POLITECNICO DI TORINO

Laurea Magistrale in Ingneria informatica

Tesi di laurea Magistrale

Software di simulazione basato su Blockchain per infrastrutture di ricarica dei veicoli elettrici

Relatori

Prof.ssa Valentina GATTESCHI

Rel. Alberto BUTERA

Rel. Esterno Alfredo FAVENZA

Rel. Esterno Alessandro MOZZATO

Candidato

Lucio Rocco INGLESE

Aprile 2024

Ad Annalisa e ai miei cari,

Sommario

L'adozione dei veicoli elettrici (EV) è in costante aumento e la conseguente domanda di infrastrutture per la loro ricarica è in crescita. Per aumentare la soddisfazione dei clienti e incoraggiare l'utilizzo dei veicoli elettrici, è fondamentale fornire un processo di ricarica sicuro ed efficiente, in grado di garantire tempi e qualità del servizio. Pertanto, per ottimizzare tale processo è opportuno raccogliere informazioni dettagliate su diversi parametri che influenzano la ricarica, che includono dati proveniente dai veicoli, dall'infrastruttura di ricarica e da modelli di simulazione del traffico urbano. Tuttavia, nonostante la disponibilità abbondante di tali dati il loro utilizzo per ottimizzare il processo di ricarica può presentare sfide complesse. Le cause variano dalla qualità di questi alla crescente preoccupazione degli utenti sulla privacy. L'obiettivo della tesi è sviluppare nuovi strumenti per garantire il funzionamento efficiente, sicuro ed affidabile dell'infrastruttura di ricarica dei veicoli elettrici attraverso l'utilizzo della blockchain. Le infrastrutture elettriche saranno integrate con questa innovativa tecnologia per garantire la privacy dei dati, la sicurezza delle transazioni e l'affidabilità del pagamento. Andremo a sfruttare la blockchain Quorum basata su Ethereum per ottimizzare il tracciamento delle informazioni di ricarica in modo da fornire un tracciamento trasparente e immutabile delle informazioni chiave per il miglioramento delle infrastrutture di ricarica dei veicoli. Infine, per garantire privacy e trasparenza nei pagamenti, sfrutteremo il protocollo Lightning Network.

Indice

Elenco delle tabelle

Elenco delle figure

Listings

Capitolo 1

Introduzione

1.1 Infrastrutture per veicoli elettrici

Le vendite di auto elettriche raggiungono nuovi record di anno in anno. Secondo il rapporto dell'IEA [1], nel 2020 su 25 automobili vendute una era elettrica, mentre nel 2023 il rapporto è di un'automobile su 5. Questo è dovuto, sostanzialmente, ai vantaggi delle auto elettriche che assicurano un elevata **Efficenza energetica** e alle politiche internazionali che spingono, sempre di più, verso una riduzione delle emissioni per attenuare il cambiamento climatico. Con l'espansione dei veicoli elettrici, anche l'accesso a infrastrutture per la ricarica è fondamentale e dovrà espandersi. Come esprime il rapporto dell'IEA riguardante le Tendenze nelle infrastrutture di Ricarica [2]:

> "Sebbene la maggior parte della domanda di ricarica sia attualmente soddisfatta dalla ricarica domestica, sono sempre più necessari caricabatterie accessibili al pubblico per fornire lo stesso livello di comodità e accessibilità del rifornimento di carburante dei veicoli convenzionali. Nelle aree urbane dense, in particolare, dove l'accesso alla ricarica domestica è più limitato, l'infrastruttura di ricarica pubblica è un fattore chiave per l'adozione dei veicoli elettrici."

Per incentivare l'utilizzo dei veicoli elettrici, si prova a far affidamento su infrastrutture di ricarica veloce (**Fast charges**), soprattuto su autostrade , per consentire agli utenti viaggi più lunghi togliendo l'ansia d'autonomia. Un fattore molto importante, collegato alle infrastrutture di ricarica veloce, è la **potenza totale di ricarica** che è possibile fornire ad ogni veicolo elettrico che si connette all'infrastruttura. Essenzialmente, se una stazione può erogare una potenza di 10kW, può fornire in totale 10kW di potenza di ricarica, per i veicoli che stanno ricaricando. Se due veicoli ricaricano contemporaneamente, la potenza disponibile, 10kW, verrà divisa tra questi due, dove ciascun veicolo riceverà una potenza di 5kW. Come afferma L'IEA

[2], durante le fasi iniziali dell'adozione dei veicoli elettrici è normale aspettarsi che la potenza disponibile per ogni veicolo elettrico sia elevata poichè l'utilizzo delle infrastrutture elettriche è basso. Con il crescente utilizzo dei veicoli elettrici, il rischio di sovraccarico della rete può portare ad un degrado della rete stessa. Una gestione efficace della potenza di ricarica è fondamentale per soddisfare la futura domanda di ricarica. Negli ultimi anni, con l'aumento della digitalizzazione, si è verificato un aumento significativo della generazione dei dati relativo sia alle auto elettriche che ai veicoli elettrici. Raccogliere e analizzare questi dati può rilevarsi una strategia vincente per pianificare la capacità di potenza delle stazioni per ogni veicolo, studiare nuovi algoritmi che permettono di migliorare l'esperienza di ricarica per un'automobilista, migliorare le reti elettriche e tanto altro.

Con l'aumento della generazione dei dati da parte dei veicoli e delle infrastrutture elettriche, i rischi, per la privacy dell'utente e il rischio di avere dati non affidabili da utilizzare per il processo di miglioramento della ricarica da parte delle infrastrutture elettriche , aumentano. Come si afferma in "I rischi e le sfide dell'integrazione dei veicoli elettrici nelle città intelligenti" [3], attraverso i dati generati dai veicoli, che vanno dalla posizione del veicolo, potenza della batteria e altro, è possibile, combinando questi con altri set di dati, portare all'identificazione e al tracciamento dell'individuo. Come discusso in precedenza, con l'aumento dei veicoli elettrici è importante, per le infrastrutture, pianificare in modo appropriato la ricarica dei veicoli elettrici e garantire privacy e sicurezza a questi. Come spiegato in "Blockchain, AI and smart grids: The three musketeers to a decentralized EV charging infrastructure" [4], solitamente le infrastrutture elettriche sono collegate con un **CMS(Central Management System)**. Spesso le infrastruttura di ricarica sono connesse a stazioni gateway, attraverso reti LAN, che facilitano il collegamento con il CSM. Questo è un software responsabile delle infrastrutture di ricarica e si occupa di registrare transazioni di pagamento, utenti, monitoraggio di prestazioni e altro. Per la comunicazione tra infrastrutture e CMS esistono molti protocolli. Ad esempio, **l'OCPP(Open Charge Point Protocol)**. L'OCPP ha alcune vulnerabilità che possono compromettere il sistema di ricarica. Alcuni degli attacchi informatici più frequenti sono [5]:

- **MitM(Man in the Middle)**:

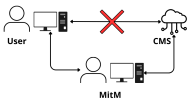

Figura 1.1: Man In The Middle

letteralmente **uomo nel mezzo**. Questa persona può intercettare e manipolare il traffico internet di un utente, mettendosi nel mezzo. L'uomo può accedere ai messaggi che vengono inviati tra due entità e può anche modificarli;

- **ARP Poisoning**: questo è un tipo di attacco **MitM**. Consiste nel simulare di essere il server reale quando in realtà non lo si è. **ARP(Address Resolution Protocol)** viene utilizzato per mappare gli indirizzi IP con gli indirizzi MAC. Il protocollo viene utilizzato nelle reti LAN dove ogni Host di questa rete possiede una tabella con le mappature tra MAC e IP. Il protocollo ARP non è stato progettato per garantire sicurezza, e quindi non verifica che una richiesta provenga realmente da un Host autorizzato. ARP Poisoning (avvelenamento ARP), sfrutta questa debolezza per associare il proprio indirizzo IP di un altro dispositivo al proprio indirizzo MAC;

- **Packet Replay**: l'attacco viene utilizzato per leggere dati sensibili privati sul veicolo elettrico o sul proprietario. Essenzialmente, si utilizzano degli strumenti di analisi del traffico di rete per catturare dei pacchetti. Una volta acquisiti questi pacchetti, che possono contenere varie informazioni, si rilanciano in rete (**reply**). Ad esempio, se una pacchetto contiene informazioni di autenticazione è possibile servirsi di questo per impersonificare un utente leggittimo e quindi ottenere l'accesso a un sistema;

- **Denial of service DoS/DDoS**:

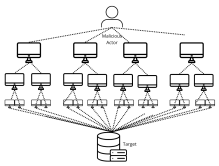

Figura 1.2: DDoS attack

questi attacchi non fanno grossi danni, ma impediscono all'Host di fornire i propri servizi sovraccaricandolo di traffico proveniente da molteplici fonti. Il

traffico generato da un solo computer non può impensierire un server. Ma se applichiamo lo stesso principio con milioni di nodi, come mostrato nella Figura 1.2, questi potranno inviare molti più pacchetti e saturare il server.

1.2 Obiettivi del presente lavoro

Come visto nella 1.1, le infrastrutture di ricarica e i veicoli elettrici presentano alcune problematiche che mettono a rischio la privacy dell'utente e l'acquisizione dei dati. L'obiettivo della tesi è sviluppare nuovi strumenti per garantire il funzionamento efficiente, sicuro ed affidabile dell'infrastruttura di ricarica dei veicoli elettrici attraverso l'utilizzo della blockchain. Le infrastrutture elettriche saranno integrate con questa innovativa tecnologia per garantire la privacy dei dati, la sicurezza delle transazioni e l'affidabilità del pagamento. Andremo a sfruttare la blockchain Quorum basata su Ethereum per ottimizzare il tracciamento delle informazioni di ricarica in modo da fornire un tracciamento trasparente e immutabile delle informazioni chiave per il miglioramento delle infrastrutture di ricarica dei veicoli. Infine, per garantire privacy e trasparenza nei pagamenti, sfrutteremo il protocollo Lightning Network.

1.3 Struttura

Il secondo capitolo 2.1 della tesi introduce la blockchain in generale, il perchè sempre più persone si avvicinano a questo nuovo mondo, e si dà una visione sui concetti base che accomunano le diverse blockchain. Questa introduzione è fondamentale per stabilire le basi per la comprensione dei successivi argomenti trattati.
Nel corso della tesi verranno utilizzate due blockchain : **Bitcoin** e **Quorum**, dove quest'ultima è basata su **Ethereum**. Nella seconda sezione, esploreremo il loro funzionamento, senza entrare nei dettagli di ciascuna implementazione.
Dopo questa panoramica, parleremo del **Protocollo Lightning Network**, fulcro del presente lavoro. Esamineremo il funzionamento del protocollo e vedremo come è possibile effettuare transazioni senza registrarle sulla Blockchain Bitcoin. Approfondiremo i suoi principi fondamentali, le sue differenze rispetto Bitcoin e le possibili applicazioni.
Infine, dedicheremo una sezione ai software utilizzati per lo sviluppo dello strumento di simulazione. Vedremo come questi strumenti sono stati integrati per creare una simulazione completa. Questa sezione fornirà una visione pratica di come è possibile utilizzarli per creare applicazioni Blockchain, con particolare riferimento ad una implementazione per le infrastrutture di ricarica.

Capitolo 2

Stato dell'Arte

2.1 Tecnologia Blockchain

La Blockchain è stata introdotta nel 2008 con l'invenzione di **Bitcoin**, per supportare la criptovaluta **bitcoin**, basata su questa tecnologia. Tale valuta virtuale è stata introdotta nel 2008 e implementata nel 2009 [6] da **Satoshi Nakamoto**. Quest'ultimo è in realtà uno pseudonimo usato per identificare l'autore del **white paper** di Bitcoin. Attualmente l'identità della persona o del gruppo di persone dietro Bitcoin è ancora sconosciuto. Questa tecnologia innovativa sta rivoluzionando numerosi settori, da quello della salute a quello dei veicoli elettrici, fino ad arrivare alle identità digitali. La tecnologia Blockchain mira a garantire i seguenti vantaggi [7] :

- **Decentralizzazione**: l'idea di base è quella di distribuire il controllo a tutti i partecipanti senza avere una autorità centrale o avere fiducia in una terza parte.
- **Immutabilità**: una volta che le transazioni vengono registrate sulla blockchain, è estremamente difficile modificarle e/o cancellarle.
- **Trasparenza**: tutte le transazioni sono condivise e visibili da tutti i partecipanti.
- **Sicurezza**: in generale, le varie blockchain offrono degli standard di sicurezza molto elevati in modo da proteggere l'ecosistema creato da possibili attacchi di nodi malevoli.

2.1.1 Definizione e concetti base

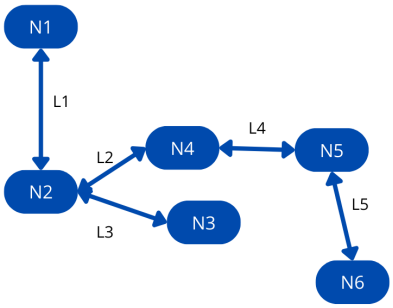

Figura 2.1: Possibile Design di un sistema distribuito

La **Blockchain** (in italiano: **blocchi concatenati**) è un registro digitale distribuito e sincronizzato tra più soggetti presenti in luoghi diversi. Per capire le basi della Blockchain dobbiamo fare un passo indietro e capire cosa sono i **sistemi distribuiti**. Come si afferma in "Mastering Blockchain" [8], i sistemi distribuiti rappresentano un paradigma nell'ambito dell'informatica, dove due o più nodi lavorano insieme per realizzare uno scopo comune. Un **nodo** può essere visto come un partecipante nel sistema distribuito. Nella rete Blockchain, il nodo può svolgere varie funzioni in base al ruolo che assume e alla Blockchain utilizzata. Ad esempio, un nodo può svolgere una funzione di verifica di pagamento (lightweight nodes), di validatore, di mining e altri. La sfida nei sistemi distribuiti è la **coordinazione** tra nodi e la **fault tolerance**, ossia la capacità di continuare a lavorare per raggiungere lo scopo comune anche se si verifica un guasto di un nodo o di un collegamento. La Blockchain è un esempio di sistema distribuito che sfrutta questa architettura per cui i nodi che lavorano insieme convalidano e registrano le transazioni su un registro condiviso e immutabile. Per lo scambio di queste informazione si utilizza una **rete peer-to-peer (P2P)**, in cui non vi è bisogno di un'autorità centrale. I principali componenti della Blockchain sono:

- **Nodi**;
- **Rete Peer-to-peer**;
- **Algoritmi di consenso**;
- **Crittografia**.

