi ultimi 5 anni, in particolare il Professor Alberto Paoluzzi, Relatore di questa tesi.

Ridurre in parole i ringraziamenti che meriterebbe l'Ing. Federico Spini, Correlatore di questa tesi non è facile. Federico non si è limitato a guidarmi e consigliarmi, ma ha saputo trasmettermi l'importanza di un metodo educativo basato su un percorso di consapevolezza dialettico.

Ancora è difficile capacitarsi della fortuna avuta nell'incontrare una persona come Marco; ringrazio il mio Amico per avere reso così interessanti le ore di studio e per avermi mostrato la verità lapalissiana celata da una distribuzione di Poisson.

Zattera tra gli impervi mari della vita, questo per me sono sempre stati i miei genitori, che in queste righe ringrazio per non essersi mai stancati di indicarmi il nord.

Inoltre ringrazio tutti i miei compagni di studio, in particolare Alessio, Antonio, Claudia e Lorenzo grazie ai quali questo percorso è stato ancora più divertente.

E infine ringrazio Sara, che mi fa sentire importante anche se non conto niente.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

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 [14].

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 [4].

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 [2].

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 [8]. 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 [19].

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 [11], [1], [12].
- **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 [13], [5].
- **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

Questo Capitolo descrive 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 sugli 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 nasce con l'obiettivo di risolvere il problema del double spending in un sistema peer-to-peer decentralizzato e trustless [14]. 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. Letteralmente trustless significa ”senza fiducia”; in questo contesto in realtà si intende che la fiducia viene spostata da un'entità centrale al protocollo.

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.

Pseudo-anonimato nella blockchain Bitcoin è pesudoanonimo. Le transazioni sono pubbliche, ma non sono direttamente accoppiate all'identità reale del pagante o del pagato, bensì ad uno loro pseudonimo (la chiave pubblica). Per migliorare la privacy, Satoshi Nakamato nel suo paper consiglia di utilizzare uno pseudonimo diverso per ogni singola transazione effettuata [14]. Questo accorgimento non basta però a rendere il sistema anonimo. Le informazioni sensibili rimangono in chiaro e pubbliche; se l'identità reale fosse mai associata alla chiave pubblica, tutte le precedenti transazioni effettuate da quest'ultimo verrebbero associate all'identità reale. Diverse soluzioni alternative a Bitcoin che migliorano la privacy degli utenti sono state proposte; Zcash è una di queste. Zcash si basa su zk-SNARKs (non-interactive zero-knowledge proof), uno zero-knowledge protocol. Una prova zero-knowledge permette a una parte di provare a l'altra che una condizione è vera, senza però rilevare i valori che rendono vera la condizione. Per esempio, dato un hash di un numero randomico, una parte può convincere l'altra del fatto che esiste un numero con questo hash, senza però rilevare

quale sia il numero. Questa tipologia di protocollo permette a Zcash di memorizzare delle transazioni sul ledger pubblico, senza però rivelare le informazioni sensibili associate.

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 [6]. 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. In questo senso uno smart contract ricorda il concetto di classe che si ritrova nei linguaggi di programmazione orientati agli oggetti e le operazione che è possibile eseguire i suoi metodi. Queste operazioni (come nei metodi) presentano dei parametri formali, ovvero gli input che l'utente può passare all'esecuzione di un'operazione. Come in altri linguaggi di programmazione orientati agli oggetti, anche negli smart contract esiste il concetto di visibilità delle operazioni. In Ethereum per esempio un metodo può essere:

- **External** Un metodo external può essere richiamato da un altro smart contract.
- **Public** Permettono di definire l'interfaccia pubblica di uno smart contract; un metodo public può essere eseguito da un utente.

- **Internal** Questa operazione può essere acceduta solo dallo smart contract corrente o da quelli che lo estendono. In Java un comportamento simile si ha con i metodi protect.
- **Private** Questa operazione può essere acceduta solo dagli altri metodi dello smart contract correnti.

Scalabilità off-chain Nel Capitolo 1 sono stati introdotti i limiti architetturali della blockchain e le tre categorie di approcci risolutivi: algoritmo del consenso, sharding e off-chain. La scalabilità off-chain è una tra le tre tipologie di soluzioni possibili. Essa consiste nel costruire uno strato applicativo superiore alla blockchain. Questo strato applicativo eredita tutte le funzionalità e le caratteristiche in termini di decentralizzazione, trustless e sicurezza, potenziandone altre. L'approccio consiste nello spostare la maggiorparte delle transazioni che comunemente verrebbero effettuate on-chain, off-chain. Con transazione off-chain si intende l'esecuzione di una transazione sulla base dello scambio di un insieme di messaggi mediante un qualunque mezzo di trasporto alternativo alla blockchain (E.G. un'email, un sms o una connessione tcp). L'idea è che le transazioni on-chain costano in termini di tempo e sono difficili da far scalare, mentre le transazioni off-chain possono scalare e possono essere eseguite in maniera istantanea. La costruzione alla base delle soluzioni di scalabilità off-chain è lo state channel, presentato in sezione 2.2.

2.2 State channel

Gli state channel rappresentano un modo ampio e semplice di pensare a delle interazioni che potrebbero verificarsi sulla blockchain. Essi permettono a due parti di modificare in maniera sicura porzioni della blockchain, limitando al minimo le interazioni con la catena, ovvero la blockchain. Le componenti principali di uno state channel sono:

- **Deposito di stato on-chain** Esso rappresenta la porzione di stato bloccata sulla catena mediante un indirizzo multisignature o uno smart contract. Questo deposito è bloccato in modo tale che un certo numero di partecipanti debba concordare un eventuale aggiornamento.
- **Deposito di stato off-chain** Questa porzione di stato non è registrata sulla blockchain. Essa viene costruita sulla base dello scambio di messaggi off-chain firmati dalle parti. Ciascun aggiornamento del deposito di stato off-chain, invalida il precedente. Costruendo questi messaggi, essi potrebbero essere utilizzati sulla blockchain, sincronizzando stato on-chain e stato off-chain, ma per adesso vengono semplicemente trattenuti. Il costo di un aggiornamento di questo tipo è quello dello scambio di pochi messaggi su un protocollo come tcp o udp.

Quando uno dei due partecipanti dello state channel decide di rendere permanente la scrittura di un deposito di stato off-chain, l'ultimo stato cofirmato viene presentato in catena. Una parte disonesta potrebbe presentare in catena uno stato precedente all'ultimo; nel caso in cui

questo avvenisse, la controparte può discutere l'aggiornamento in catena, provando che è stato presentato uno stato precedente all'ultimo. La prova consiste nel mostrare una proposta con numero di sequenza maggiore firmata dall'utente disonesto.

Come detto questi messaggi scambiati off-chain descrivono un aggiornamento di stato, per esempio la prossima mossa di una partita di tris o un pagamento [7].

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 potenzialmente illimitato di pagamenti off-chain; nella costruzione di seguito presentata la successione degli aggiornamenti di stato viene descritta da un intero senza segno a 256 bit; questo permette di scambiare un numero di aggiornamenti limitato a 2^{256}. 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. I payment channel oltre a rappresentare una soluzione al problema

della scalabilità, migliorano anche la confidenzialità della blockchain. Utilizzando un payment channel, le uniche transazioni visibili sul ledger pubblico sono quelle di apertura e di chiusura del canale; le transazioni off-chain intermedie invece, sono visibili esclusivamente agli utenti che partecipano al canale. Tuttavia questa caratteristica non preclude la possibilità a una delle due parti, di pubblicare i messaggi off-chain della parte avversaria, esponendo in questo modo informazioni sensibili.

Architettura L'architettura del payment channel di seguito descritta è quella utilizzata come base del lavoro svolto in questa tesi. Come detto in Sezione 2.2, le componenti principali di uno state channel sono il deposito di stato off-chain e il deposito di stato on-chain. Nel contesto dei payment channel questi depositi descrivono lo stato attuale del bilancio delle due parti. In particolare il deposito di stato on-chain è memorizzato all'interno di uno smart contract deployato sulla blockchain di Ethereum, il deposito di stato off-chain invece viene memorizzato sulla macchina locale di entrambi gli utenti. Entrambi gli utenti mettono poi a disposizione un server http con degli endpoint pubblici. Questi endpoint pubblici permettono lo scambio dei messaggi off-chain, ovvero dei pagamenti.

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 smart contract, che nelle successive fasi verrà adottato per richiamare le operazioni on-chain che si intende richiamare; ad esempio l'invio di un aggiornamento del deposito di

stato off-chain. In questa fase lo stato on-chain del payment channel è detto `INIT`. Nella fase `INIT` lo smart contract permette di eseguire esclusivamente l'operazione di apertura del canale da parte di Alice.

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, risparmiando in termini di transazioni 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 on-chain del canale diventa `OPENED`. Nello stato `OPENED`, lo smart contract accetta esclusivamente l'esecuzione dell'operazione `join` da parte di Berto.

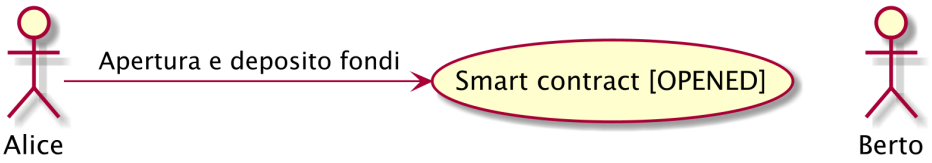

Figura 2: Apertura e deposito fondi on-chain in un payment channel.