2.1.2 Blocchi e Catena di Blocchi

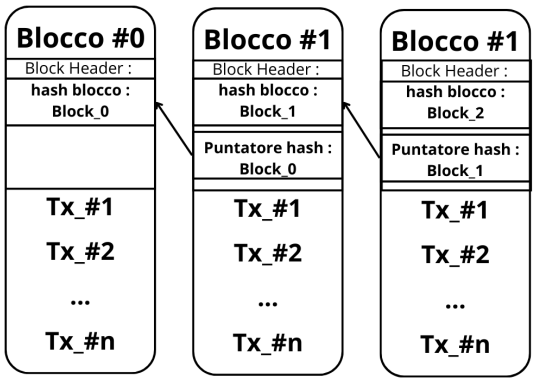

Figura 2.2: Catena di blocchi della Blockchain

Uno degli elementi base di ogni Blockchain è il **Blocco**. La sua struttura dipende dal tipo e dal design della blockchain utilizzata. In generale le blockchain condividono alcune caratteristiche riguardanti i blocchi. Questi ultimi sono delle strutture dati che contengono un insieme di transazioni e altre informazioni utili alla blockchain. In tutti i blocchi è possibile trovare una referenza al blocco precedente (**un puntatore Hash**), tranne che nel primo, chiamato **blocco genesi**. Il blocco genesi è il fondamento su cui viene costruita la rete Blockchain. Va a definire i parametri iniziali e inserisce le informazioni di base che la rete utilizzerà. Ad esempio, il timestamp d'inizio, eventuali parametri di consenso e altri. La dimensione del blocco è variabile rispetto alla blockhchain in uso. Inoltre, dalla creazione di un blocco al successivo può trascorrere del tempo, influenzato da alcuni fattori come la difficoltà del mining, le dimensioni del mempool, il protocollo di consenso e altro. Un altro attributo che viene usato in un blocco è il **nonce(number used once)**, un numero casuale o pseudo-casuale utilizzato dai minatori per creare un nuovo blocco e anche come **contatore delle transazioni**. La Blockchain aggiunge nuovi blocchi nel seguente modo [6]:

1. un nodo firma la transizione con la sua chiave privata;

2. la transazione viene inserita in una mempool, ovvero uno spazio dove risiedono le transazioni che devono essere incluse in un blocco e che non sono state ancora validate;

3. i blocchi devono essere estratti da nodi della rete chiamati **Miner**, essi devono risolvere un gioco matematico per poter estrarre il blocco;

4. i miner selezionano un insieme di transazioni dalla mempool e li convalidano secondo alcuni criteri dettati dalla Blockchain in uso e successivamente creano il blocco;

5. il blocco deve essere validato dai nodi della rete per garantire che rispetti tutte le regole del protocollo, utilizzando un meccanismo di consenso che varia in base alla Blockchain utilizzata;

6. una volta validato il blocco viene trasmesso e aggiunto alla rete Blockchain. Esso diventa parte permanente della blockchain, insieme alle transazioni che contiene, e successivamente viene aggiunto ad ogni nodo della rete.

Il prossimo blocco che verrà creato avrà un puntatore hash a questo blocco.

2.1.3 Transazioni e Registri Distrubuiti

Le **Transazioni** sono un altro elemento base di tutte le blockchain. A seconda della blockchain utilizzata la transazione rappresenta lo scambio di valori o di dati tra due partecipanti alla rete. Come visto in 2.1.2, le Transazioni vengono incluse in un blocco e successivamente registrate in modo permanente sulla Blockchain. Le parti coinvolte in una transazione, e quindi in uno scambio di valore o di dati sono il **mittente** e il **destinatario**. Il mittente è colui che avvia la transazione mentre il destinatario è chi la riceve. Spesso si pensa erroneamente che la transazione rappresenti unicamente lo scambio di valore tra mittente e destinatario. In realtà, i dati possono includere le istruzioni per l'esecuzione di un contratto, come nel caso di **Ethereum**, o lo scambio di NFT (token non fungibili). Per garantire che la trasazione sia stata inviata dal mittente corretto e che i dati all'interno siano integri, nella sua creazione si utilizzano delle tecniche crittografiche che verranno accenate in 2.1.5. Inoltre, nel momento in cui crea la transazione, il mittente deve firmarla con chiavi asimmetriche per garantirne **l'autenticità**, e deve utilizzare tecniche di hashing per assicurarne **l'integrità** [7].

2.1.4 Algoritmo di Consenso

Come affermato in precedenza, la Blockchain è un sistema basato sulla **Decentralizzazione**, secondo cui le decisioni sull'approvazione o meno di una transazione non viene presa da un singolo individuato ma da tutti i nodi della rete. La decentralizzazione richiede un meccanismo di **Consenso** per approvare le transazioni dei partecipanti alla rete e di conseguenza per aggiornare la Blockchain, così da arrivare ad avere un Registro univoco distribuito fra tutti i nodi. Esistono molti Algoritmi di Consenso, e ciascuno di essi ha dei vantaggi e degli svantaggi. In questa tesi tratteremo due di questi algoritmi [6]:

1. **Proof of Woork (PoW)**: algoritmo di Consenso utilizzato da Bitcoin, in cui alcuni nodi, detti miner, devono risolvere un problema crittografico complesso per produrre un blocco e ricevere una ricompensa. Il primo **miner** che trova la soluzione al problema può produrre il blocco aggiungendo le transazioni. Se il blocco è accettato dalla rete, viene aggiunto alla blockchain;

2. **Proof of Stake (PoS)**: algoritmo di consenso utilizzato da Ethereum2.0. In questo algoritmo, solo alcuni nodi della rete possono proporre nuovi blocchi. I nodi in questione sono selezionati in base alla quantità di criptovaluta che bloccano come garanzia(**staking**). Se si prova a "truffare" la rete, la quantità di criptovaluta in staking verrà persa. Si ritiene che chi detiene più criptovaluta in stacking sia più affidabile e abbia maggiore possibilità di essere scelto, poichè non rischierà di truffare la rete e di conseguenza di perdere tutta la sua ricchezza in staking.

Analizzeremo entrambi gli algoritmi in maniera più dettagliata nelle prossime sezioni.

2.1.5 Crittografia e Sicurezza

La crittografia è uno degli elementi chiave nelle Blockchain e assicura alcune delle sue proprietà [8]:

- **Confidentiality(Confidenzialità)**: assicura che i dati siano disponibili solo per entità autorizzate;
- **Integrity(Integrità)**: assicura che l'informazione possa essere modificata solo da entità autorizzate;
- **Authentication(Autenticazione)**: assicura che l'identità di un'entità della rete sia verificata;
- **Non repudation(Non Ripudio)**: si riferisce alla capacità delle Blockchain di dimostrare che un azione sia stata eseguita da un utente.

Per assicurare queste proprietà abbiamo bisogno della crittografia. Non è nello scopo della tesi approfondire queste tematiche, ma vedremo l'idea e le funzioni applicate dalla Crittografia nelle Blockchain. Per garantire **l'Integrità** e **l'Autenticità** dei dati, viene tipicamente utilizzata la **funzione di Hash**.

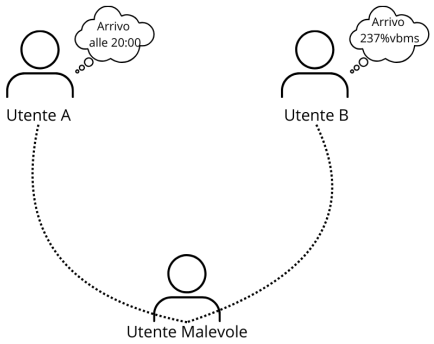

Figura 2.3: Scambio di messaggi senza funzione di Hash

Quando due entità scambiano dei dati, qualcuno potrebbe essere in ascolto e potrebbe intercettarli e modificarli, come mostrato in Figura 2.3. Per controllare l'integrità dei dati e capire se sono stati modificati è possibile utilizzare su di essi una funzione, il cui scopo è quello di generare una prova di come sono fatti nel momento in cui sono stati mandati. Quando il destinatario li riceve, riapplica la stessa funzione calcolando la sua prova. Se le due prove sono uguali, allora il messaggio non è stato alterato. La **Funzione di Hash** serve proprio a generare la prova di integrità del messaggio. Presi dei dati in input, essa genera un riassunto casuale dei dati a lunghezza fissa, detto **digest**. Se i dati cambiano, anche di un solo carattere, il digest cambia completamente forma. Uno degli algoritmi più utilizzato è **SHA 3**, che è l'ultimo arrivato nella famiglia delle funzioni SHA. All'interno della Blockchain, SHA spesso viene utilizzato per calcolare l'Hash del blocco e per proteggere l'integrità delle transazioni all'interno del blocco.
Per garantire l'autenticazione, viene utilizzata la **Crittografia Asimmetrica** (anche detta **crittografia a chiave pubblica**). In questa crittografia, vengono utilizzate due chiavi, rispettivamente per cifrare e per decifrare. Per cifrare viene utilizzata una **chiave privata**, mentre per decifrare una **chiave pubblica**. Le due chiavi sono collegate tra di loro attraverso delle relazioni matematiche. Esse non sono quindi generate randomicamente, bensì sono il risultato di lunghi e complessi processi matematici.

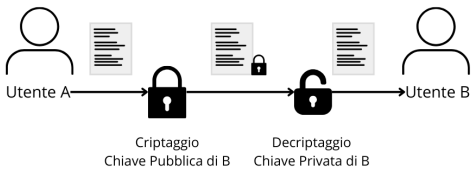

Figura 2.4: Crittografia Asimmetrica

È necessario avere due chiavi, una pubblica e una privata, che interagiscono come spiegato di seguito:
l'entità A invia un testo cifrato con la chiave pubblica dell'entità B, come mostrato in Figura 2.4. L'entità A sa che solo l'entità B può decifrare il testo poichè è l'unico che possiede la chiave privata per poterlo fare. Viceversa, la chiave privata rimarrà segreta e si userà quando si vuole cifrare qualcosa. L'entità B, ricevuto il testo, può decifrarlo con la sua chiave privata. Un classico utilizzo della crittografia asimetrica è l'autenticazione: se io mittente voglio dimostrare la mia identità, mi basterà cifrare il testo con la mia chiave privata. A questo punto, se qualcuno vuole decifrare il testo, può utilizzare la mia chiave pubblica. Se riesce a decifrarlo, sarà sicuro che il mittente sono io.

2.1.6 Benefici della Blockchain

La tecnologia blockchain offre una serie di benefici in diversi settori, che vanno oltre la semplice criptovaluta. Alcuni dei vantaggi della Blockchain sono [8]:

- **Decentralizzazione**: non vi è bisogno di fiducia in un'azienda terza e non esistono intermediari. Il meccanismo di consenso si occupa di validare le transazioni;

- **Immutabilità**: una volta che le transazioni sono state inserite in un blocco e approvate, difficilmente sarà possibile cambiarle grazie alla funzioni crittografiche;

- **Trasparenza e assenza di Fiducia**: chiunque può vedere le transazioni che ci sono sulla blockchain;

- **Sicurezza**: tutte le transazioni vengono registrate solo se si dimostrano essere integre.

2.2 Bitcoin Blockchain

2.2.1 Introduzione a Bitcoin

Il termine Bitcoin è spesso associato ad una criptovaluta non tenendo in considerazione tutti i concetti e le tecnologie che costituiscono la base di questo ecosistema. Bitcoin è un sistema peer-to-peer distribuito e connesso tramite internet. Non esiste quindi un punto di controllo centrale. L'unità di valuta è il **bitcoin**, esso viene utilizzato per memorizzare e per trasmettere valore tra i partecipanti alla rete. Rispetto alle tradizioni monete, questa è totalmente virtuale. Per dimostrare il possesso di bitcoin, gli utenti posseggono delle **chiavi** che sono salvate all'interno di **wallet digitali**. Attraverso le transazioni, è possibile scambiare valore. Le transazioni, prima di essere confermate, vengono inserite in un blocco, validate e successivamente confermate sulla rete. Ogni nodo della rete possiede un registro (**Ledger**), che viene aggiornato ogni qualvolta si va ad aggiungere un blocco alla Blockchain, in modo che i nodi della rete condividano una singola verità. La moneta bitcoin viene creata attraverso un processo chiamato **mining** regolato attraverso algoritmi specifici. La Blockchain Bitcoin è composta da [9]:

- una rete peer-to-peer **decentralizzata**;
- un **registro pubblico** per le transazioni;
- delle **regole di consenso** per la convalida delle transazioni e per l'emissione di valuta;
- un algoritmo **Proof-of-Work** per un meccanismo di consenso decentralizzato.

2.2.2 Struttura della Blockchain Bitcoin

L'ecosistema Bitcoin è basato su diversi componenti che lavorano insieme per il suo funzionamento. Una panoramica dei principali componenti è la seguente:

1. **Blockchain** [10]: la blockchain rappresenta la storia di ogni transazione Bitcoin confermata. Essa è una struttura dati rappresentata da un elenco ordinato e collegato di blocchi composti da transazioni. Essi sono collegati a ritroso, ciascun blocco ha un riferimento al blocco precedente. Ogni blocco all'interno della Blockchain è identificato con un hash, generato utilizzando l'algoritmo di hash crittografico SHA256. Il blocco dispone del campo **hash del blocco precedente**. La grandezza di un Blocco Bitcoin è di 1MB.

2. **Protocollo Bitcoin**: il protoccolo Bitcoin è l'insieme di regole e procedure che regolano il funzionamento della rete Bitcoin.

3. **Nodi** [9]: i nodi sono dei computer su cui è possibile eseguire un software client Bitcoin. Esistono diversi tipi di nodi:

   - **Full nodes**: mantengono una copia completa della blockchain.
   - **Light nodes**: si affidano a dei nodi completi per accedere alle informazioni.

4. **Portafogli Bitcoin** [9]: un portafoglio Bitcoin è un l'interfaccia utente per il sistema Bitcoin. Esso permette di gestire e di trasferire bitcoin. Contiene, inoltre, una o più coppie di chiavi crittografiche, di cui la chiave privata viene utilizzata per firmare digitalmente le transazioni e l'identità del mittente, mentre la chiave pubblica viene utilizzata per ricevere Bitcoin.

5. **Transazioni** [11]: una transizione comunica alla rete Bitcoin che vi è stato uno scambio di valore tra un mittente e un destinatario.

6. **Minatori** [12]: i minatori sono dei nodi della rete Bitcoin e competono per aggiungere nuovi blocchi alla blockchain. I minatori ricevono due tipi di premio:

   - bitcoin per ogni nuovo blocco creato;
   - commissioni per ogni transazione inclusa nel blocco.

   Per guadagnare queste ricompense, i minatori competono in una sfida basata su un algoritmo di hash crittografico.

2.2.3 Transazione Bitcoin

Particolarmente rilevante per lo scopo della tesi è capire come avviene una transazione Bitcoin. Questo perchè il protoccolo Lightning che si appoggia su Bitcoin, come vedremo in 3.3.2, sfrutta una transazione Bitcoin per aprire un canale di pagamento. La transazione Bitcoin comunica alla rete che un proprietario di bitcoin vuole trasferirne da un wallet ad un altro.

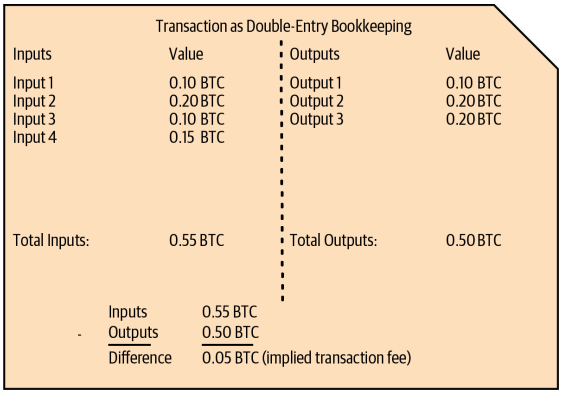

Figura 2.5: Transazione in partita doppia[13]

Come affermato in "Mastering Bitcoin"[11], le transazioni Bitcoin posseggono uno o più **input**. Questi ultimi, si riferiscono a delle transazioni precedenti che contengono bitcoin non spesi: possono essere utilizzati come fondi per una nuova transazione. Le transazioni contengono uno o più **Output**, che definiscono dove vanno i fondi. Come è possibile vedere dalla Figura 2.5, gli input e gli output non rappresentano lo stesso importo. La differnza consiste in una commissione data ai miner per includere la transazione nella blockchain. Per riferirci ad una transazione in Bitcoin utilizziamo il **txid**, generato con una funzione di Hash(SHA-256) che prende in considerazione tutti i dati della transazione, anche input e output. In alcuni casi è necessario che una transazione venga approvata da più entità prima di essere effettivamente registrata nella blockchain. Le transazioni Bitcoin possono utilizzare il **multisignature (multifirma)** che richiedono più chiavi private e quindi più firme affinchè la transazione possa essere effettuata e poi registrata sulla Blockchain Bitcoin. Come riportato in "Mastering Bitcoin" [14], il problema che sorge è il seguente:
Due entità vogliono depositare dei fondi in uno script che può essere speso con la firma di entrambe, ma ciascuna di esse vuole riavere indietro i propri fondi nel caso in cui l'altra entità diventi inattiva. Una soluzione non attuabile con le transazioni legacy è la seguente: Immaginando che le due entità si chiamino Alice e Bob,

- Transazione_0 paga denaro ad Alice e Bob con uno script che richiede la firma di entrambi;
- Viene creata una Transazione_1 che spende l'outpout della Transazione_0 su due output: una che rimoborsa Alice e una che rimborsa Bob. Successivamente, basta far firmare la transazione_1 prima della transazione_0, in modo da avere una via di uscita. Praticamente, Alice firma l'output di Bob e Bob firma l'output di Alice. Cosi facendo, entrambi possono richiedere il rimborso, firmando la transazione che ha già firmato l'altro, senza aver fiducia.

Il problema di questa soluzione nelle transazioni legacy sta nel derivare la transazione_0, in modo da poter conoscere il suo output specifico e farlo diventare l'input della transazione_1. Come affermato precedentemente, è possibile riferirsi alla transazione attraverso il txid calcolato con la funzione di Hash, attraverso l'unione di input e output. Il campo input deve contenere anche le firme di Alice e Bob per essere valido, di conseguenza bisogna per forza creare prima la transazione_0 e poi la transazione_1. **SegWit (testimone segreto)** risolve questo problema, andando a separare l'input dal resto dei dati, e quindi è possibile calcolare il txid senza input e far riferimento ad esso.

2.2.4 Proof-of-Work e Mining

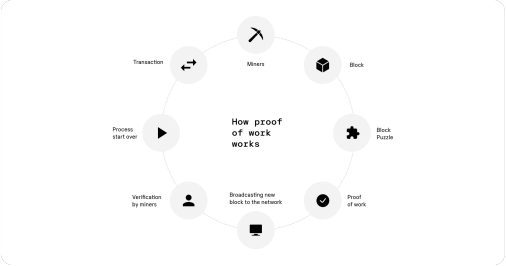

Figura 2.6: Bitcoin Proof of Work[15]

Il **Proof of Work(PoW)** è stato il primo meccanismo di consenso utilizzato nella blockchain, introdotto con Bitcoin nel 2009. Come discusso in [6], questo meccanismo è utilizzato per confermare e convalidare le transazioni e aggiungere nuovi blocchi alla catena. La tecnica è utilizzata per determinare quale nodo otterrà il diritto a pubblicare un nuovo blocco. I nodi della rete, per creare un blocco e successivamente aggiungerlo alla rete, devono risolvere un gioco matematico di difficoltà variabile. I nodi che hanno questo compito vengono chiamati **miner** e il processo è chiamato **mining**. Molti miner sono coinvolti nella creazione di un blocco ma solo il primo a risolvere il problema ottiene una ricompensa in bitcoin e commissioni per le transazioni incluse nel blocco.
Per risolvere il problema matematico, i miner raccolgono le transazioni non ancora confermate situate in una mempool e si combinano con un **nonce**, che rappresenta un numero casuale. Successivamente si calcola l'hash dell'unione delle transazioni e del nonce e si verifica che rispetti alcuni requisiti di difficoltà per essere valido. I requisiti riguardano gli zeri all'inizio dell'hash: più zeri sono richiesti più la difficoltà aumenta. Quando l'Hash è stato trovato, viene diffuso insieme al blocco agli altri miner. Il processo di verifica da parte degli altri nodi è molto veloce in quanto basta ricalcolare l'hash del blocco e del nonce e verificare che sia uguale a quello fornito. Dopo la verifica, il blocco viene diffuso in tutta la rete.

2.3 Ethereum Blockchain

2.3.1 Introduzione a Ethereum

Ethereum è una Blockchain decentralizzata creata con l'obiettivo di eleminare le limitazioni imposte dal linguaggio di scripting di Bitcoin, per consentire agli sviluppatori di costruire e di distribuire delle applicazioni decentralizzate chiamate **DApps**. Possiamo pensare ad Ethereum come una macchina a stati che mantiene traccia di un archivio che può contenere qualsiasi dato esprimibile come una tupla chiave-valore[16]. Ethereum può caricare del codice nella macchina a stati, eseguirlo e memorizzare l'output nella blockchain che porterà ad un cambiamento di stato quando eseguito. Le transazioni sono inviate da utenti attraveso un account Ethereum. Per inviare la transazione vi è bisogno dell'**Ether(ETH)**, che è la criptovaluta di Ethereum. A differenza di Bitcoin, qui le transazioni possono anche riguardare il codice che verrà eseguito. Il codice è presente in speciali contratti chiamati **Contratti Intelligenti(Smart Contract)**. A differenza di Bitcoin, un nuovo blocco è creato ogni 12 secondi ma si utilizza lo stesso meccanismo di referenza, dove un blocco è identificato da un hash e ognuno punta al precedente hash. Lo stato è rappresentato dal bilancio Ether e dai valori salvati[17]. Esso viene modificato quando si presenta una nuova transazione che modifica i valori salvati. Il tutto è gestito dall'**Ethereum Virtual Machine(EVM)**. L'algoritmo di consenso che si occupa di costruire un blocco, vedere se è valido e aggiungerlo alla blockchain Ethereum è il **PoS(Proof of Stake)**.

2.3.2 Struttura della Blockchain Ethereum

Le caratteristiche distintive della Blockchain Ethereum sono[16] :

- **EVM (Ethereum Virtual Machine)**: Ethereum include una macchina virtuale Turing complete chiamata EVM, che consente l'esecuzione di contratti intelligenti su ogni nodo nella rete Ethereum.
- **Contratti Intelligenti(Smart Contract)**: sono programmi scritti in un linguaggio ad alto livello (es. **Solidity**) eseguiti su EVM.
- **Turing Complete**: a differenza di Bitcoin, Ethereum è "turing complete". Un linguaggio di programmazione è Turing-complete se la sua semantica permette di implementare una qualsiasi macchina di Turing. Esso può essere usato per risolvere qualsiasi problema che ammetta soluzione.
- **ETH (Ether)**: Ether è la criptovaluta di Ethereum. Non è nata con lo scopo di divenire una criptovaluta, ma con quello di **valuta di utilità**, usata per le commissioni di transazione e per incoraggiare i minatori a proteggere la rete.

- **Algoritmo di consenso**: Ethereum2.0 a differenza di Bitcoin utilizza il **PoS(Proof of Stake)** per migliorare l'efficienza energetica e per ridurre i costi delle transazioni.

2.3.3 Contratti Intelligenti

In Ethereum a differenza di Bitcoin abbiamo due possibili tipi di conti :

- **conti di proprietà**: sono i conti degli utenti;
- **conti contrattuali**: sono conti controllati dal codice. Questi non hanno una chiave privata come un conto normale, ma eseguono semplicemente il codice che li descrive.

Un **contratto Intelligente(Smart Contract)** è un programma informatico, eseguito in maniera deterministica dall'EVM [18]. Essi vengono scritti attraverso un linguaggio di alto livello, ad esempio **Solidity**. Una volta compilati, vengono inviati alla Blockchain Ethereum. Il contratto viene referenziato attraverso un indirizzo Ethereum. I contratti possono essere richiamati attraverso una transazione, specificando la funzione da richiamare e i fondi che vengono utilizzati per eseguirlo. Una transazione è **atomica**, cioè ci sono due possibili risultati: conclusa con successo o ripristinata. Non è possibile modificare il codice di un contratto quando questo è già stato registrato sulla piattaforma blockchain, ma può essere cancellato, lasciando un account vuoto. Quando il contratto viene richiamato attraverso una transazione, fornisce un risultato **deterministico**.

2.3.4 Proof of Stake

Come affermato in "Exploiting ethereum after "the merge": The interplay between pos and mev strategies"[17], nel Proof of Stake, il tempo è diviso in slot, ognuno dei quali dura 12 secondi. In ciascuno slot viene selezionato un validatore che si occupa della creazione di un nuovo blocco e del suo invio agli altri nodi della rete. Per divenire un validatore e poter essere selezionato, bisogna depositare 32 Ether in uno smart contract. In questo modo si incoraggiano i validatori ad essere onesti, mettendo in gioco i propri Ether, punendo, con la loro perdita, coloro che si comportano in maniera disonesta. Ci sono diversi vantaggi nell'utilizzare il proof of Stake rispetto al Proof of work [19]:

- **Maggiore efficienza energetica**, poichè il processo di validazione non richiede molto lavoro computazionale;
- **Minori barriere di accesso**, poichè i requisiti hardware, per un validatore, sono ridotti;

- **Sanzioni economiche**, in quanto i comportamenti scorretti rendono gli attacchi più costosi del 51%.

2.3.5 DApps

Le **DApps o Decentralized Applications**, sono applicazioni decentralizzate che operano su una rete peer-to-peer anzichè su server centralizzati. Una DApp è qualcosa di più rispetto ad un contratto intelligente. Essa comprende come minimo[16] due elementi : un contratto intelligente e un' interfaccia web. I vantaggi rispetto ad un'applicazione centralizzata sono [20]:

- **Trasparenza**: ogni nodo della rete può esplorare il codice ed essere più sicuro;
- **Resistenza alla censura**: finchè un nodo è connesso alla rete Ethereum, potrà interagire con essa senza interferenze.
- **Resilienza**: una DApp non avrà tempi di inattività, verrà eseguita fino a quando la piattaforma sarà operativa.

Le DApps sono utilizzate in una varietà di settori, tra cui :

- **DeFi(Finanza Decentralizzata)**: DApps che offrono servizi finanziari,
- **DGames (Giochi Decentralizzati)**: giochi online basati su blockchain;
- **DEX(Mercati Decentralizzati)**: piattaforme di Scambio;
- **Identità digitale decentralizzata**: DApps che consentono agli utenti di controllare e di condividere la propria identità digitale in modo sicuro e privato.