Join In un secondo momento Berto effettua il join del canale di pagamento aperto da Alice; è possibile eseguire questa operazione solamente quando lo smart contract si trova nello stato `OPENED`. 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`. Da questo momento in poi lo smart contract accetta l'invio di messaggi che descrivono l'ultimo aggiornamento del deposito di stato off-chain.

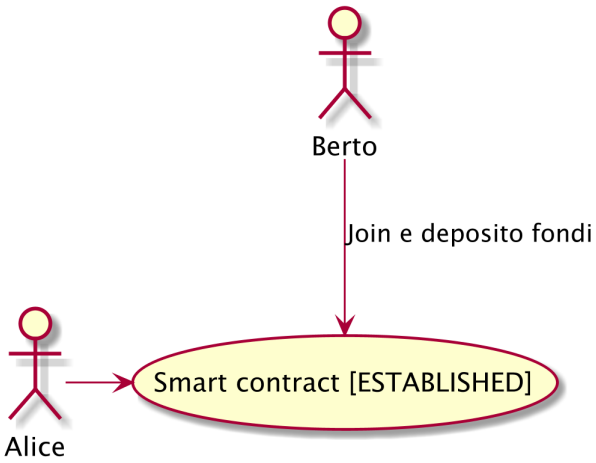

Figura 3: Join e deposito fondi on-chain in un payment channel.

Schema propose/accept I pagamenti off-chain avvengono mediante lo schema propose/accept. Alice (o Berto) propone un aggiornamento dello stato del canale firmando un messaggio. Nell'ambito dello schema propose/accept gli aggiornamenti di stato off-chain prendono il nome di proposta. La proposta viene firmata e inviata da Alice. Berto riceve la proposta, ne verifica la validità ed eventualmente l'accetta rispondendo con la proposta controfirmata. A questo punto è possibile considerare il pagamento come confermato, senza la necessità di ulteriori tempi di attesa. Sebbene l'aggiornamento di stato non sia ancora stato portato

in catena, una proposta cofirmata rappresenta per entrambi le parti una prova inconfutabile di avvenuto pagamento.

Gli endpoint pubblici Nello schema propose/accept 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. In in Tabella 1 si presenta la struttura di una proposta.

Richiesta di chiusura Chiudere un canale significa aggiornare il bilancio on-chain delle parti in modo tale che corrisponda a quello dell'ultima proposta comunemente accordata. Con proposta comunemente accordata si intende un aggiornamento di stato firmato da entrambe le parti. La prima fase di questo processo è detta richiesta di chiusura. In particolare si porta in catena l'ultima proposta 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.

Tabella 1: Struttura di una proposta

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

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` (tempo di grazia).

Discutere una proposta Alice (o Berto) potrebbe non comportarsi correttamente, portando in chiusura una proposta diversa dalla più recente. In questo caso Berto può discutere la proposta durante il `grace period`. Discutere significa portare in catena una proposta firmata da Alice con numero di sequenza maggiore rispetto a quella presentata (vedi Tabella 1). 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 proposta a Berto senza ricevere la controfirma, Berto ha il vantaggio di poter scegliere di chiudere il canale con due proposta, la penultima o l'ultima. Inviare una proposta però coincide con inviare un pagamento, quindi sebbene Berto possa decidere di presentare in catena la penultima proposta, 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 bilancio di Alice e di Berto.

Spesso questi canali sono sbilanciati, ovvero una delle due controparti effettua più pagamenti dell'altra (si pensi a un canale di pagamento instaurato tra il proprietario di un ecommerce e un suo utente). Un canale sbilanciato nel tempo prosciuga il balance di una delle due parti, rendendo il payment channel inutilizzabile. Il canale diventa inutilizzabile poiché una delle due parti ha un bilancio pari a zero e quindi non può più effettuare dei pagamenti. Nella tipologia di canale di pagamento presentata in Sezione 2.2.1, l'unica soluzione a questo problema consiste nel chiudere il payment channel corrente e aprirne un nuovo, caricando i nuovi fondi. Questa soluzione però richiede delle onerose operazioni on-chain; in particolare occorre effettuare il deploy di un nuovo smart contract e successivamente instaurare la connessione con le operazioni di apertura e di join. Gli inextinguishable payment channel (o IPC) superano questo problema, proponendo dei canali di pagamento che permettono di ricaricare o prelevare un'entità `N` di coin a caldo dal proprio bilancio, evitando quindi di dover stabilire un nuovo canale di pagamento [18]. Questo permette di instaurare dei canali che possono rimanere aperti per un tempo indefinitamente lungo; infatti quando il bilancio di una delle due parti si prosciuga, quest'ultima potrà decidere di ricaricare a caldo un certo quantitativo di coin con una singola operazione on-chain. Se invece una delle due parti decide di voler spostare i fondi off-chain sulla catena, potrà farlo con un prelievo a caldo, evitando di dover chiudere il canale.

Schema detach/attach Questo protocollo 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 quan-

titativo di coin del bilancio. La struttura di un token è illustrata 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

Precondizioni Alice e Berto hanno instaurato un IPC. Entrambi hanno un bilancio off-chain pari a 1 ETH.

Ritiro a caldo Alice vuole ritirare a caldo 0.5 ETH; effettua il detach off-chain di un token; invia a Berto una proposta contenente un token di 0.5 ETH che scala dal proprio bilancio. Berto risponde con proposta e token firmati. Il token firmato rappresenta la PoD (Proof of Detach-

ment). 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'attach on-chain di un token depositando nello smart contract 0.5 ETH. Questa operazione on-chain viene notificata a Berto dallo smart contract; tale notifica rappresenta la PoA (Proof of Attachment). A questo punto Alice invia a Berto una proposta in cui effettua l'attach di un token di pari valore e incrementa di 0.5 ETH il proprio bilancio. Berto risponde con la proposta 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 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 payment channel siano una svolta dal punto di vista della scalabilità della blockchain, essi rappresentano uno strumento ancora rudimentale e con un'esperienza utente limitata. Con gli inextinguishable payment channel vengono apportati dei miglioramenti dal punto di vista della UX e della scalabilità; essi infatti grazie alle ricariche e i prelievi a caldo rendono dinamico il quantitativo di fondi bloccato

in un payment channel, limitando al minimo le onerose operazioni di stabilimento del canale. Tuttavia rimane ancora impensabile dover inizializzare un canale di pagamento con ciascun individuo con cui si voglia instaurare un rapporto economico. A questo si preferisce un sistema che permetta di instaurare un singolo payment channel e che consenta di effettuare dei pagamenti con chiunque. Da questa necessità nasce Fulgur Hub [18], ovvero migliorare l'esperienza utente degli IPC e potenziare alcune delle loro caratteristiche.

2.3.2 Caratteristiche

Transazioni istantanee ed economiche In Bitcoin una transazione è usualmente considerata confermata dopo la conferma di 6 blocchi, il che richiede all'incirca 60 minuti. In un IPC basta lo scambio di due messaggi su protocollo http per effettuare e confermare un pagamento. Questo apre nuove prospettive economiche, ad esempio una macchina in cloud potrebbe essere pagata dopo ogni secondo di utilizzo o si potrebbe vedere il proprio stipendio accreditato dopo ogni minuto di lavoro effettuato; 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 bilanci, 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 [10].

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 [3].

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 [16] [15]. 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; un affondo su questa topologia viene fatto in Capitolo 3.

Capitolo 3

Analisi

Questo Capitolo descrive il processo di analisi svolto in questa tesi. 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

3.1.1 Dimostrazione di fattibilità

Un obiettivo di questa tesi è stato dimostrare la fattibilità delle principali feature di FulgurHub, progettando, implementando e verificando la correttezza delle caratteristiche principali di seguito esposte:

Apertura di un wallet In questo contesto aprire un wallet significa aprire un canale di pagamento con un FulgurHub. L'apertura di un canale di pagamento comporta un'operazione on-chain da parte dell'utente e consente di effettuare un numero potenzialmente illimitato di transazioni off-chain.

Pagamento X-Y Come detto in Capitolo 2 ciascun utente di FulgurHub possiede due bilanci, uno on-chain e uno off-chain. FulgurHub consente il trasferimento di fondi da un tipo di bilancio all'altro. Con la formula pagamento X-Y, si intende un tipo di pagamento che sposta i fondi dal tipo di bilancio X (on-chain/off-chain) al tipo di bilancio Y (on-chain/off-chain). In particolare di seguito si elencano tutti i tipi di pagamento di cui questa tesi ha avuto l'obbiettivo di dimostrare la fattibilità:

- **Pagamento OffChain-OffChain** Questo è il pagamento più conveniente in FulgurHub in quanto non necessità di nessuna onerosa operazione on-chain; in particolare questo pagamento sposta un certo quantitativo di coin dal bilancio off-chain del pagante al bilancio off-chain del pagato.
- **Pagamento OnChain-OnChain** Riduce il bilancio on-chain del pagante e incrementa il bilancio on-chain del pagato; non differisce di molto da una classica operazione di pagamento sulla blockchain e infatti richiede un'operazione on-chain.
- **Pagamenti OffChain-OnChain** Questo è il primo tipo di pagamento ibrido. Con pagamento ibrido si intende una transazione che sposta fondi da due tipi di depositi diversi; in particolare un

pagamento OffChain-OnChain con un'operazione atomica riduce il deposito di stato off-chain del pagante e incrementa il deposito di stato on-chain del pagato.