Capitolo 3

Lightning Network

3.1 Perchè Lightning Network?

Lightning Network propone una nuova infrastruttura, un protocollo che si appoggia su Bitcoin e, come questo, viene utilizzato dagli utenti per effettuare pagamenti. Lightning Network viene definita **trustless**. Il termine esprime la caratteristica di non richiedere fiducia negli altri partecipanti e quindi si tratta di un sistema **Decentralizzato** come Bitcoin. Inoltre, offre **Trasparenza** e **Sicurezza**.
Ma allora perchè utilizzare Lightning Network e non Bitcoin?
Bitcoin ha diversi problemi: è un sistema basato su un **registro pubblico** globalmente replicato. Questo significa che ogni partecipante deve validare e memorizzare ogni transazione. Ciò genera grande quantità di dati, difficili da scalare. Alcuni dei principali problemi di scalabilità includono [21]:

- **Dimensione Blocco**: attualmente la capacità del blocco Bitcoin è fissa a 1MB. Questo comporta che solo un numero fisso di transazioni possono essere incluse in un blocco. Ad esempio, considerando che ogni blocco di Bitcoin è grande 1MB e ognuno viene creato circa ogni 10 minuti. Considerando che una transazione Bitcoin occupa circa 300B, bitcoin può supportare meno di 7 transazioni al secondo (tps). Volendo fare un paragone con la rete di pagamento Visa, essa ha raggiunto un picco di transazioni di 47000 tps durante l'estate del 2023 [22];

- **Tempo di conferma delle transazioni**a: quando ci sono periodi di congestione, in cui la domanda di transazioni cresce, il blocco viene riempito velocemente. Quando il blocco è pieno,le transazioni in eccesso vengono lasciate in coda;

- **Commissioni elevate**: se ci troviamo nello scenario precedente con blocchi pieni, la concorrenza sulle commissioni fa aumentare il costo di ciascuna

transazione, poichè i partecipanti sono disposti a pagare di più per assicurarsi che la propria transazione venga inserita nel prossimo blocco.

Per risolvere questi problemi potremmo pensare di aumentare la dimensione dei blocchi o diminuire il tempo di creazione di un nuovo blocco. Supponendo che la rete Bitcoin cresca e che arrivi ad elaborare 40000 transazioni al secondo(tps), che sono simili alle tps supportate da Visa durante i picchi. La memorizzazione di queste informazioni localmente comporterebbe 864 GB al giorno [21]. Ciò renderebbe infattibile la gestione di un nodo per chiunque , tranne che per grandi imprese. Il risultato sarebbe una rete centralizzata dove pochi utenti possono validare le transazioni.

3.2 Introduzione a Lightning Network

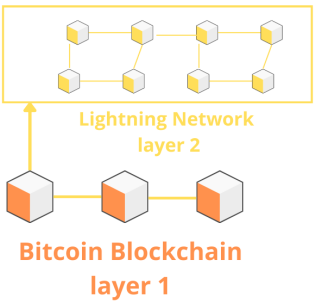

Figura 3.1: Visione generica Lightning Network

La **Lightning Network** è una rete peer-to-peer di canali di pagamento implementati come contratti intelligenti sulla blockchain di Bitcoin, nonché un protocollo di comunicazione che definisce il modo in cui i partecipanti impostano ed eseguono questi contratti intelligenti [23]. L'accesso alla rete Lightning avviene tramite applicazioni software in grado di reespattare il protocollo Lightning Network. I nodi Lightning per prima cosa sono **portafogli** e necessitano dell'accesso alla Blockchain Bitcoin. Solitamente questi portafogli si comportano sia da portafoglio Bitcoin sia da portafoglio Lightning. Per sfruttare la Lightning Network e portare i propri Bitcoin fuori catena occorre creare un **canale di pagamento**. Questo canale, che risiede al di sopra di Bitcoin, va a collegare i due nodi permettendo pagamenti senza registrare transazioni sulla Blockchain Bitcoin. Per aprire un canale si va a selezionare un nodo e si seleziona la quantità di Bitcoin che si vuole portare in questo. L'apertura del canale non è un operazione gratuita poichè si

va a costruire una transazione speciale chiamata **transazione di finanziamento** (**funding transaction**). Questa viene inviata alla rete Blockchain e si dovrà attendere la registrazione(circa sei conferme). Una volta confermata, il canale è aperto e pronto per eseguire pagamenti verso il nodo selezionato e verso i nodi connessi a quest'ultimo. Il canale aperto non ha un tempo definito per la chiusura. Anzi, l'obiettivo è quello di lasciare aperto il canale il più a lungo possibile in modo da evitare ripetute aperture poichè apertura e chiusura richiedono una transazione sulla blockchain Bitcoin e quindi una spesa extra. Tuttavia, delle volte è necessario chiudere un canale, per esempio quando il partner non risponde più o perchè si desidera riportare il saldo su Bitcoin. Ci sono tre modi per chiudere il canale:

- Chiusura reciproca;
- Chiusura Forzata;
- Violazione del protocollo.

Esploreremo queste opzioni nelle prossime sezioni.

3.3 Come funziona Lightning Network?

3.3.1 Architettura Lightning Network

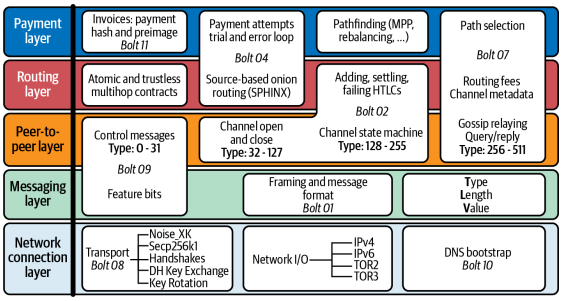

Figura 3.2: Architettura protoccolo Lightning Network [24]

La Lightning Network è composta da una complessa raccolta di protocolli che funzionano su Internet. A grandi linee possiamo distinguire questi protocolli in cinque livelli che costituiscono uno stack di protocolli in cui ogni livello utilizza i protocolli del livello sottostante. I livelli sono i seguenti [25]:

- **Livello di connessione di rete**: contiene i protocolli che interagiscono direttamente con i protocolli Internet core (TCP/IP), i protocolli overlay (Tor v2/v3) e i servizi Internet (DNS).

- **messaggistica**: questo livello contiene i protocolli utilizzati dai nodi per negoziare funzionalità, formattare messaggi e codificare i campi dei messaggi.
- **Livello peer-to-peer (P2P)**: questo livello è il livello di protocollo primario per la comunicazione tra i nodi Lightning e contiene tutti i diversi messaggi scambiati tra i nodi.
- **Livello di instradamento**: questo livello contiene i protocolli utilizzati per instradare i pagamenti tra i nodi, end-to-end e atomicamente. Questo livello contiene la funzionalità principale di Lightning Network: i pagamenti instradati.
- **Livello di pagamento**: lo strato più alto della rete, che presenta un'interfaccia di pagamento affidabile per le applicazioni.

3.3.2 Canali di Pagamento

Come affermato in "Mastering Lightning Network" [26], un **canale di pagamento** è una relazione finanziara tra due nodi Lightning. Alla base del canale di pagamento c'è un indirizzo **multifirma 2 su 2**. Solitamente quando qualcuno "possiede" Bitcoin significa che conosce la chiave privata di un indirizzo Bitcoin e quindi ha alcuni output di transazioni non spesi. Ma Bitcoin ha anche indirizzi **multifirma** dove sono necessarie più chiavi private prima di spendere.
Prima che due nodi aprano un canale devono stabilire una connessione internet. Ogni nodo quando viene creato genera una **chiave privata root**. Da essa, si va a generare una **chiave pubblica** del nodo che identifica univocamente esso sulla Lightning Network. Inoltre, ogni nodo pubblicizza anche un idirizzo di rete dove può essere raggiunto nel formato **TCP/IP** o **TCP/Tor**. L'identificatore è scritto come **indirizzo:porta** [26]. Quindi affinchè i due nodi possano connettersi si avrà bisogno della chiave pubblica dell'altro nodo o la chiave pubblica, indirizzo IP o Tor e porta. L'apertura vera e propria del canale si ottiene attraverso lo scambio di messaggi tra i nodi.

Apertura canale

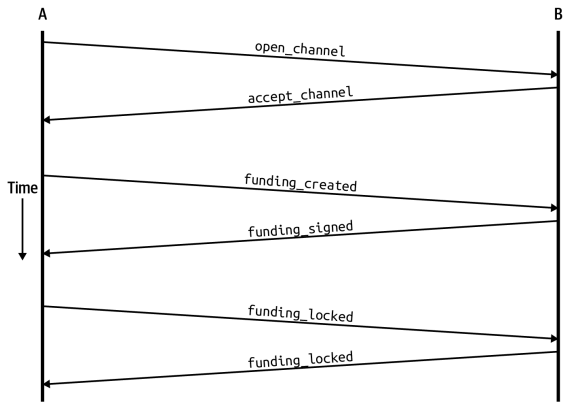

Figura 3.3: Messaggi Apertura Canale Lightning Network [27]

Ora che i nodi Lightning sono collegati si può iniziare il processo di apertura di un canale di pagamento. La creazione del canale si ottiene mediante lo scambio di sei messaggi tra i nodi. I messaggi scambiati sono [26]:

1. **open_channel**
2. **accept_channel**
3. **funding_created**
4. **funding_signed**
5. **funding_locked**

Supponiamo che un nodo A voglia aprire un canale con un nodo B. Il nodo A dovrà inviare un messaggio open_channel che contiene delle "aspettative" per l'impostazione del canale, che il nodo B può accettare o rifiutare. I campi contenuti nel messaggio open_channel specificano i parametri del canale che il nodo A desidera. Alcuni dei campi più importanti sono [26]:

- **funding_satoshis**: l'importo che il nodo A utilizzerà per finanziare il canale;
- **channel_reserve_satoshis**: il saldo minimo, in satoshi, riservato su ciascun lato del canale;
- **push_msat**: un campo facoltativo utilizzato dal nodo A per dare fondi al nodo B;

- **to_self_delay**: campo di sicurezza importante che esiste per consentire a ciascuna parte di esprimere per quanto tempo l'altra parte debba attendere per richiedere unilateralmente i fondi in una transazione di impegno. Ossia, se il nodo B decide di chiudere il canale contro la volontà del nodo B, attraverso questo campo si indica l'impegno di non accedere ai propri fondi per il tempo definito in questo campo.

In risposta a questo campo il nodo B restituisce il messaggio accept_channel per accettare o rifiutare il canale. I campi più importanti sono:

- **funding_pubkey**: la chiave pubblica del nodo di Bob contribuisce all'indirizzo multisig 2 su 2 che **ancora** il canale.
- **minimum_depth**: il numero di conferme che il nodo di Bob si aspetta per la transazione di finanziamento prima di considerare il canale "aperto" e pronto all'uso.

A questo punto il nodo A può costruire la **Transazione di finanziamento (funding transaction)** che va ad ancorare il canale alla blockchain Bitcoin.

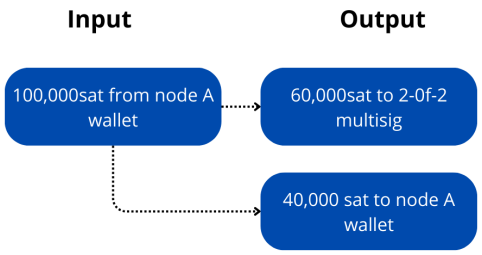

Figura 3.4: Funding Transaction

Come detto più volte, un canale di pagamento è ancorato ad un indirizzo multifirma 2 su 2. Quindi la prima operazione consiste nel generare questo indirizzo che ci permetterà di costruire la transazione di finanziamento. Il ruolo della transazione di finanziamento è fondamentalmente quello di inviare un certo numero di Bitcoin, che sono quelli specificati in **founding_satoshi** nel messaggio openChannel. Generato l'indirizzo multifirma è rischioso immettere i propri fondi in questo. L'inidirizzo multifirma richiede più firme per sbloccare i fondi, in questo caso la firma dei due nodi. Se l'altro nodo decide di non firmare i fondi sono bloccati e non più spendibili. Per evitare che ciò accada, il nodo A deve creare una nuova transazione,

una transazione di rimborso, che gli permetta di recuperare i fondi anche se il nodo B non collabora o scompare. Questa è la **Transazione di Rimborso (Refund Transaction)**.

Transazione di Rimborso(Refund Transaction)

Una **Transazione di Impegno** non ha il solo scopo di proteggere inizialmente un nodo, ma verrà utilizzata anche per aggiornare lo **stato** del canale e quindi i fondi dei due partecipanti. Questa transazione garantisce che i partner non debbano fidarsi l'uno dell'altro. Quello che si fa è una sorta di **accordo prematrimoniale**, dove i partner, prima di sposarsi, firmano un accordo che specifca come separare i beni in caso di separazione. La prima transazione di Impegno che si crea è detta **Transazione di Rimborso(Refund Transaction)**. Questa viene creata immediatamente dopo aver costruito la transazione di finanziamento, **ma non trasmessa**. La transazione di rimborso serve per **spendere** il multisig 2-of-2.
Ma come è possibile spendere un output che non è stato ancora confermato sulla blockchain Bitcoin?
Il tutto è reso possibile grazie ad una funzionalità di bitcoin chiamata **Segregated Witness**, discussa precedentemente. Quando decidiamo di creare un canale di pagamento abbiamo bisogno di una transazione di finanziamento che richiede un multisig 2-of-2. Questa include un output che verrà speso in futuro. Anche se la

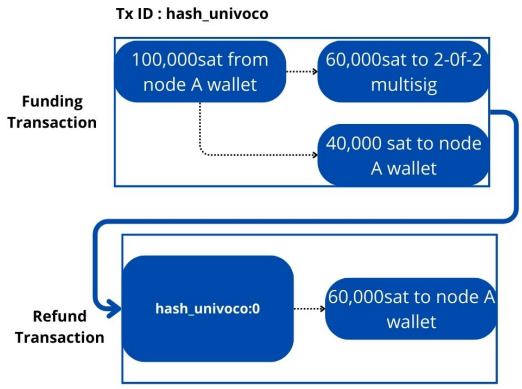

Figura 3.5: Refund Transaction

transazione non è stata trasmessa alla Blockchain Bitcoin, è possibile calcolarne l'hash che verrà utilizzato come identificativo univoco, quindi è possibile far riferimento ad essa anche se questa non è stata ancora confermata. Da questa è possibile creare una Transazione di Rimborso che va a spendere l'output della transazione di finanziamento. Come è possibile vedere nella **Figura 3.5**, abbiamo un **hash univoco** che va a referenziare la Funding Transaction creata dal nodo A anche

se non è stata ancora trasmessa alla Blockchain Bitcoin. In cascata è possibile creare la Refund Transaction che va a spendere l'output:0 della transazione di finanziamento. Affinchè la transazione di rimborso diventi valida, devono verificarsi due cose [26] :

- la transazione di finanziamento deve essere trasmessa alla rete Bitcoin.
- l'inserimento della transazione di rimborso richiede le firme dei due partner.