- **Pagamenti OnChain-OffChain** Altro pagamento di tipo ibrido; questa tipologia di pagamento sposta i fondi dal bilancio on-chain del pagante al bilancio off-chain del pagato.
- **Prelievi a caldo** Questa feature viene ereditata dagli IPC e permette a un utente di un FulgurHub di effettuare un prelievo a caldo dei fondi off-chain senza chiudere il canale di pagamento.
- **Ricariche a caldo** Anche questa operazione viene ereditata dagli IPC e consente a un utente di un FulgurHub di ricaricare il bilancio off-chain di un canale di pagamento già aperto.
- **Chiusura di un canale** Un utente del FulgurHub può chiudere il canale di pagamento ritirando i fondi relativi al bilancio off-chain, al bilancio on-chain e eventuali pending token non utilizzati.

3.1.2 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

In FulgurHub ciascun utente possiede due bilancio, uno on-chain e uno off-chain. Effettuare un pagamento significa quindi aggiornare o il deposito di stato on-chain o il deposito di stato off-chain o entrambi nel caso dei pagamenti ibridi. Il deposito di stato on-chain è bloccato da uno smart contract. Mantenere le informazioni relative al deposito di stato off-chain è invece responsabilità dell'utente; a tale scopo l'utente utilizza un client che memorizza le informazioni necessarie su un database dedicato. In Figura 4 si mostra la topologia hub-and-spoke in cui 4 utenti (Alice, Berto, Cecilia e Dario) operano su FulgurHub.

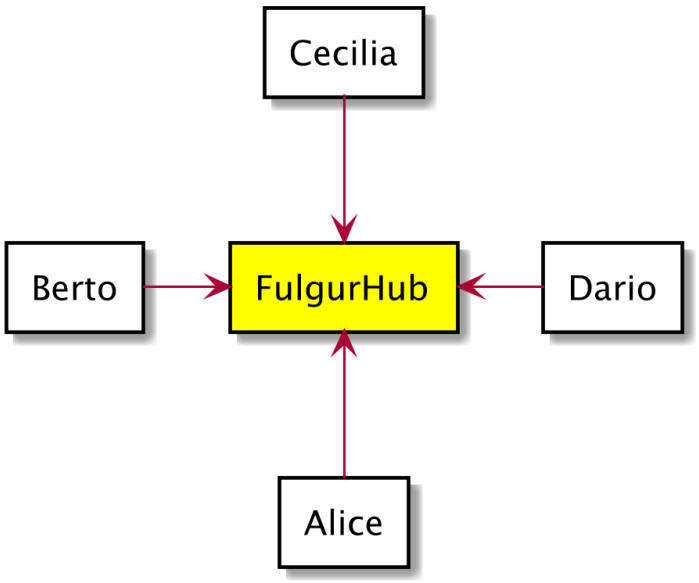

Figura 4: Architettura hub-and-spoke di FulgurHub

Hub L'hub è supportato da un modulo software che interagisce con lo smart contract. Il modulo è stateless, questo permette di replicarlo e di

distribuire il carico su più macchine mediante un loadbalancer, favorendo disponibilità e scalabilità. L'hub è passivo, ovvero non contatta mai direttamente gli utenti; solo gli utenti possono contattare l'hub. La comunicazione da parte degli utenti verso l'hub avviene mediante una connessione http; a tale scopo l'hub mette a disposizione degli endpoint pubblici che permettono di effettuare tutte le principali operazioni, come ad esempio l'apertura di un wallet, le varie tipologie di pagamenti, la discussione di un aggiornamento errato o la chiusura del canale.

Client L'utente contatta l'hub per effettuare le operazioni di cui necessita. La comunicazione tra utente e hub viene mediata da un modulo software detto client. La relazione tra client e hub può essere descritta come una "registrazione trustless" del client al servizio di intermediazione offerto dall'hub [18]. Il client è supportato da un modulo software che interagisce con lo smart contract e l'hub. La registrazione dell'utente coincide con l'instaurazione di una particolare forma di inextinguishable payment channel tra utente e hub 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 da parte dell'hub.

Smart contract Lo smart contract ha varie responsabilità e rappresenta il punto di contatto tra gli utenti dell'hub e la blockchain. Il primo uso tangibile dello smart contract, lo si ha in fase di registrazione di un wallet; questo scenario d'uso applicativo infatti si fonda sull'apertura di un payment channel, che richiede come visto in Capitolo 2 un'operazione on-chain, ovvero un'operazione che faccia uso dello

smart contract. Inoltre lo smart contract viene utilizzato ogni qualvolta si debba effettuare un pagamento che abbia come punto di partenza o di arrivo il deposito di stato on-chain, in particolare i pagamenti: OnChain-OnChain, OnChain-OffChain e OffChain-OnChain. Altra responsabilità dello smart contract riguarda la ricarica e il ritiro di coin a caldo e la chiusura di un canale di pagamento. Infine esso supporta una relazione trustless tra i client e l'hub, ovvero permette l'uso dell'hub in assenza di fiducia reciproca. In particolare lo smart contract deve essere utilizzato ogni qualvolta una delle parti non si comporta correttamente.

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 ($\phi_i^{\sigma_C,\sigma_H}$).

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

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.

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

Una propose $\phi_i^{\sigma_C}$ con un token τ_y detached ($\mathbb{D}$) firmato dal pagato 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.

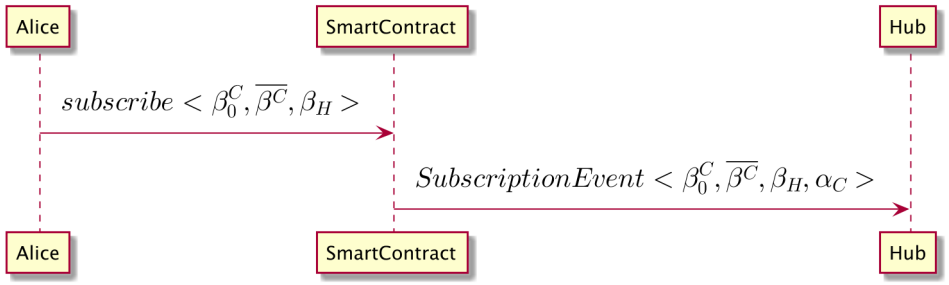

Figura 5: Sottoscrizione di un FulgurHub

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.

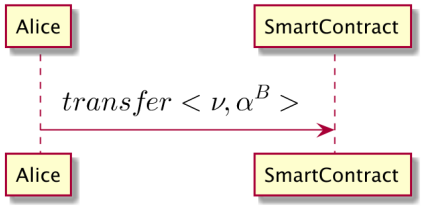

Figura 6: Pagamento OnChain-OnChain in FulgurHub.

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.

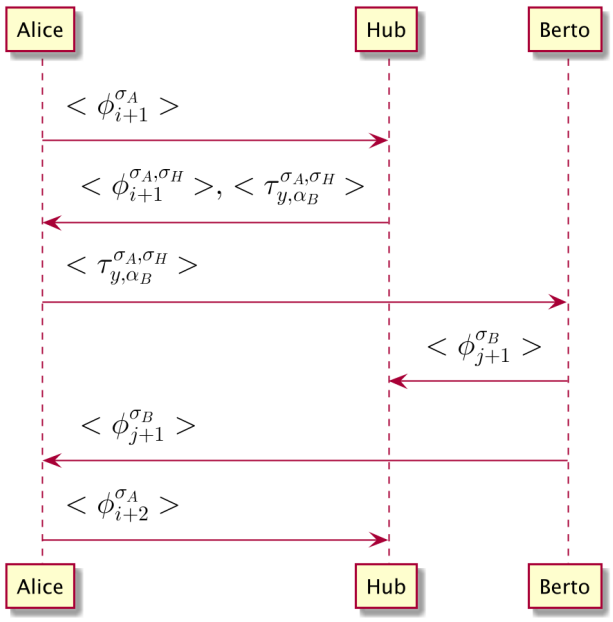

Figura 7: Pagamento OffChain-OffChain in FulgurHub.

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.

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.

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.

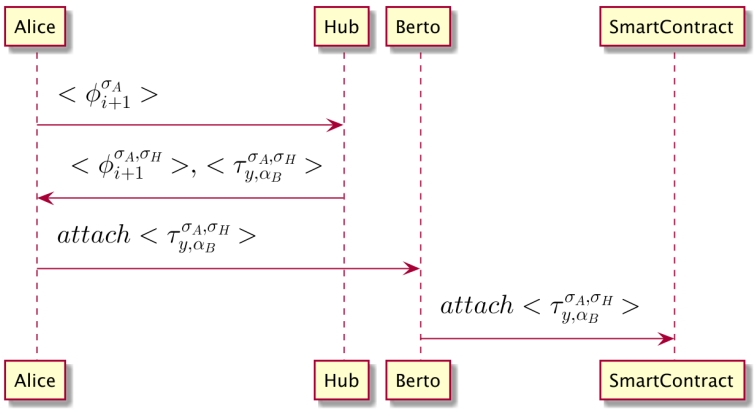

Figura 8: Pagamento OffChain-OnChain in FulgurHub.