Ora che la Transazione di Finanziamento è pronta, continua lo scambio dei messaggi tra i due nodi. Il messaggio **funding_created** può ora essere creato dal nodo A. In questo ci sono informazioni importanti sulla transazione di finanziamento, come [26] :

- **funding_txid**: rappresenta l'id della transazione di finanziamento e verrà utilizzato per creare l'id del canale.
- **funding_output_txid**: rappresenta l'indice di output, serve all'altro nodo per sapere l'output multisig 2-of-2.

Inoltre, il nodo A invia anche la firma utilizzata per spendere dal multisig 2 of 2. Questa è necessaria poichè anche l'altro nodo dovrà creare la propria versione di una transazione di impegno. Il nodo B che riceve questo messaggio dovrà inviare la sua firma attraverso il messaggio **funding_signed**. Ora che il nodo A dispone delle firme necessarie per la transazione di rimborso può trasmettere la transazione di finanziamento alla blockchain Bitcoin senza timore di avere i suoi fondi bloccati. Per l'apertura del canale bisogna solo aspettare il numero minime di conferme, **minimum_depth** impostato nel messaggio **open_channel**. non appena questo numero di conferme viene raggiunto, I due nodi inviano il messaggio **funding_Locked** e il canale è pronto per l'uso.

Pagamenti all'interno del canale

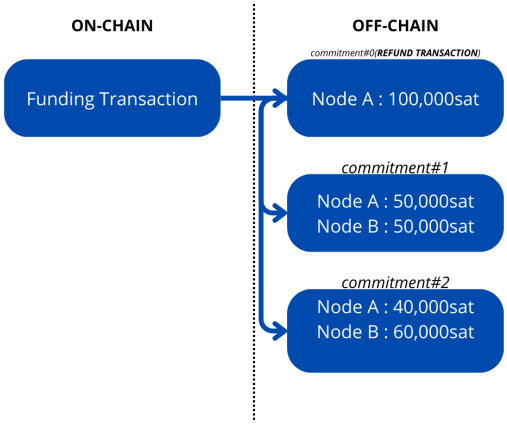

Figura 3.6: Transazioni di Impegno Multiple

Una volta che il canale è stato aperto è possibile inviare pagamenti tra i due nodi. I nodi hanno il compito di far avanzare lo stato del canale attraverso le **Transazione di Impegno**. Ogni transazione aggiorna i saldi dei nodi per rifletteri i pagamenti che sono stati effettuati. Tutto ciò che bisogna fare è creare una transazione che spende multisig 2 of 2 su due output. Seguendo questo schema, ci ritroveremo con **Transazioni di impegno multiple** firmate e validate che possono essere utilizzate da entrambi i partner in qualsiasi momento per chiudere il canale. Il problema è che solo l'ultima Transazione di Impegno riflette il reale saldo del canale. Se non si utilizza una strategia diversa uno dei nodi può imbrogliare trasmettendo una vecchia transazione con saldo a suo vantaggio. Non è possibile annullare, fermare o censurare una transazione una volta trasmessa sulla rete Bitcoin. Quindi non è possibile controllare la capacità di trasmettere una transazione. La soluzione utilizzata è un **meccanismo di penalità**, dove un partecipante può sempre trasmettere una transazione, anche se non è l'ultima, ma se lo fa probabilmente perde tutto il denaro. Sono tre gli elementi che permettono di utilizzare questa strategia [26]:

- **spesa ritardata**;
- **chiavi di revoca**;
- **transazioni di Impegno asimmetriche**;

Nelle **transazioni di impegno asimmetriche** i nodi detengono due transazioni :

- **toSelf** : transazione che può essere spesa con le chiavi del nodo(nodo A);

- **toRemote** : transazione che può essere spesa con le chiavi dell'altro nodo(nodo B);

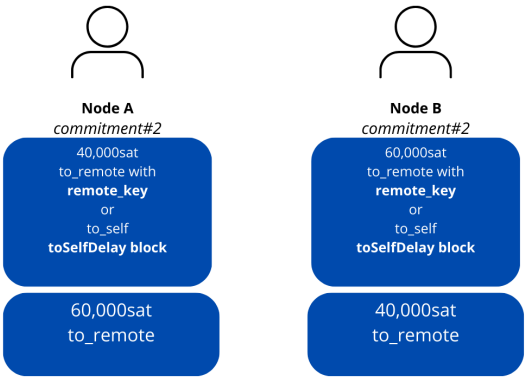

Figura 3.7: Transazioni di Impegno Asimmetriche, ritardate e revocabili [28]

Grazie a queste possiamo attribuire facilmente la colpa alla parte che imbroglia. Questo perchè la parte toLocal è bloccata nel tempo e non può essere spesa immediatamente mentre la parte toRemote non è bloccato è può essere spesa immediatamente. Per fare questo, la parte onesta deve avere il tempo di confutare la revoca dei propri fondi. Ad esempio, toLocal è bloccata per un certo numero di blocchi, espresso dal campo to_self_delay nel messaggio open_channel. Quindi se il nodo A chiude il canale trasmettendo la sua transazione di Impegno in suo possesso, non può spendere il suo saldo per un certo numeri di blocchi ma il nodo B può reclamare il suo saldo immediatamente. Il **ritardo** esiste per consentire alla parte remota di esercitare la penalità nel caso dovesse essere trasmesso un vecchio impegno. L'opzione di penalità è applicata attraverso la **chiave di revoca**. L'output toLocal non sarà bloccato solo nel tempo ma ha anche una condizione di spesa che può essere attivata dal nodo remoto con la chiave di revoca.

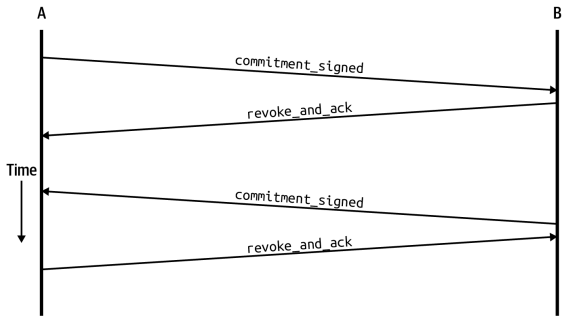

Figura 3.8: Flusso messaggi impegno e revoca [28]

Per aggiornare lo stato del canale le parti coinvolte devono scambiarsi due messaggi che sono:

- **commitmentSigned**;
- **revokeAndAck**.

Con il primo messaggio, il nodo A, dà al nodo B i mezzi per creare una nuova Transazione di Impegno. Il nodo B può revocare la vecchia transazione e fornire al nodo B un **per_commitment_secret**. Quest'ultimo viene utilizzato per costruire la **chiave di revoca** per la vecchia transazione, che consente di pignorare tutti i fondi dell'altro nodo andando ad esercitare la penale.

Chiusura canale

La chiusura dei canali richiede una transazione sulla blockchain Bitcoin, una spesa aggiuntiva, per cui sarebbe meglio non chiuderli. Ma, come già accennato, la chiusura, alcune volte, è inevitabile. Un modo per chiudere il canale è la **chiusura**

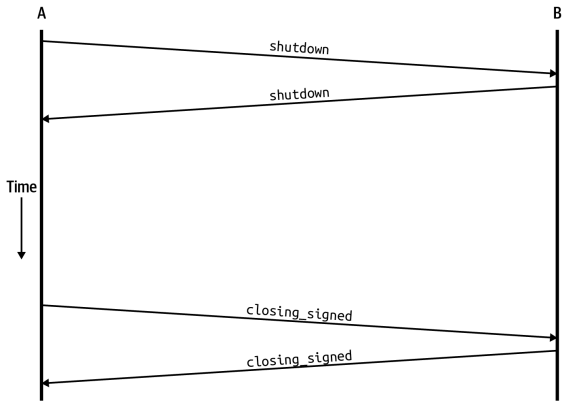

Figura 3.9: Messaggi Chiusura Canale Lightning Network [29]

reciproca. Avviene quando entrambi i nodi accettano di chiudere il canale. I due partner di canale negoziano una transazione di impegno finale chiamata **transazione di chiusura** che paga immediatamente a ciascuna parte il saldo presente sul canale di pagamento. I messaggi che vengono scambiati in questo caso sono:

- **shutdown**;
- **closing signed**.

Uno dei due nodi può decidere di inviare il messaggio shutdown. Il nodo che invia il messaggio chiede all'altro nodo di fare una transazione di chiusura che paghi il saldo del canale su un determinato portafoglio specificato in uno dei suoi campi. L'altro nodo farà lo stesso. Una volta fatto questo si può inviare il messaggio closing signed. Questo messaggio propone una commissione per la transazione on-chain e la firma del nodo (il multisig 2 su 2) per la transazione di chiusura. Se l'altro nodo è d'accordo può rispondere con un messaggio closing signed anche lui, altrimenti propone una tariffa diversa.

3.3.3 Introduzione al Routing

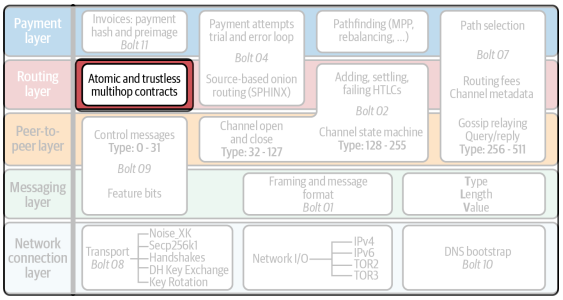

Figura 3.10: Istradamento pagamenti Atomici in Lightning Network [30]

Il **Routing** rappresenta le interazioni attraverso la rete che si verificano per inoltrare un pagamento da un punto A ad un punto B. Il percorso sarà selezionato attraverso il **pathfinding**. Ciò che si vuole raggiungere con il routing [31] :

- **Atomicità**: il pagamento viene eseguito oppure fallisce;
- **Operazioni senza fiducia**: i partecipanti non devono fidarsi l'uno dell'altro;
- **Multihop**: la sicurezza si enstende su tutti i canali coinvolti.

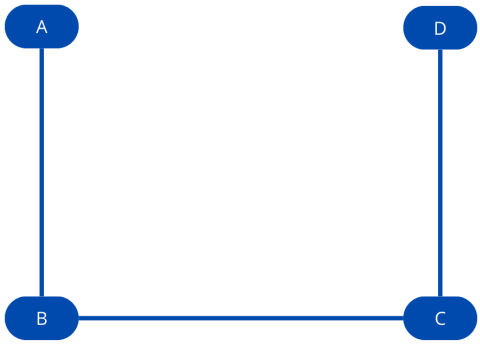

Figura 3.11: Collegamento Indiretto Nodo A e Nodo D

La Figura 3.11 mostra una possibile configurazione di rete Lightning. In questa rete, anche se tutti i nodi non sono direttamente connessi tra di loro(come il Nodo A e Nodo D), tutti possono instradare pagamenti verso tutti. Il Nodo A e il Nodo D, pur non essendo direttamente connessi, possono inviare pagamenti tra di loro. Consideriamo il caso, in cui il Nodo A invia un pagamento al Nodo D. I Nodi lungo il percorso vengono definiti **Nodi di instradamento**. Qualsiasi nodo Lightning può istradare pagamenti. I nodi di instradamento spesso addebitano una commissione per agire come intermediari ma possono sceglierre di instradare i pagamenti anche gratuitamente.

Routing sui canali di pagamento

Riprendendo la Figura 3.11, affinchè il Nodo A possa pagare il Nodo D, il Nodo D dovrà generare una **Fattura Lightning**. Discuteremo in seguito della fattura. L'interazione tra il Nodo A e il Nodo D per la generazione della fattura avviene all'esterno della rete Lightning, ad esempio può avvenire sul Web.

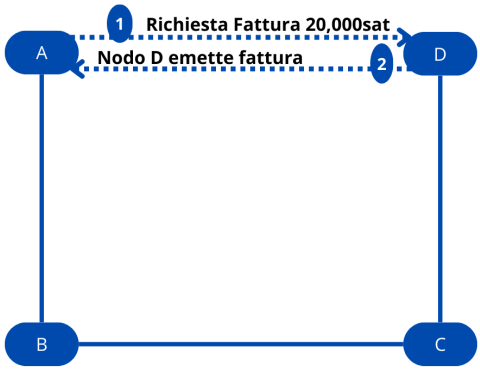

Figura 3.12: Richiesta Fattura

In Nodo A quindi dovrà richiedere una fattura, ad esempio di 20.000sat come mostrato in Figura 3.12. Il nodo D dovrà generare questa fattura e inviarla al Nodo A. Prima di pagare la fattura, il Nodo A deve trovare un percorso verso il Nodo D attraverso il **pathfinding**. Parleremo del pathfinding in seguito. Supponendo che questo sia già stato fatto, il Nodo A deve costruire un pagamento Lightning. Questo avviene Costrundo un **hash time locked Contracts(HTLC)**.