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.

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

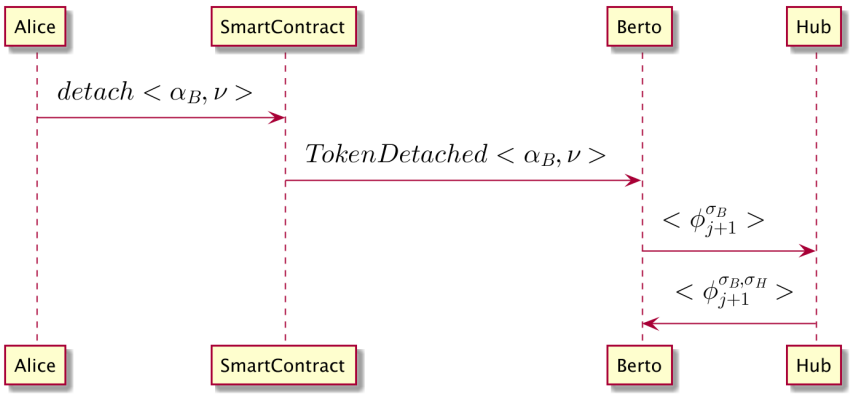

Figura 9: Pagamento OnChain-OffChain in FulgurHub.

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 $\overline{\beta_A} + \nu$. Un diagramma delle interazioni viene fornito in Figura 10.

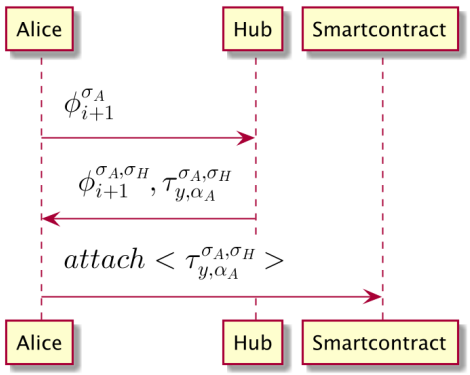

Figura 10: Prelievo a caldo in FulgurHub.

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. Un diagramma del protocollo viene fornito in Figura 11.

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

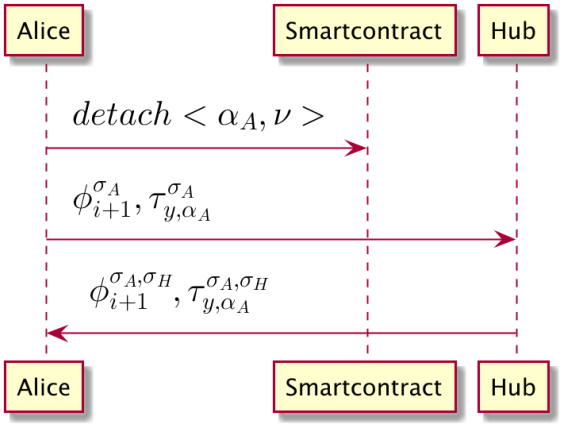

Figura 11: Ricarica a caldo in FulgurHub.

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.

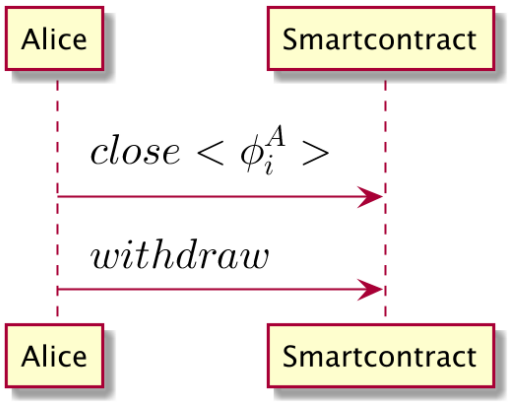

Figura 12: Chiusura canale in FulgurHub.

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

Questo Capitolo descrive responsabilità, requisiti, motivazioni tecnologiche e dettagli implementativi di FulgurHub. In particolare in Sezione 4.1 si descrivono le funzionalità dello smart contract e la sua interfaccia, in Sezione 4.2 si descrive il client e in Sezione 4.3 si descrive l'hub.

4.1 Smart contract

4.1.1 Requisiti e responsabilità

Lo smart contract è il punto di contatto tra lo stato off-chain e quello on-chain di FulgurHub. Esso deve permettere la gestione delle informazioni on-chain necessarie mediante una mappa del tipo $ID(utente) \rightarrow Wallet$. $ID(utente)$ è un identificativo univoco dell'utente (E.G. il suo indirizzo

pubblico) e `Wallet` è una struttura dati; di seguito vengono illustrati tutti i campi di questa struttura dati:

- **Balance on-chain** Si tratta di un intero senza segno che rappresenta il bilancio dell'utente registrato sulla blockchain. Questo bilancio varia ogni volta che viene effettuata un pagamento da o verso la catena. In particolare i pagamenti che modificano il valore di questo campo sono i pagamenti OnChain-OnChain, i pagamenti OnChain-OffChain e pagamenti OffChain-OnChain.
- **PoDs** Questo campo rappresenta una lista di prove di avvenuto distacco di un token da parte dell'utente associato al wallet corrente. Quando un token viene staccato in catena, esso viene memorizzato all'interno di questa lista. Ciascun utente ha la propria lista di token staccati. In questo modo non è possibile staccare più volte lo stesso token.
- **PoAs** Un token oltre ad essere staccato può essere attaccato che equivale al concetto di spesa di un token. Anche in questo caso è presente una lista per ciascun utente, denominata `PoAs` (proofs of attachment). Questa lista contiene tutti i token che sono stati attaccati dall'utente. Memorizzare la lista di proof of attachment consente di evitare il problema della doppia spesa di un token. Ogni qualvolta un utente dell'hub tenta di attaccare un token, lo smart contract verifica che esso non sia contenuto all'interno di questa lista; nel caso in cui il token sia già presente viene sollevata un'eccezione e l'operazione non viene portata a termine.
- **Latest propose** La chiusura del canale avviene in due fasi, la richiesta di chiusura e la finalizzazione della chiusura con il relativo

sblocco dei fondi. La richiesta di chiusura viene effettuata da uno degli utenti dell'hub che decide di voler chiudere il proprio wallet. Essa avviene mediante l'esecuzione di un'operazione on-chain in cui viene portata in catena l'ultima propose concordata tra utente e hub. Questa propose presentata in chiusura viene memorizzata nel campo latest propose. La memorizzazione di questo campo on-chain è necessaria per permettere alla controparte di discutere la proposta nel caso in cui non fosse realmente l'ultima concordata (vedi Capitolo 3).

- **Timestamp chiusura** Quando viene richiesta la chiusura del canale, oltre all'ultima propose viene memorizzato un timestamp. Questo campo è necessario in quanto l'operazione di finalizzazione può essere eseguita solo quando è trascorso un periodo di tempo pari al `grace period`. Lo smart contract confronta il timestamp attuale con quello di chiusura per verificare che sia trascorso il tempo necessario.

Un utente dell'hub che vuole interagire con il suo stato on-chain può farlo eseguendo una delle operazioni messe a disposizione. Queste operazioni riguardano l'iscrizione all'hub, i pagamenti ibridi, la chiusura di un wallet e la riscossione di pending token. Oltre a questo lo smart contract mette a disposizione degli eventi. Gli eventi sono dei messaggi che possono essere pubblicati nel momento in cui una qualche funzionalità viene eseguita. Questi eventi sono pubblici e chiunque può mettercisi in ascolto. Di seguito gli eventi messi a disposizione:

- **Subscribed** Un utente per registrarsi a un FulgurHub non de-

ve contattare direttamente l'hub. L'unica operazione richiesta dall'utente è l'esecuzione dell'operazione di registrazione del relativo smart contract. Quando l'utente esegue questa operazione on-chain, un evento denominato `Subscribed` deve essere sollevato dallo smart contract; questo evento descrive le caratteristiche del Wallet registrato: l'identificativo dell'utente, i bilanci off-chain iniziali di utente e hub e il bilancio on-chain iniziale dell'utente. Per considerare una registrazione conclusa, l'hub deve prendere coscienza di essa, memorizzando le informazioni relative al wallet sul proprio database locale; a tale scopo l'hub si registra all'evento `Subscribed`.

- **TokenDetached** Quando si effettua un prelievo a caldo utente e hub concordano il distacco di un token mediante lo scambio di messaggi off-chain. Terminata questa operazione l'utente presenta in catena il token effettuando il distacco. In questo contesto lo smart contract deve rilasciare un evento denominato `TokenDetached`. L'hub si registra a questo evento; registrandosi a questo evento prende coscienza del fatto che un token che ha firmato è stato effettivamente distaccato.
- **TokenAttached** Quando un token viene attaccato in catena, l'evento `TokenAttached` deve essere sollevato. Questo evento permette all'hub di prendere coscienza dell'avvenuta spesa di un token da parte dell'utente.
- **WalletClosed** La richiesta di chiusura di un canale con l'esecuzione della relativa operazione on-chain deve coincidere con il rilascio dell'evento `WalletClosed`. Questo evento permette

all'hub di prendere coscienza dell'avvenuta richiesta di chiusura del canale, permettendogli di discutere la proposta presentata nel caso in cui non fosse valida.

4.1.2 Motivazioni tecnologiche

La blockchain presa come riferimento è Ethereum. Le motivazioni che hanno mosso la scelta di questa blockchain rispetto ad altre riguardano il supporto di smart contract e l'ambiente di sviluppo maturo. In particolare è stato utilizzato Solidity per lo sviluppo dello smart contract, Ganache come blockchain di test locale e web3 come interfaccia JavaScript per interagire con la blockchain di Ethereum.

Linguaggio di programmazione dello smart contract Solidity è il linguaggio di programmazione C-like turing completo con il quale è possibile sviluppare gli smart contract in FulgurHub. Esso 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.

Rete blockchain di test Ganache è una blockchain di test locale, che semplifica la fase di test di uno smart contract; permette di mettere in produzione ed eseguire uno smart contract, senza utilizzare la rete principale di Ethereum, abbattendo costi e tempi di sviluppo.

Interfaccia smart contract Web3 è un'interfaccia in JavaScript che

permette di eseguire le operazioni più comuni sulla blockchain di Ethereum (E.G. il deployment di uno smart contract, l'esecuzione di una funzione o un pagamento). Le interazioni con lo smart contract non avvengono direttamente con web3, ma sono wrappate da un'interfaccia di più alto livello. Sì è deciso di utilizzare questa interfaccia per non legare il particolare tipo di blockchain adottata con l'implementazione in se. Sebbene infatti la scelta progettuale sia ricaduta su Ethereum, questo approccio consente di estendere le funzionalità implementate su diverse tipologie di blockchain. Il linguaggio di programmazione adottato per implementare l'interfaccia di livello più alto è TypeScript; è stato utilizzato TypeScript rispetto a JavaScript dato il supporto della tipizzazione forte. Questo ha permesso di definire interfacce stabili e di intercettare eventuali bug già in fase di compilazione.

Altre soluzioni tecnologiche Esistono altre interessanti soluzioni alternative a Ethereum. Una in particolare è Tezos. Tezos come Ethereum mette a disposizione la possibilità di mettere in produzione uno 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 dell'interfaccia di alto livello dello smart contract.

4.1.3 Dettagli implementativi

Interfaccia in TypeScript Di seguito viene esposta l'interfaccia di alto livello dello smart contract in TypeScript. Il funzionamento delle singole operazioni è descritto in dettaglio nel Capitolo 3.

```
interface SmartContract {
    subscribe(wallet: Wallet);
    detachToken(token: Token);
    attachToken(token: Token);
    transfer(payeeAddress: string, amount: BigNumber);
    close(propose: Propose);
    redeemToken(token: Token);
    argueRedemptionToken(token: Token);
    withdraw();
    argueClosure(propose: Propose);
}
```

Il tipo Wallet Il tipo Wallet rappresenta la registrazione di un utente su FulgurHub. Esso contiene l'indirizzo pubblico del client e dello smart contract, il bilancio on-chain/off-chain iniziale del client e il bilancio off-chain dell'hub.

Il tipo Propose Il tipo Propose autocontiene tutte le informazioni che descrivono una proposta: il nonce, l'indirizzo pubblico dell'utente, l'indirizzo dello smart contract, il bilancio off-chain corrente del client e dell'hub, il relativo token che si è deciso di attaccare o staccare e la firma della propose.

Il tipo Token Rappresenta un token. In particolare contiene: nonce, indirizzo dello smart contract, indirizzo pubblico del pagato, il quantitativo spostato, il tipo di catena dove può essere attaccato (off-chain o on-chain), la data di scadenza e la relativa firma.

4.2 Client

4.2.1 Responsabilità e requisiti

Il client è il modulo che permette a un utente di interagire con l'hub, gli altri client e lo smart contract; deve rimanere attivo per il tempo di vita del canale di pagamento instaurato con l'hub. Le sue responsabilità riguardano: esecuzione di comandi privati/pubblici, gestione di eventi asincroni e registrazione dei messaggi off-chain scambiati.

Comandi privati/pubblici Un comando privato può essere eseguito solamente dall'utente associato al canale di pagamento. Questi comandi permettono di registrarsi all'hub, effettuare dei pagamenti, chiudere un canale e riscuotere pending token. Un comando pubblico è accessibile a qualunque utente associato a un certo FulgurHub; questi permettono di ricevere pagamenti off-chain e ricevute di pagamento.

Messaggi asincroni Lo smart contract genera delle notifiche; le notifiche sono dei messaggi asincroni. Il client deve poter ricevere e gestire questi messaggi asincroni. Queste notifiche riguardano il detach di un token on-chain e la ricezione di una proof of detachment.

Registrazione messaggi off-chain Tutti i messaggi scambiati off-chain devono poter essere memorizzati in maniera permanente dal client.

4.2.2 Motivazioni tecnologiche

RPC privata / endpoint pubblici L'RPC privata e gli endpoint pubblici permettono di eseguire rispettivamente i comandi privati e pubblici. Entrambi sono stati implementati con un server http Node.js; questo ha permesso di utilizzare TypeScript, mantenendo un unico linguaggio di programmazione per il backend. L'RPC è esposta su una porta privata (`10101`), mentre i comandi che devono esserre accessibili a tutti sono esposti su una porta pubblica (`80`).

Il monitor L'architettura FulgurHub deve gestire un gran numero di eventi asincroni; solo la corretta gestione degli utenti permette di ottenere una corretta e sicura costruzione di FulgurHub. Data la cruciale importanza della loro gestione, si è deciso di localizzare questa responsabilità in un modulo dedicato denominato monitor. Il monitor gestisce due eventi asincroni: `onChainDetachment` e `onProofOfDetachmentPushed`.

- `onChainDetachment` è un evento generato dallo smart contract quando qualcuno effettua il detach di un token on-chain a favore dell'utente corrente.
- `onProofOfDetachmentPushed` è un evento generato quando l'utente corrente riceve una nuova proof of detachment.

Il comportamento legato a un evento non è contenuto all'interno del monitor; il monitor infatti permette solo di agganciare o sganciare a un evento un certo insieme di comportamenti, ovvero di funzioni. Questo approccio consente di estendere facilmente le funzionalità del modulo e quindi migliora la modificabilità del progetto.

Il database La registrazione dei messaggi off-chain è stata delegata a un database. Priorità assoluta di questo database è che non rappresenti un collo di bottiglia per il throughput dei pagamenti. La scelta è ricaduta su LevelDB, un database chiave-valore embedded, single process, multi thread basato sulle API linux POSIX. Le motivazioni che supportano questa scelta riguardano le ottime performance in scrittura di LevelDB [9].

4.2.3 Dettagli implementativi

In questa sezione si descrivono gli endpoint dell'utente. Tutti gli endpoint dell'utente che iniziano con il prefissono `/rpc/` sono privati; gli endpoint che non hanno questo prefisso invece sono pubblici. Gli endpoint privati permettono all'utente di comandare il proprio nodo e di eseguire le operazioni che richiedono la sua autorizzazione, come un pagamento o la richiesta di chiusura di un conto. Gli endpoint pubblici invece non vengono utilizzati dall'utente che possiede il client corrente, ma vengono utilizzati da altri client per inviare delle informazioni all'utente corrente. Nel caso specifico del client relativo agli utenti di un FulgurHub, l'unico endpoint pubblico è `/sendPaymentReceipt`.

Endpoint pubblici e privati del client

- `/rpc/subscribe`
- `/rpc/transferOnChainOnChain`
- `/rpc/detachOffChainTokenOffChain`
- `/rpc/sendProofOfDetachment`
- `/rpc/popProofOfDetachment`
- `/rpc/settleOffChainOffChainTransfer`
- `/rpc/detachOnChainTokenOffChain`
- `/rpc/detachOffChainTokenOnChain`
- `/rpc/attachTokenOffChain`
- `/rpc/redeemToken`
- `/rpc/retrievePaymentReceipt`
- `/rpc/close`
- `/rpc/withdraw`
- `/sendPaymentReceipt`

Un utente utilizzando l'endpoint privato `/rpc/subscribe` può registrare un wallet su FulgurHub. L'unico parametro necessario è denominato wallet e ha il tipo `Wallet`. Il tipo `Wallet` contiene tutte le informazioni necessarie all'apertura di un conto e viene passato come unico parametro http.

```
POST: /rpc/subscribe
{
  wallet: Wallet
}
```

Come visto in Capitolo 3 i pagamenti OnChain-OnChain vengono gestiti dallo smart contract. L'endpoint privato `/rpc/transferOnChainOnChain` avvia il trasferimento eseguendo l'operazione `transfer` dello smart contract. I parametri necessari a eseguire un pagamento OnChain-OnChain sono `recipientAddress`, ovvero l'indirizzo del pagato e `amount` ovvero l'importo che si vuole trasferire.

```
POST: /rpc/transferOnChainOnChain
{
    recipientAddress: string,
    amount: BigNumber
}
```

La prima fase di un pagamento OffChain-OffChain consiste nell'effettuare il detach di un token OffChain-OffChain, ovvero di un token che è stato staccato off-chain e che verrà attaccato off-chain. Questa operazione viene effettuata con l'endpoint `/rpc/detachOffChainTokenOffChain`. I parametri necessari a eseguire il detach sono `addressPayee`, ovvero l'indirizzo pubblico del pagato, `uriPayee` l'indirizzo del server del pagato, `amount` il quantitativo che si intende trasferire e `ttl`.

```
POST: /rpc/detachOffChainTokenOffChain
{
    addressPayee: string,
    uriPayee: string,
    amount: BigNumber,
    ttl: BigNumber
}
```

Una volta ricevuta la proof of detachment dall'hub, essa può essere inviata al client mediante l'endpoint privato `/rpc/sendProofOfDetachment`.

```
POST: /sendProofOfDetachment
{
    proofOfDetachment: Token
}
```

Le proof of detachment vengono aggiunte su uno stack. Il pagato può recuperare la proofOfDetachment affiorante mediante l'uso dell'endpoint privato `/rpc/popProofOfDetachment`, il quale non richiede parametri. Se la PoD è valida, il pagato invia al pagante la ricevuta di pagamento sul suo endpoint pubblico denominato `/sendPaymentReceipt`.

```
POST: /rpc/popProofOfDetachment
```

Quando il pagante di una transazione OffChain-OffChain vuole ribilanciare il canale usa l'endpoint privato `/rpc/settleOffChainOffChainTransfer`. Ribilanciare un canale significa restituire all'hub il quantitativo di token anticipati. Con la corretta esecuzione di questo comando una transazione OffChain-OffChain viene considerata conclusa e confermata.