Hash Time Locked Contracts

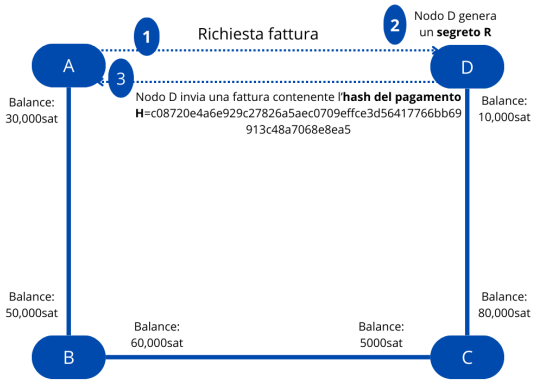

Figura 3.13: Il Nodo A riceve l'hash del pagamento dal Nodo D

Il nodo A e il Nodo D non sono connessi direttamente, quindi il pagamento che deve effettuare il Nodo A passerà attraverso il Nodo B e il Nodo C.
Come fa il Nodo A a sapere che il pagamento è stato effettuato correttamente?
In Lightning la **prova di pagamento** assume la forma di un **segreto** [31]. Nella

figura 3.13, dopo che il Nodo A richiede una fattura, il Nodo D genera un Segreto. Assumiamo per semplicità che il segreto, chiamato **preimmagine** [31], è : **R** = "Segreto del Nodo D". Nella realtà il segreto è un numero casuale molto grande. Il Nodo D non invia direttamente questo segreto al Nodo A, che conserva per lui, ma invia, tramite canali esterni non connessi a Lightning, l'Hash di R calcolato nel seguente modo:

$$\mathbf{H = SHA\text{-}256(R)}$$

chiamato l'**hash del pagamento** [31].
Il Nodo A, anche se non conosce il segreto, può costruire un **HTLC** utilizzando l'hash inviato dal Nodo D. La prima parte dell'**HTLC** è proprio H (hash del pagamento). Solo chi è a conoscenza di R (preimmagine) può riscattare il pagamento. Nel nostro esempio, solo il Nodo D può.

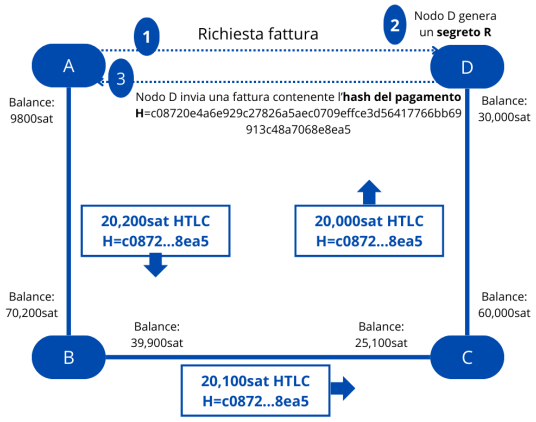

Figura 3.14: Propagazione HTLC

Il pagamento deve quindi attraversare i nodi di instradamento e poi giungere al Nodo D. Ogni canale è composto da due nodi e ogni nodo ha un saldo associato. Durante il passaggio del pagamento i fondi vengono prelevati dal canale stesso. Pertanto, il denaro effettivamente usato è quello dei nodi intermedi coinvolti. Gli intermediari non possono riscattare il pagamento perchè non conoscono il segreto ma hanno il compito di instradare l'HTLC. Come precedentemente menzionato, i nodi intermedi posso richiedere un compenso per far transitare il messaggio tra un nodo a un altro. Nella figura 3.14, il Nodo A prepara un pagamento di 20,200sat invece di 20,000sat per dare un compenso di 100sat ai nodi intermedi. È importante notare che i nodi **non possono** riscattare i pagamenti finchè il segreto non viene rilevato.

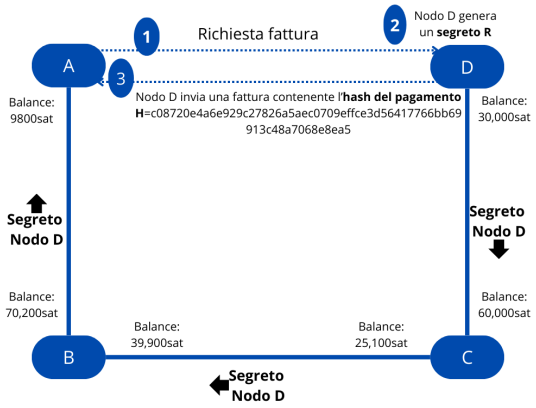

Figura 3.15: Propagazione Preimmagine

Come mostrato in Figura 3.15, una volta che l'HTLC arriva a destinazione, per riscattare il pagamento, il nodo D va a rivelare la preimmagine che ha stabilito come prova del pagamento. Questo segreto va a ritroso finchè non raggiunge il Nodo A. Il Nodo D invia il segreto al Nodo C che verifica se il segreto corrisponde all'hash fornito. Se è cosi sblocca i fondi per il Nodo D. Ora il nodo C è a conoscenza del segreto e può inviarlo al Nodo B. Anche esso verificherà la correttezza tra preimmagine e Hash e sbloccherà il pagamento per il Nodo C. Lo stesso farà il Nodo B con il Nodo A. L'uso della funzione Hash, garantisce il funzionamento di questo meccanismo **Senza Fiducia**.

Errore di Timeout

Finora abbiamo dato per scontato che tutti i nodi sono online al momento e tutti sono intenzionati a collaborare.
Cosa succede se qualche nodo è offline o non collabora?
Per evitare che i fondi restino bloccati per sempre, ogni script HTLC include una clausola collegata ad un blocco temporale. Quest'ultima' garantisce **l'atomicità**, poichè il pagamento può avere solo esito positivo o negativo. La clausola di scadenza temporale viene impostata diversamente in base al nodo coinvolto. La differenza tra i timelock per ogni hop è chiamato **cltv_expiry_delta** [31]. Per esempio il Nodo A può impostare, per il Nodo B, un blocco temporale pari a 300 Blocchi. Mentre il Nodo B può impostare, per il Nodo C, un blocco temporale pari a 250 blocchi.

Flusso di messaggi HTLC

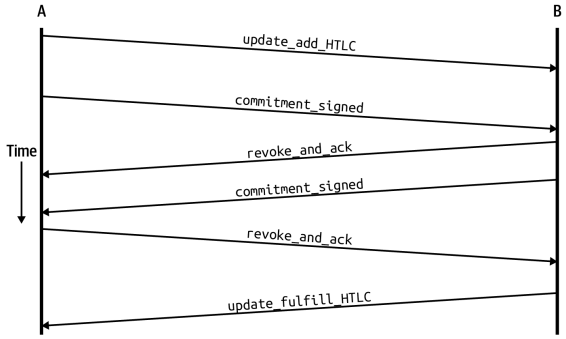

Figura 3.16: Flusso di messaggi tra i partner di canale [32]

Il ciascun canale di pagamento vengono scambiati i messaggi in Figura 3.16. Questi messaggi hanno lo scopo di aggiornare lo stato del canale e riflettere correttamente i saldi dei due partner. Riprendendo il nostro esempio, il Nodo A vuole inviare un pagamento al Nodo D. Per riuscire in questo, il Nodo A deve inviare un HTLC al Nodo B di 20,200sat. Per aggiungere un HTLC, il Nodo A avvia il flusso di messaggi inviando il primo update_add_HTLC. Quest'ultimo contiene i campi [33]:

- **channel_id**: specifica il canale da utilizzare verso il Nodo B (poichè i due nodi potrebbero avere più canali;)
- **id**: contatore HTLC, viene incrementato ogni qualvolta si aggiunge un HTLC sul canale;
- **amount_msat**: valore dell'HTLC in millisatoshi;
- **payment_hash**: hash calcolato dalla fattura del Nodo D;
- **cltv_expiry**: tempo di scadenza per questo HTLC;
- **onion_routing_packet**: percorso crittografato che dice al Nodo B dove inoltrare HTLC.

Ricevuto il messaggio, il Nodo B può creare una nuova transazione di impegno(commitment transaction) che ha gli stessi due output to_self e to_remote, menzionati in 3.3.2, e un nuovo output che rappresenta l'HTLC. Esistono tre modi per richiedere il pagamento dell'output HTLC [33]:

1. riscatto tramite **chiave di revoca**;
2. riscatto poichè la **preimmagine** è stata revocata;

3. rimborso dell'HTLC se il cltv_expiry è scaduto.

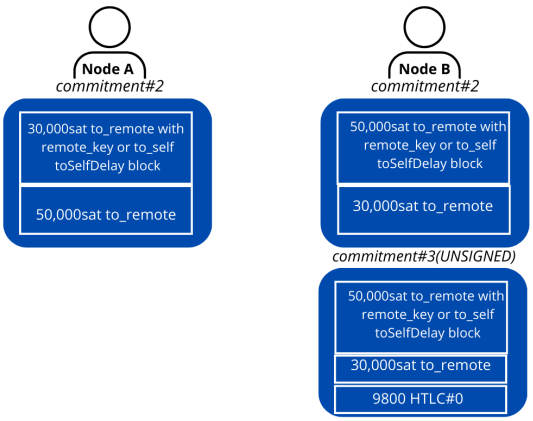

Figura 3.17: Nuovo Impegno del Nodo B con Output HTLC

Com'è possibile vedere dalla Figura 3.17, il nuovo impegno HTLC del Nodo B è pari a 9800sat (30.000 - 20,200), che rappresenta il saldo futuro di Alice se il pgamento attraverso HTLC avviene con successo. L'impegno, inizialmente, è provvisorio poichè manca la firma del Nodo A. Il Nodo A, per inviare la firma, utilizza il messaggio commitment_signed.

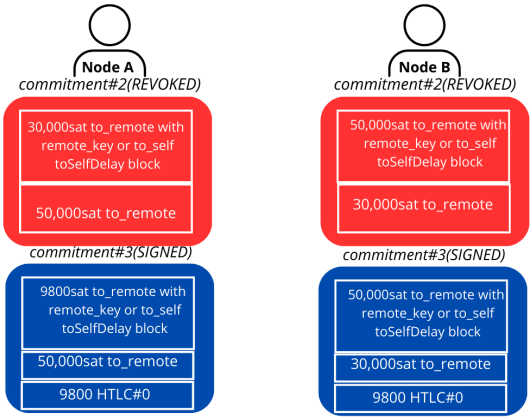

Figura 3.18: Nuovo Impegno del Nodo A e del Nodo B con Output HTLC

Ricevuta la firma del Nodo A, il Nodo B può inviare il messaggio revoke_and_ack che conferma di aver revocato la vecchia transazione. Il Nodo A esegue le stesse operazioni del Nodo B, creando il nuovo impegno e revocando il vecchio. Tutti gli altri canali conivolti eseguiranno queste operazioni, finchè non si arriverà al Nodo D che conosce la preimmagine.

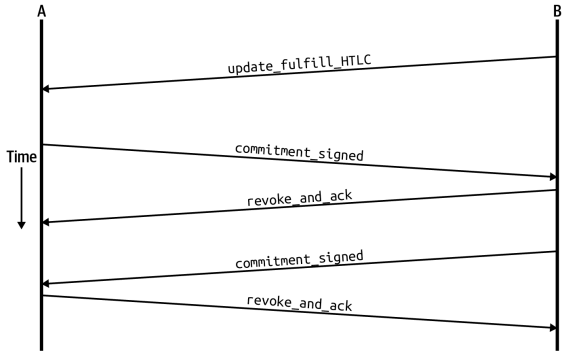

Figura 3.19: Flusso di messaggi riscatto HTLC [34]

Per dimostrare che il nodo conosce la preimmagine, si invierà un messaggio **update_fulfill_htlc**. Questo messaggio contiente il campo **payment_preimage**, che ospiterà il segreto e verrà utilizzato per riscattare il pgamento. Il segreto è utilizzato per riscattare l'HTLC e verrà passato lungo la catena. Ogni Nodo invierà al Nodo antecendente il messaggio **update_fulfill_htlc** con annesso il segreto per riscattare il pagamento.

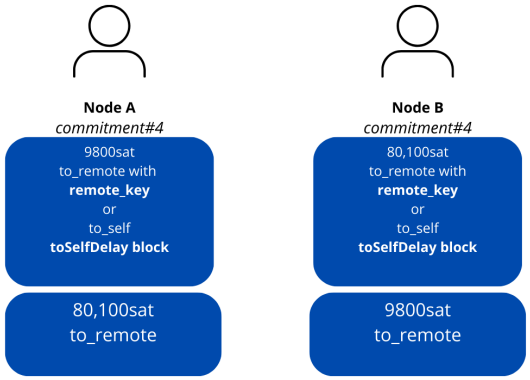

Figura 3.20: Aggiornamento Saldo Finale

Quando un nodo riceve il messaggio, generare l'hash del segreto e lo confronta con quello in suo possesso. Se la verifica va a buon fine si creare un nuovo impegno e si aggiorna definitivamente lo stato del canale, come mostrato in Figura 3.20. Nel caso che un HTLC non può essere soddisfatto, si utilizza lo stesso processo di impegno e revoca. I messaggi che vengono scambiati dai nodi in questo caso sono : **update_fail_htlc** o **update_fail_malformed_htlc**. Seguirebbe un aggiornamento dello stato del canale con un nuovo impegno. In conclusione gli HTLC nella pratica vengano utilizzati sia per i pagamenti locali, tra i due partner di canale, sia per i pagamenti remoti, quando non abbiamo una connessione diretta

con un nodo. Il processo di un pagamento locale tramite HTLC è esattamente uguale a quello spiegato. L'unica differenza è che l'HTLC viaggerà per un solo hop, per arrivare al partner di canale. Quest'ultimo, ricevuto il messaggio, invierà immediatamente il messaggio **update_fulfill_htlc** e successivamente aggiornerà lo stato.