Per avviare un pagamento OnChain-OffChain occorre utilizzare l'endpoint privato `/rpc/detachOnChainTokenOffChain`. I parametri necessari sono `addressPayee` l'indirizzo ethereum del pagato, `uriPayee` l'indirizzo del pagato, `amount` la cifra che si intende pagare e `ttl` ovvero il tempo di vita del token. Questa operazione permette di concordare

con l'hub una proposta in cui si effettua il distacco di un token che successivamente potrà essere attaccato in catena dal pagato.

```
POST: /rpc/detachOnChainTokenOffChain
{
    addressPayee: string,
    uriPayee: string,
    amount: BigNumber,
    ttl: BigNumber
}
```

Un pagamento OffChain-OnChain è avviato con l'endpoint `/rpc/detachOffChainTokenOnChain`. I parametri necessari per l'esecuzione di questa operazione sono `addressPayee` l'indirizzo ethereum del pagato, `uriPayee` l'indirizzo del pagato, `amount` la cifra che si intende pagare.

```
POST: /rpc/detachOffChainTokenOnChain
{
    addressPayee: string,
    uriPayee: string,
    amount: BigNumber
}
```

Una volta ricevuto un off-chain token, questo può essere riscosso mediante l'endpoint privato `/rpc/attachTokenOffChain`. L'unico parametro necessario a questo endpoint è la PoD, ovvero la prova di avvenuto distacco.

#+end$_{\text{src}}$

Un pending token può essere incassato durante il `grace period` del canale mediante l'endpoint `/rpc/redeemToken`.

```
POST: /rpc/redeemToken
{
    token: Token
}
```

Nel caso in cui il pagato non sia collaborativo un utente può richiedere una ricevuta di pagamento all'hub utilizzando l'endpoint privato `/rpc/retrievePaymentReceipt`, fornendo come unico parametro `clientAddress` l'indirizzo ethereum del pagato.

```
POST: /rpc/retrievePaymentReceipt
{
    clientAddress: string
}
```

Per avvia la chiusura del canale di pagamento occorre utilizzare l'endpoint `/rpc/close`. La chiusura avviene presentando in catena `latestPropose`, ovvero l'ultima propose concordata tra client e hub.

```
POST: /rpc/close
{
    latestPropose: Propose
}
```

Terminato il `grace period`, il client può effettuare il `withdraw`, finalizzando la chiusura del canale. L'operazione di finalizzazione di chiusura di un canale può essere effettuata con l'endpoint `/rpc/withdraw` che non richiede alcun parametro.

`POST`: `/rpc/withdraw`

4.3 Hub

4.3.1 Responsabilità e requisiti

Chiunque abbia abbastanza fondi on-chain può inizializzare un FulgurHub. Per fare questo occorre deployare il relativo smart contract e mantenere costantemente attivo il modulo descritto in questa Sezione. L'hub è un modulo software molto simile al client. Le sue responsabilità riguardano:

- **Esecuzione di comandi pubblici** Gli utenti devono poter contattare l'hub eseguendo dei comandi pubblici.
- **Gestione di eventi asincroni** L'hub deve poter gestire degli eventi asincroni. Nell specifico le notifiche generate dallo smart contract.
- **Registrazione messaggi off-chain** Tutti i messaggi off-chain scambiati con gli utenti dell'hub devono poter essere memorizzati; essi infatti rappresentano delle prove di avvenuto pagamento che potrebbero dover essere presentate nel futuro in catena.

I principali requisiti architetturali dell'hub sono i seguenti:

- **Performance** L'hub deve eseguire le singole operazioni velocemente; questo è essenziale specialmente nel caso in cui occorra gestire frequenti micropagamenti.
- **Scalabilità** L'hub deve poter scalare orizzontalmente; questo significa che per far fronte a un crescente numero di transazione basterà aggiungere dei nodi di calcolo.
- **Modificabilità** La base di codice deve poter essere facilmente modificabile ed estensibile. In particolare non ci si vuole legare fortemente alle tecnologie adottate.

4.3.2 Motivazioni tecnologiche

Gli endpoint pubblici L'hub è un modulo passivo; questo significa che non contatta mai deliberatamente un utente, ma è quest'ultimo che passivamente riceve dei comandi dall'hub. Questi comandi vengono impartiti mediante degli endpoint http pubblici. Come nel client, il server http è stato implementato mediante Node.js; questo ha permesso di mantenere TypeScript come unico linguaggio di backend.

Il monitor Come nel client anche nell'hub la gestione degli eventi asincroni è delegata a un modulo denominato monitor. Il modulo permette di agganciare a un evento un certo comportamento, senza cambiare il contenuto del monitor stesso. L'aggiunta o la rimozione degli eventi è rara, mentre invece la modifica del comportamento legato a un

evento può cambiare frequentemente. Questo facilita l'estensione della gestione degli eventi, migliorando la modificabilità dell'architettura.

Database Come descritto in Capitolo 3 l'hub riceve 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 questo motivo si è deciso di utilizzare un database chiave valore, in particolare Redis, dato il suo considerevole throughput in scrittura [17]. 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'architettura può avere solo due tra queste caratteristiche contemporaneamente:

- Consistenza
- Disponibilità
- Partizionamento

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 un throughput ampiamente sufficiente [17].

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.

4.3.3 Dettagli implementativi

Di seguito vengono descritti gli endpoint pubblici che mette a disposizione un hub Fulgur.

Endpoint pubblici di un FulgurHub

- `/sendPropose`
- `/retrievePaymentReceipt`

Come visto in Capitolo 3 il client effettua dei pagamenti proponendo l'aggiornamento del bilancio off-chain all'hub. Questa proposta viene servita dal client mediante l'endpoint pubblico `/sendPropose` messo a disposizione dall'hub. L'hub a sua volta verifica la proposta, aggiorna lo stato off-chain del canale di pagamento scrivendolo sul database in locale e invia la proposta controfirmata al client.

```
POST: /sendPropose
{
  clientSignedPropose: Propose
}
```

Un client per essere certo che un pagamento OffChain-OffChain sia andato a buon fine necessita di una ricevuta di pagamento. Quando il pagato è completamente collaborativo è lui stesso a fornire questa ricevuta di pagamento al pagante. Quando in un pagamento OffChain-OffChain il pagato non è collaborativo è l'hub a dover fornire la ricevuta di pagamento. Come già detto precedentemente l'hub però è passivo, il che significa che non può contattare direttamente il client. Per questo

motivo un endpoint pubblico `/retrievePaymentReceipt` viene messo a disposizione. Il client infatti eseguendo questo endpoint e fornendo il proprio indirizzo pubblico può ottenere la corrispettiva ricevuta di pagamento.

```
POST: /retrievePaymentReceipt
{
  clientAddress: string
}
```

Capitolo 5

Prove sperimentali

Questo Capitolo discute le prove sperimentali condotte sull'implementazione di FulgurHub. In particolare in Sezione 5.1 si discutono gli obiettivi, in Sezione 5.2 l'approccio adottato, in Sezione 5.3 si mostrano i risultati relativi al client, in Sezione 5.4 si discutono i risultati dell'hub, in Sezione 5.5 viene descritto il profiling dell'operazione di pagamento OffChain-OffChain e infine in Sezione 5.6 si fanno delle considerazioni generali sui risultati ottenuti in termini di performance e scalabilità.

5.1 Gli obiettivi

Verifica performance Un obiettivo delle prove sperimentali è stato verificare le performance dell'architettura; in particolare l'analisi del throughput di client e server relativamente ai pagamenti OffChain-

OffChain. Sebbene siano state implementate anche altre tipologie di pagamento (OffChain-OnChain, OnChain-OffChain e OnChain-OnChain) si è preferito non effettuare prove di performance di tutte le operazioni che interagiscono con la catena. Il throughput delle operazioni che interagiscono con la catena sarebbe limitato superiormente dal throughput della blockchain di riferimento. In questo contesto con throughput si intende il numero di transazioni completate in un secondo.

Profiling Altro obiettivo delle prove sperimentali è stato il profiling dei pagamenti OffChain-OffChain; come visto in Capitolo 3, un pagamento OffChain-OffChain è costituito da un insieme di sotto task; la durata di ciascun sotto task è stata profilata, con l'intento di trovare eventuali colli di bottiglia e di capire quale sia la distribuzione delle operazioni nel tempo.

5.2 L'approccio adottato

Benchmark server Eseguire un test delle performance di FulgurHub richiede il setup di un ambiente complesso e distribuito. In particolare ciascun client e hub dovrebbe risiede su un nodo di calcolo dedicato. A tale scopo è stato realizzato un benchmark server. Il benchmark server permette di automatizzare il setup dell'ambiente di test e di eseguire dei performance test parametrizzati. L'esecuzione delle operazioni avviene mediante una semplice API http REST. Di seguito l'elenco dei comandi messi a disposizione: `/environment`, `/benchmark/offchain/offchain`.

L'endpoint `/environment` permette di effettuare il setup dell'ambiente di test, in particolare vengono deployati i seguenti servizi:

- **Redis** Un'istanza di Redis viene deployata. Essa rappresenta il database dell'hub.
- **Hub** Con hub si intende il server dell'hub. Anch'esso viene deployato su un nodo dedicato.
- **Client** Ciascun client viene deployato su un nodo dedicato. Il numero di client da deployare viene specificato mediante il parametro `numberOfClients`.
- **Ganache** Una blockchain di test deve essere deployata per supportare le operazioni on-chain necessarie come la sottoscrizione dell'hub. A tale scopo è stato utilizzato Ganache.

Per eseguire un benchmark sui pagamenti OffChain-OffChain `/benchmark/offchain/offchain` è l'endpoint che occorre eseguire; è possibile specificare due parametri: `concurrent` e `requests`. `concurrent` indica il numero di coppie di utenti che devono scambiare dei pagamenti in maniera concorrente. `requests` indica quanti pagamenti deve eseguire ciascuna coppia di utenti.

Docker Ciascun nodo dell'ambiente di test è stato deployato su un container LXC. In particolare il benchmark server utilizza le API di Docker, per costruire e distruggere i container di cui necessita. L'uso di nodi virtualizzati rispetto a nodi fisici reali ha vari vantaggi: tra cui l'abbattimento dei costi e dei tempi di sviluppo e la possibilità di aumentare o diminuire le risorse hardware dedicate di ciascun container modificandone la configurazione. D'altra parte la virtualizzazione non

permette di tenere conto della latenza di rete.

Simulazione della latenza di rete Come detto nel precedente paragrafo, la virtualizzazione non permette di tenere conto della latenza di rete. Per questo motivo in fase di test è stata simulata introducendo un ritardo artefatto.

Hardware di riferimento L'hardware utilizzato per eseguire tutti i test di performance ha un processore Intel Xeon E5-2686 v4 (32 cores, 64 threads), 256 GiB di RAM (Amazon EC2 m4.16xlarge). Questa macchina è stata adottata per l'installazione di Docker. Con Docker a ciascun container sono state assegnate le risorse necessarie, sulla base del tipo di test condotto.

5.3 Throughput del client

Con throughput del client si intende il numero di pagamenti seriali al secondo confermati che un singolo utente può effettuare. Esso è stato verificato sia al variare della RAM che della latenza. Il client è realizzato in Node.js, questo significa che viene eseguito un unico processo senza alcun thread di supporto; per questo motivo non sono state effettuate delle prove sperimentali al variare del numero di core a disposizione.

Al variare della RAM Questo test è stato effettuato variando il quantitativo di RAM assegnato ai nodi del client e senza variare quella dedicata all'hub. Come è possibile verificare dai dati in tabella e dal

grafico in Figura 13 il client raggiunge il throughput massimo con 1 o 2 GB di RAM.

Tabella 4: Throughput client al variare della RAM.

N°	RAM	Throughput (tx/s)
1	500MB	27
2	1GB	40
3	2GB	41

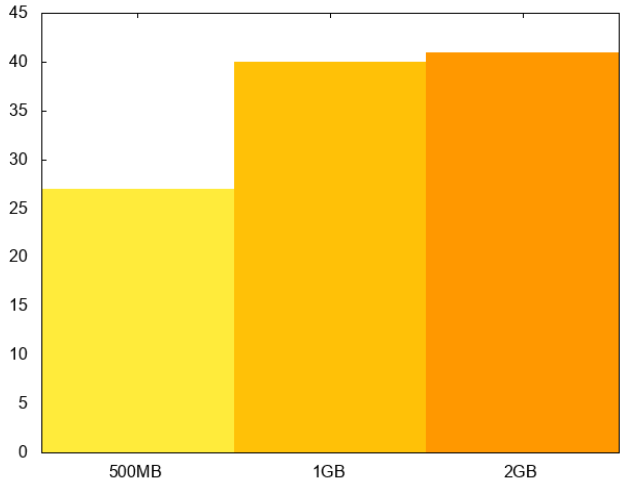

Figura 13: Throughput client al variare della RAM

Al variare della latenza I test sulla latenza sono stati eseguiti fissando la RAM del nodo a 2GB e simulando un ritardo tra tutte le connessioni remote instaurate. In Figura 14 è possibile visualizzare come la latenza incide in maniera negativa sul throughput del client.

Tabella 5: Throughput client al variare della latenza.

N°	Ritardo simulato (ms)	Throughput (tx/s)
1	0	41
2	5	40
3	20	39
4	240	36
5	960	22

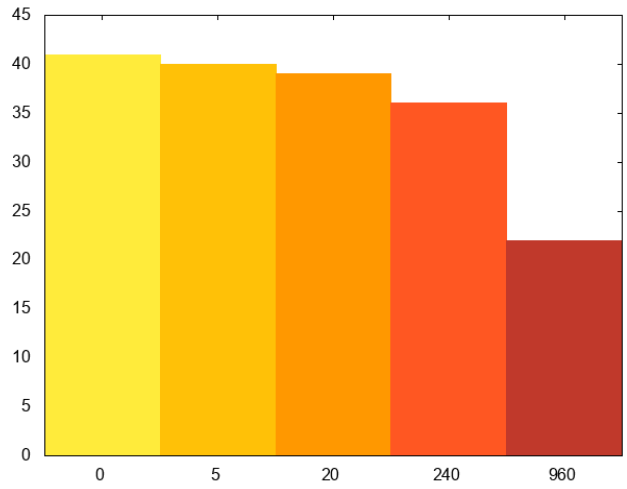

Figura 14: Barchart throughput client al variare della latenza.

5.4 Throughput dell'hub

Con throughput dell'hub si intende il numero di pagamenti concorrenti al secondo confermati che un singolo hub può gestire. Esso è stato verificato sia al variare della RAM che della latenza.

Al variare della RAM Variando il quantitativo di RAM assegnata all'hub e fissando il numero di core utilizzati a 4 è possibile verificare dal grafico in Figura 15 come l'hub raggiunge il throughput massimo con 2/4 GB di RAM. In particolare su un nodo di calcolo con 500MB di RAM il numero di transazioni completate al secondo è pari a 801. Aumentando la RAM a 1GB il throughput raddoppia, raggiungendo 1850 transazioni al secondo. Infine con 2GB e 4GB raggiunge il suo throughput massimo, con 3000 transazioni al secondo.

Tabella 6: Throughput hub al variare della RAM.

N°	RAM	Throughput (tx/s)
1	500MB	801
2	1GB	1850
3	2GB	3084
4	4GB	3091

Al variare della latenza I test sulla latenza relativi all'hub sono stati effettuati fissando le risorse del container relativo all'hub; in particolare sono stati assegnati 2GiB di RAM e 4 core. Inoltre è stato simulato un ritardo tra tutte le connessioni remote instaurate. Come è possibile notare in Figura 16, la latenza incide in maniera negativa sul throughput del hub.

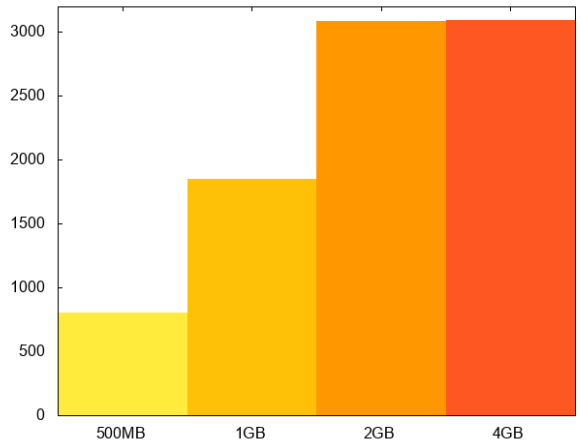

Figura 15: Barchart throughput hub al variare della RAM

Tabella 7: Throughput hub al variare della latenza.