Fattura Lightning

Nel momento della sua creazione la fattura si presenta cosi :

lntb2500u1pvjluezpp5qqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqf
qqqsyqcyq5rqwzqfqypqdq5xysxxatsyp3k7enxv4jsxqzpuaztr
nwngzn3kdzw5hydlzf03qdgm2hdq27cqv3agm2awhz5se903vruatf
hq77w3ls4evs3ch9zw97j25emudupq63nyw24cg27h2rspfj9srp

Alcune parti di questa stringa sono leggibili come **lntb** che rappresenta il prefisso che permette di identificare l'istanza di Lightning Network.
I possibili prefissi sono [35]:

- **rete principale**: lnbc
- **testnet**: lntb
- **simnet/regtest**: lnbcrt

Successivamente troviamo un numero intero come importo base seguito da un moltiplicatore ""u"" nella fattura precedente. Ma i possibili sono :

Moltiplicatore	Unità di Bitcoin	Fattore di moltiplicazione
m	milli	0.001
u	micro	0.000001
n	nano	0.000000001
p	pico	0.000000000001

Tabella 3.1: Moltiplicatori dell'importo [36]

La parte successiva, ""illegibile"", utilizza lo schema di codifica utilizzato da Bitcoin chiamato **bech32**. Questa porzione di dati può essere suddivisa in tre sezioni [35]:

- **Timestamp**;
- Zero o più **coppie chiave-valore**;
- La **firma** sull'intera fattura.

La **firma** copre l'intera fattura e consente al mittente di verificare che la richiesta sia stata effettivamente creata dal destinatario del pagamento.

3.3.4 Onion Routing

Quando il Nodo A invia un pagamento al Nodo D, i Nodi intermedi possono visionare il contenuto. La mancanza di privacy, in una rete di pagamento, è una scelta non adeguata. Lightning Network utilizza un'implementazione del protocollo onion basata su **Sphinx** [37], che ci permette di garantire privacy tra i vari nodi della rete. La tesi non si propone di esplorare nei dettagli l'onion routing. Piuttosto, forniremo una visione generale del concetto.

Esempio di Onion Routing

Il nome **Onion Routing** va a descrivere una crittografia a più livelli, dove un livello viene rimosso ad ogni hop, come uno strato di cipolla. I nodi intermedi possono solo staccare un livello e vedere qual è il prossimo percorso di comunicazione. Questo garantisce che nessuono, tranne il mittente, conoscsa la destinazione o la lunghezza del percorso. Lightning utilizza il **source routing** [37] dove il mittente è tenuto a scegliere il percorso.

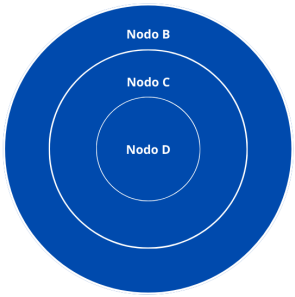

Figura 3.21: Costruzione della cipolla

Nella Figura 3.21, ogni strato rappresenta uno strato crittografico, che consente solo al destinatario di aprirla e leggerla. Il Nodo A vuole inviare un pagamento al Nodo D con intermediari il Nodo B e il Nodo C. Per costruire la cipolla, il Nodo A parte dal destinatario fino ad arrivare ad uno dei partner di canale. Nel nostro esempio è presente, come partner di canale, solo il nodo B. Quindi gli strati della cipolla sono costruiti in ordine inverso rispetto all'ordine di instradamento del pagamento.

Si inizierà creando il primo strato per il Nodo D, il secondo per il Nodo C e il terzo per il Nodo B. Questo perchè i nodi intermedi rimuovano il proprio strato della cipolla in ordine inverso.

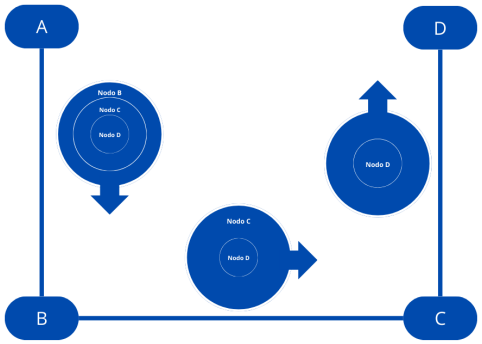

Figura 3.22: Esempio Onion Routing

Nella Figura 3.22, il Nodo B riceva la cipolla dal Nodo A. Sa che proviene dal Nodo A ma non sa se è il mittente o un nodo intermedio. Il Nodo B, rimosso lo strato della cipolla, trova un altro strato che dice ”al Nodo C”. Il Nodo B non può togliere questo strato poichè solo il Nodo C può con le sue chiavi. Quando la cipolla arriva al Nodo D, sa di essere il destinatario del messaggio poichè non trova altri strati. Il Nodo D può utilizzare il segreto per ricevere il pagamento. Quando il nodo c riceve il segreto non può sapere se il Nodo D è il destinatario del pagamento o un nodo intermedio che ha ricevuto il segreto.

3.3.5 Gossip and Channel Graph

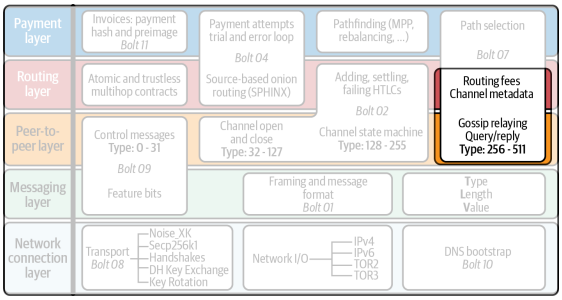

Figura 3.23: Protocollo gossip in Lightning Network

Come affermato in ”Mastering Bitcoin” [38], il protocollo **Gossip** viene utilizzato per distribuire informazioni pubbliche sui canali a tutti i partecipanti. I partner di canale possono accettare di annunciare il proprio canale all’interno di Lightning Network, rendendo cosi il **canale pubblico**. Annunciare pubblicamente i canali consente ad altri nodi di utilizzarli per l’instradamento dei pagamenti, generando

così anche commissioni di instradamento per i partner del canale. Al contrario si può scegliere anche di creare un **canale privato** scegliendo di non annunciarlo. Quando un canale e la sua capacità vengono annunciati pubblicamente utilizzando il protocollo gossip, l'annuncio può anche includere informazioni sul canale (metadati), come le tariffe di instradamento e la durata del blocco temporale.

Routing vs Pathfinding

I pagamenti su Lightning Network vengono inoltrati lungo un percorso fatto di canali che collegano un partecipante all'altro, dalla fonte del pagamento alla destinazione del pagamento. Il processo di ricerca di un percorso dall'origine alla destinazione è chiamato **pathfinding**. Mentre il processo di utilizzo del percorso trovato è chiamato **Routing**. Le informazioni sul saldo di tutti i canali non sono e non possono essere conosciute da tutti i partecipanti alla rete. Questo comporta il non poter calcolare facilmente il percorso. La strategia attualmente utilizzata è: i nodi Lightning provarno iterativamente a cercare un percorso finchè non si raggiunge il successo o non si trova. Quando si invia un pagamento al nostro partner di canale il percorso è banale. In tutti gli altri casi, si utilizza il protocollo Gossip per eseguire il pathfinding [38]. Grazie al protoccolo gossip possiamo tener traccia dei canali di pagamento pubblici attualmente conosciuti, la topologia di rete, la capacità totale dei canali noti e anche le politiche tariffarie.

Capitolo 4

Simulatore Infrastrutture di Ricarica

4.1 Introduzione alla Simulazione

La simulazione è una tecnica di modellazione che va ad imitare il comportamento di un sistema reale nel tempo, eseguendo esperimenti su un modello realizzato al computer. La simulazione viene adottata per valutare il comportamento di un sistema quando non è possibile utilizzare un sistema reale. In questa tesi esploreremo due simulatori. Il primo punta a simulare l'intero ciclo di vita di un veicolo che ha bisogno di ricaricare, che inizia dal dirigersi verso l'infrastruttura elettrica per finire con il pagamento. Il secondo, propone una strategia di ricarica intelligente, esplorato in "Smart electric vehicle charging for reducing photovoltaic energy curtailment" [39], basato sulla definizione dei pesi fuzzy. Gli strumenti utilizzati sono gli stessi e verranno descritti in "Architettura del Software" 4.2. Per presentare la simulazione, inizieremo presentando l'interfaccia, poichè con questa gli utenti solitamente si interfacciano. Tuttavia, l'interfaccia è solo una piccola parte dell'applicazione, un punto di connessione con il sistema sottostante. Man mano che avanziamo con la simulazione, ci sposteremo verso il backend, che possiamo considerarlo il cervello della nostra applicazione, colui che getisce i calcoli, si occupa di comunicare con la Blockchain Quorum e di interfacciarsi con il protoccolo Lightning Network per aprire i canali di pagamento, per emettere fatture e per effettuare i pagamenti. Quindi, esploreremo, contemporaneamente, interfaccia e backend per comprende appieno il funzionameto della simulazione. Poichè nella seconda simulazione il processo di pagamento è simile alla prima simulzione, se non per la presenza dell'hub, quest'ultimo verrà presentato solo nella prima simulazione.

4.1.1 Come funzionano i simulatori

4.1.2 Primo simulatore

In questa simulazione andremo a modellare l'intero ciclo di vita di automobilisti che decidono, in parallelo, di ricaricare il loro veicolo elettrico. Il primo step nella simulazione è creare i canali Lightning Network tra hub e colonnine, se non esistono già. In realtà è possibile creare più di un canle tra due partner. Ma per semplicità si è scelto di avere solo un canale aperto. Inizialmente verranno scambiati dei messaggi tra i veicoli che serviranno per calcolare in quale infrastruttura di ricarica dirigersi, e saranno salvati sulla Blockchain **Quorum**. I messaggi saranno dati in input ad un algoritmo specifico, ricevermo come risposta il percorso migliore. Attraverso i dati forniti all'algoritmo, è possibile capire dove e in quale colonnina il veicolo andrà a ricaricare, il tempo impiegato per ricaricare, il tempo di attesa prima di ricaricare e altri. Quando un veicolo inizia a dirigersi verso la colonnina elettrica, l'algoritmo inizia a monitorarlo. Una volta che i veicoli sono avviati verso la colonnina possono ritrovarsi in tre fasi:

- **Go** : il veicolo si sta muovendo per raggiungere la colonnina;
- **Wait** : il veicolo è in attesa poichè la colonnina è occupata;
- **Charging** : il veicolo è alla colonnina e sta ricaricando.

Quando il veicolo è in carica, ogni minuto, viene registrata una transazione sulla blockchain Quorum che memorizza l'avanzamento della carica del veicolo. Giunti al termine della ricarica, avviene il pagamento attraverso Lightning Network. Il veicolo, se non ha un canale aperto con l'hub lo crea. Una volta completato il processo di apertura, si passa al pagamento. Il nodo della colonnina crea una fattura con l'importo da pagare e la invia al veicolo. Il veicolo, ricevuta la fattura può procedere al pagamento. Terminato il pagamento lascia spazio al prossimo veicolo in coda. In fine, è stata realizzata un interfaccia che va a mostrare come avviene lo spostamento del veicolo nelle tre fasi descritte precedentemente, riassume i dati memorizzati su Quorum e i pagamenti effettuati su Lightning Network.

4.1.3 Secondo simulatore

A differenza del primo simulatore dove ogni colonnina può ospitare un solo veicolo avendo un solo **plug**, in questa simulazione ci interfacceremo con una solo colonnina che ha a disposizione sei plug e che può quindi ospitare sei veicoli. Come affermato in "Infrastrutture per veicoli elettrici"1.1, un fattore molte importante è la **potenza totale di ricarica** che un'infrastruttura può fornire ad ogni veicolo che si collega ad essa. Per assicurare una gestione efficace della ricarica adotteremo una strategia

basata sul **Fuzzy weights**. Lo scopo della simulazione non è capire come vengono calcolati i pesi, se si è interessati al calcolo di questi è spiegato in "Smart electric vehicle charging for reducing photovoltaic energy curtailment"[39]. Per lo scopo della tesi, è sufficiente sapere che ad ogni veicolo è associato un **Fuzzy weight (peso Fuzzy)**. Quando i veicoli sono in ricarica, per ogni minuto, si controlla il Fuzzy di ciascuno di esso. Se quest'ultimo è superiore alla media dei fuzzy dei veicoli che stanno ricaricando ed è superiore ad una soglia definita, il veicolo guadagna un profitto sotto forma di aumento di potenza erogata da parte della colonnina a cui è collegato. In questo caso, si genera una transazione, salvata in modo permanente sulla Blockchain Quorum per garantire la piena trasparenza e tracciabilità delle operazioni di ricarica. Come nella precedente simulazione, quando viene conclusa la ricarica, un veicolo effettua il pagamento attraverso Lightning Network. Non ci sono sostanziali differenze rispetto la precedente simulazione, tranne per il fatto che i veicoli aprono il canale direttamente con la colonnina e quindi non hanno spese di commissione.

4.2 Architettura del Software

Un aspetto fondamentale dello sviluppo di qualsiasi sistema softare è **l'Architettura del sistema**. Essa fornisce una visione ad alto livello della struttura e del funzionamento del Software.

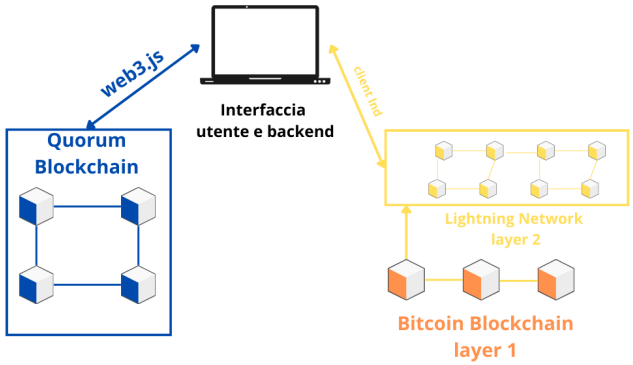

Figura 4.1: Architettura del sistema

Per comprendere il funzionamento del sistema, è essenziale esaminare la l'architettura che facilita le interazioni tra i vari componenti, Figura 4.1. Il sistema è

suddiviso in diverse componenti interconnesse, ognuna delle quali svolge un ruolo cruciale:

1. **Interfaccia Utente**: fornisce agli utenti un'interfaccia per avviare e monitorare le transazioni, sia su Quorum che su Lightning Network, oltre a visualizzare le diverse fasi della simulazione. L'interfaccia utente è stata realizzata attraverso React.js insieme alla libreria React-Bootstrap che fornisce un set completo di elementi UI pronti all'uso;

2. **Backend**: costituisce il cuore del sistema e gestisce tutte le logiche. È responsabile di fornire all'interfaccia utente i dati da visualizzare e interagisce con i nodi Quorum e Lightning Network. Il backend è stato sviluppato utilizzando Next.js, un framework JavaScript basato su Node.js, che offre funzionalità avanzate per applicazioni web performanti;

3. **Blockchain Quorum**: il backend comunica con la blockchain Quorum per la registrazione di transazioni di ricarica, gestione dei contratti intelligenti relativi alla ricarica dei veicoli. Per comunicare con Quorum, utilizziamo **web3.js** una libreria JavaScript.