N°	Latenza (ms)	Throughput (tx/s)
1	0	3091
2	5	3078
3	20	3040
4	240	2519
5	960	781

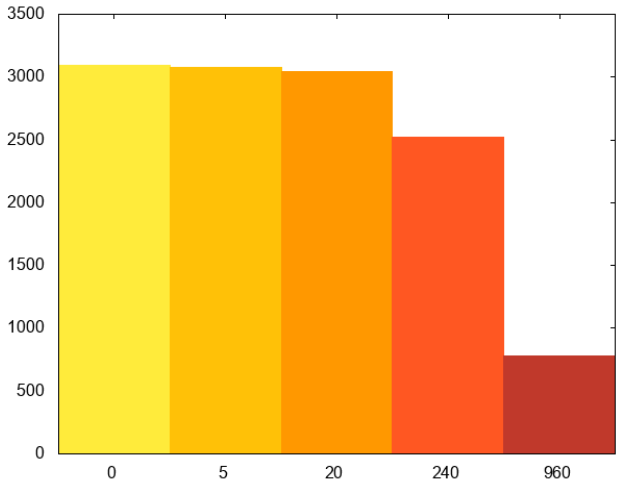

Figura 16: Barchart throughput hub al variare della latenza.

messaggi, vedi Figura 17. In Sezione 5.3 si è visto che su una macchina con 2GB di RAM e un core un client riesce a completare fino a 41 pagamenti al secondo; quindi un pagamento viene completato in circa 24ms.

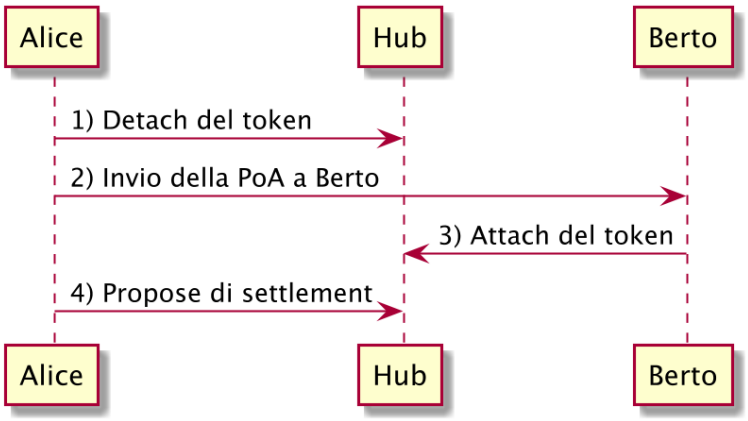

Figura 17: Pagamento OffChain-OffChain in FulgurHub.

Rispetto alle operazioni indicate in Figura 17, la loro distribuzione nel tempo è indicata nel diagramma in Figura 18.

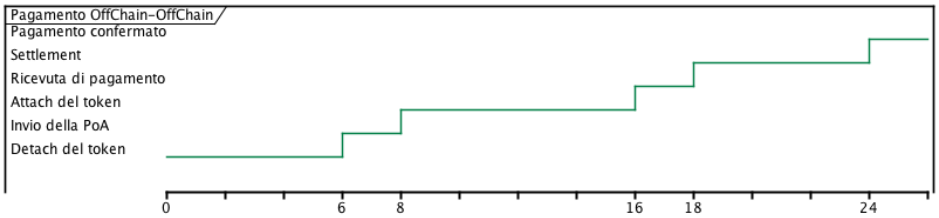

Figura 18: Profiling pagamento OffChain-OffChain

Come è possibile notare dalla Figura 18 le operazioni più onerose risultano essere il detach del token, l'attach del token e l'invio della propose di settlement, ovvero tutte le operazioni che richiedono la firma di un messaggio. Questo risultato è quello aspettato, infatti la firma critto-

grafica richiede un lavoro computazionale maggiore rispetto alla lettura e invio di un messaggio. Node.js dal punto di vista dell'ottimizzazione di operazioni computazionali onerose non è la tecnologia adatta, data la sua natura single thread; per questo motivo la firma dei messaggi è delegata a un modulo software dedicato e ottimizzato scritto in C basato sulle N-API di Node.js.

5.6 Considerazioni

Sulla base dei dati presentati nelle precedenti sezioni, si discutono i risultati in termini di performance e di scalabilità.

Performance Su un singolo nodo con hardware adeguato, l'implementazione di FulgurHub presentata in questa tesi può arrivare a gestire fino a 3091 transazioni al secondo. In Tabella 8 si confronta il throughput di FulgurHub con quello delle principali blockchain. Come è possibile notare FulgurHub permette di ottenere un throughput con tre ordini di grandezza maggiore rispetto a Bitcoin e due ordini di grandezza maggiore rispetto alla blockchain sottostante, Ethereum.

Tabella 8: Throughput FulgurHub a confronto con le principali blockchain.

N°	Tecnologia	Throughput (tx/s)
1	BitCoin	7
2	Ethereum	15
3	FulgurHub	3091

Scalabilità dell'hub Un singolo nodo di FulgurHub permette di ottenere un throughput sensibilmente maggiore rispetto a quello delle principali blockchain. Oltre a questo l'hub è un modulo stateless, ovvero il singolo nodo hardware non ha uno stato; lo stato dell'hub infatti è memorizzato su un database dedicato (Redis); questa caratteristica permette di replicarlo senza particolari difficoltà. Con una configurazione adeguata, basata su un load balancer che distribuisca le richieste su più istanze, il numero di transazioni gestite dall'hub può scalare linearmente, in maniera direttamente proporzionale al numero di nodi.

Capitolo 6

Conclusioni e sviluppi futuri

6.1 Risultati ottenuti

Durante il periodo di lavoro di questa tesi è stata realizzata l'implementazione di un FulgurHub. Questa implementazione ha un throughput di transazioni confermate sensibilmente maggiore rispetto alle principali blockchain e inoltre data la sua natura stateless risulta essere facilmente distribuibile su più nodi di calcolo, favorendone la scalabilità orizzontale. I risultati ottenuti in termini di throughput sono paragonabili al circuito VISA, uno dei più noti circuiti di pagamento centralizzati e trusted. A differenza di un sistema centralizzato FulgurHub mantiene però la proprietà di trustless ereditata dalla blockchain e anzi ne potenzia l'anonimizzazione dei pagamenti, limitando la loro conoscenza esclusivamente ai partecipanti della transazione e all'hub.

6.2 Sviluppi futuri

6.2.1 Pagamenti in denominazione non omogenea

Nell'attuale implementazione di FulgurHub devono collateralizzare il canale con il medesimo simbolo o token, in particolare i pagamenti descritti in Capitolo 3 riguardano solamente lo scambio della cryptomoneta relativa alla blockchain sottostante l'implementazione; nel caso specifico Ethereum (ETH). I token scambiati in FulgurHub possono però rappresentare anche altri concetti.

Il lavoro di implementazione sta attualmente affrontando questo aspetto, la possibilità di consentire lo scambio mediante un FulgurHub di token di denominazione non omogenea. Questo consentirà lo swap atomico di token diversi.

6.2.2 Autogestione finanziaria

Come visto in Capitolo 3 quando si riceve un pagamento OffChain-OffChain esso deve avere un valore minore o uguale al bilancio off-chain dell'hub sul canale di pagamento corrente; in caso contrario il pagamento non può essere ricevuto.

D'altra parte l'hub non conosce le esigenze finanziare dei suoi utenti e quindi non sa a priori quanti fondi off-chain bloccare nel suo bilancio off-chain o con quale frequenza rimpinguarlo. Di seguito viene esposto

un approccio detto di autogestione finanziaria dell'hub, il quale non richiede politiche di gestione attive da parte del manager dell'hub.

La soluzione proposta da FulgurHub si basa sul seguente artificio: l'utente in fase di registrazione di un wallet contrattualizza con l'hub un rapporto tra il suo bilancio off-chain e il bilancio off-chain dell'hub; ad esempio, si ipotizzi che Alice apra un wallet con un rapporto pari a 1; questo significa che il bilancio off-chain dell'hub per conratto deve essere pari almeno al bilancio off-chain di Alice.

Il refill a caldo, in questo scenario che definiamo di "autogestione finanziaria", comporta non solo il refill a caldo del balance off-chain del client richiedente, ma contemporaneamente anche un ritiro a caldo automaticamente innescato per l'hub. Al fine di ripristinare un rapporto concordato e riportato nello smart contract. Specularmente un ritiro a caldo innescato da un utente, attiva un refill a caldo da parte dell'hub. Questo permette all'hub di non partecipare in maniera attiva alla gestione del bilancio.

L'utente chiaramente paga una sottoscrizione temporale, una sorta di abbonamento ai servizi dell'hub che remunera l'anticipo di capitale supportato dall'hub. Un rapporto particolarmente basso potrebbe essere indicato per un utente che riceve pochi pagamenti, pagando di conseguenza una sottoscrizione bassa a causa nel minor anticipo di capitale richiesto all'hub; un utente che invece riceve un gran numero di pagamenti avrà maggiore interesse nel concordare un rapporto più alto, richiedendo dunque un anticipo di capitale più alto e pagando di conseguenza una sottoscrizione più alta.

Gli strumenti per mettere in piedi l'autogestione finanziaria già sono stati implementati nel lavoro di questa tesi, lo step successivo consiste nel combinarli assieme in maniera corretta.

[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] blockchain.com. Average number of transactions per block.

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

[4] Maria Bustillos. 2013. The bitcoin boom.

[5] Vitalik Buterin. 2016. A proof of stake design philosophy.

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

[7] Jeff Coleman. 2015. State channels, an explanation.

[8] 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.

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

[10] 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*.

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

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

[13] 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.

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

[15] Raiden Network. 2017. What is the raiden network?

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

[17] Redis. 2015. How fast is redis?

[18] Federico Spini. 2018. State channels for blockchain scalability: Capabilities, limitations, perspectives. *PhD* dissertation, Engineering Department of Rome Tre University.

[19] Visa. 2015. 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.