4. **Lightning Network**: il backend comunica con la rete Lightning attraverso il client **LND (Lightning Network daemon)** tramite **gRPC (Remote Procedure Calls)**. Attraverso le API fornite da LND possiamo gestire l'apertura del canale, creazione di fatture e invii di pagamenti.

In questa sezione, esamineremo i vari software utilizzati nel contesto di questo progetto.

4.2.1 Node.js

Node.js [40] è un ambiente runtime di JavaScript multipiattaforma open source. Questo consente agli sviluppatori di eseguire codice JavaScript (js) oltre che lato browsers anche lato Server. Grazie a Node.js, è possibile utilizzare un solo linguaggio di programmazione, lato client e lato server, per creare applicazioni web. Node.js è costruito su un modello di I/O non bloccante e basato su eventi il che consente alle applicazioni di gestire un gran numero di connessioni simultanee senza bloccarsi. Questo lo rende adatto ad applicazioni ad alta concorrenza.

4.2.2 React-Bootstrap

React [41] è una libreria JavaScript open source. Viene utilizzata per costruire (**interfacce utente(User Interface)**) dinamiche e reattive. React permette agli sviluppatori di suddividere l'interfaccia utente in tanti componenti che sono

indipendenti e riutilizzabili. Questo permette di svilupparli e testarli separatamente in modo da ridurre la complessità del progetto. Per ottimizzare le prestazioni dell'applicazione web si utilizza il **Virtual Dom**, che è una rappresentazione dell'interfaccia utente, memorizzata in memoria. Grazie al Virtual Dom React può aggiornare l'interfaccia in modo efficiente, modificando solo le parti che sono realmente cambiate. React gode di un vasto ecosistema di librerie e strumenti di supporto. Ciò include, una vasta gamma di componenti UI disponibili attraverso librerie come **React-Bootstrap** [42]. Questo essenzialmente va ad offrire dei componenti React pre-costruiti, rendendoli facili da integrare nelle applicazioni.

4.2.3 Next.js

Next.js [43] è un framwork di sviluppo web basato su React e utilizzato per la creazione di applicazioni web veloci e scalabili. Caratteristica principale di Next.js è il rendering ibrido:

- **SSR (Server side Rendering)**: con approccio SSR, la generazione del contenuto HTML avviene lato server prima che venga inviato al Client;
- **CSR (Client-side Rendering)**: con l'approccio CSR, l'intera applicazione web viene caricata nel browser del client. Una volta caricata, il JavaScript viene eseguito nel browser del client per creare e aggiornare l'interfaccia utente.

Possiamo dire che Next.js va ad estendere le funzionalità di React aggiungendo delle caratteristiche, come il rendering lato server(SSR). Inoltre fornisce un sistema per le rotte integrato.

4.2.4 web3.js

Quando sentiamo parlare di **Web3** ci riferiamo ad una tipologia di web che punta alla **decentralizzazione**. **Web3.js** è una collezione di librerie che permettono l'interazione con un nodo Ethereum locale o remoto usando HTTP, IPC o WebSocket [44]. Utilizzando Web3.js, è possibile interagire con i contratti intelligenti Ethereum direttamente dal codice JavaScript. Per prima cosa, dobbiamo interagire con la Blockchain. Per interagire con **Quroum**, è possibile utilizzare uno o più nodi come **provider**. Questi hanno il ruolo di interfacciarsi con la blockchain. Web3.js mette a disposizione il metodo **WebSocketProvider** che si connette ad un nodo, nel nostro caso un nodo membro, attraverso un URL. Fatto questo, è possibile creare una nuova istanza **Web3** utilizzando il WebSocketProvider appena creato [45]. Visto che la simulazione avviene in maniera totalmente automatica è possibile usufruire di Web3.js per semplificare la gestione delle chiavi private e per firmare le transazioni all'interno delle applicazione in maniera automatica attraverso l'uso del modulo

Wallet. Questo può gestire più account, ognuno con la propria chiave privata e indirizzo [46]. Per interagire con il contratto si è utilizzato **web3.eth.Contract** [47]. Per creare un'istanza di questo oggetto sono richiesti :

- L'interfaccia Json dello Smart Contract;
- L'indirizzo del contratto.

Questi elementi vengono creati quando avviene il deploy dello Smart Contract sulla Blockchain. Appena creata un'istanza del contratto è possibile selezionare il metodo da chiamare attraverso **methods** e inviare una transazione, che la blockchain, eseguirà attraverso **send()** di web3.js. Quando si utilizza questo metodo si deve specificare chi sta inviando la transazione e opzionalmente l'importo massimo, in **gas**, che si vuole spendere per questa. Successivamente, è possibile sfruttare la callback di send() per gestire il risultato della transazione.

4.2.5 Truffle-Solidity

Web3.js. come detto, ci permette di comunicare con i nodi andando ad interagire con gli smart contract. Per implementare gli smart contract ci affideremo a due strumenti essenziali che sono **Truffle** [48] e **Solidity** [49]. Solidity è un linguaggio di programmazione ad alto livello orientato agli oggetti, progettato per la scrittura di smart contract su Ethereum e altre piattaforme compatibili come **Quorum**. Truffle è un framework di sviluppo Ethereum che semplifica la creazione, il test e la distribuzione di contratti intelligenti. Truffle è progettato per gestire l'intero flusso di lavoro dei developer blockchain.

4.2.6 Quorum

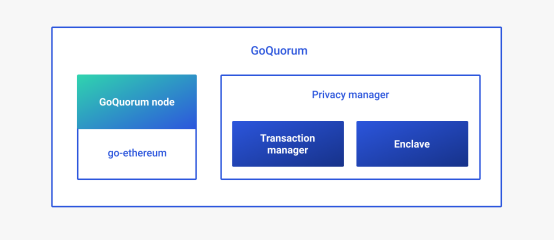

Figura 4.2: Architettura GoQuorum [50]

Abbiamo già esplorato come utilizzare web3.js per interagire con i nostri contratti e come è possibile creare contratti utilizzando Solidity. Tuttavia un aspetto che resta da affrontare è la creazione di una blockchain locale dedicata ai test. Esistono

diversi strumenti per realizzare una blockchain di test. Noi utilizzeremo Quorum che è una piattaforma blockchain enterprise open-source basata su Ethereum.
Con **blockchain enterprise** si riferisce ad una blochchain implementata e gestita per soddisfare esigenze specifiche aziendali. Rispetto alle blockchain pubbliche, le blockchain enterprise spesso presentano caratteristiche e requisiti diversi. GoQuorum è una versione leggera di geth, che è un client software basato sul linguaggio di programmazione Go (Golang) e rappresenta una delle implementazioni principali del protocollo Ethereum. In sostanza, Geth è un nodo software che consente la partecipazione alla rete Ethereum e facilita l'interazione con essa.
Volendo fare un confronto tra **Quorum** e **Ethereum**, una delle principali caratteristiche è la capacità di garantire **privacy** alle transazioni, introducendo smart contract privati e consente di mantenere riservate le informazioni sensibili all'interno della blockchain. Per quanto riguarda l'algoritmo di consenso, Quorum offre la flessibilità di scegliere tra diversi algoritmo in base alle esegigenze della rete. Nel nostro progetto utilizzeremo il **QBFT (Quorum Byzantine Fault Tolerance)** [51] che condivide alcuni similitudini con l'algoritmo **Proof of Stake** di Ethereum ma anche sostanziali differenze. Entrambi sono algoritmi di consenso distribuito utilizzati per aggiungere un accordo sulla validità delle transazioni e sulla modifica dello stato della Blockchain. Sono pensati per tollerare eventuali guasti dei nodi e i nodi partecipano per validare le transazioni. Una differenza sostanziale è che il QBFT si basa su un modello più centralizzato, dove i nodi validatori sono selezionati e sono responsabili per l'accettazione delle transazioni. Mentre nel PoS i nodi validatori vengono selezionati casualmente in base alla quantità di criptovaluta posseduta(**staking**).
Per impostazione predefinita, GoQuorum è una **rete gas gratuita** [52], il che significa che il gas ha un prezzo pari a zero e non viene consumato durante l'elaborazione delle transazioni. Tuttavia, il prezzo del gas può essere abilitato se necessario. Le transazioni utilizzano risorse computazionali, quindi hanno costi associati. Il gas è l'unità di costo e il prezzo del gas è il prezzo per unità di gas. Il costo della transazione è il gas utilizzato moltiplicato per il prezzo del gas. Se il gas è abilitato, allora come nelle reti Ethereum pubbliche, l'account che invia la transazione paga il costo della transazione, in Ether. Il minatore (o il validatore, nelle reti di proof of authority) che include la transazione in un blocco riceve il costo della transazione. Ci sono tre tipi principali di nodi:

- **Nodo Validatore**: la sua funzione principale è convalidare le transazioni, un processo critico che comporta la partecipazione al consenso per garantire che le transazioni siano legittime. Dopo la convalida, il nodo contribuisce ad aggiungere nuovi blocchi alla blockchain, ognuno contenente un insieme di transazioni convalidate. È inoltre responsabile del mantenimento dello stato attuale della blockchain, comprese informazioni come i saldi degli account e lo stato dei contratti intelligenti.

- **Nodi RPC (Remote Procedure Call)**: servono come interfaccia di comunicazione remota. Essenzialmente, agiscono come un mezzo attraverso il quale le applicazioni o gli sviluppatori possono interagire con un nodo blockchain da una posizione esterna. i nodi RPC supportano portafogli digitali e interfacce utente, consentendo agli utenti di visualizzare e interagire con le loro informazioni sulla blockchain.
- **Nodi Bootstrap**: agiscono come "punti di riferimento" centrali a cui i nuovi nodi possono rivolgersi per informazioni sulla rete e per essere presentati ad altri nodi. Questo processo semplifica significativamente la scoperta iniziale dei nodi, specialmente per coloro che si uniscono alla rete per la prima volta.

4.2.7 Polar-LND

Per velocizzare la creazione dell'ambiente di test per Lightning Network si è scelto di utilizzare **Lightning Polar**[53]. Questa fornisce un interfaccia facile da utilizzare per configurare il nostro ambiente Lightning Network. Grazie a Polar, possiamo creare tutte le reti che vogliamo velocemnte. Per creare una rete avremo sempre bisogno di almeno un nodo **Bitcoin Core**. Questo perchè, il protoccolo Lichtning Network richiede una connessione alla rete Bitcoin per funzionare correttamente. Nello specifico, per aprire un canale di pagamento è necessario effettuare una transazione sulla catena Bitcoin. Questo si traduce nell'avere dei fondi in un portafoglio Bitcoin. La nostra rete nello specifico, oltre al nodo Bitcoin Core, avrà diveri nodi Lightning. Nella prima simulazione sono presenti: tre nodi Lightning che rappresentano le colonnine, nove nodi Lightning che rappresentano i veicoli elettrici e un nodo Lightning che rappresenta l'**hub**. L'hub è stato pensato come un nodo che facilita la connessione tra i veicoli e le colonnine. È un punto centrale che instaura un canale con tutte le colonnine esisteni e con tutti i veicoli così che un veicolo non debba creare un canle per ogni colonnina ma basta un unico canale, con l'hub, che funge da intermediario per inoltrare pagamenti verso le colonnine. Polar supporta tre diversi software che permettono di gestire nodi Lightning. Per la simulazione, si è scelto **LND (Lightning Network Daemon)**, un software open-source che consente agli utenti di creare e gestire i nodi Lightning.
LND è in grado di [54]:

- **Gestire canali di pagamento**: apertura di un canale, chiusura, gestione completa di tutti gli stati del canale;
- **Istradamento**: effettua automaticamente la ricerca del percorso all'interno del canale;
- **offrire Api**: LND offre una serie di API che consentono di creare applicazioni che interagiscano con la rete Lightning.

- **Compatibilità con diverse piattaforme**;

Inoltre LND è progettato per essere eseguito su varie piattaforme, compresi Linux, macOs e Windows. Polar inoltre, per ogni nodo, mostra informazioni comuni sulla connessione RPC (remote procedure call), che permettono di connettere i nodi alle applicazioni.

4.3 Design del Software - Primo strumento di simulazione

4.3.1 Blockchain Quorum

La blockchain Quorum utilizzata nella prima simulazione è organizzata nel seguente modo:

- **Bootnode**: il bootnode fornisce un punto di accesso iniziale alla rete Quorum. Quando un nodo si unisce alla rete, si connette attraverso il bootnode che fornisce informazioni sulla topologia della rete.
- **Tre Validatori**: rappresentano le colonnine utilizzate nella simulazione. I validatori sono i nodi che partecipano al processo di consenso e validano le transazioni sulla blockchain. Quando vi è una nuova transazione, i Validatori collaborano per verificare la validità e raggiungere un accordo sullo stato della blockchain.
- **Nove Membri**: rappresentano i veicoli elettrici. I membri sono nodi che partecipano alla rete ma non sono coinvolti nel processo di consenso o validazione delle transazioni.

4.3.2 Analisi backend

```
1
2     let acc = createConnection(
3         process.env.TOTAL_NODES
4     );
5
6
7     let client_HUB = await lndClient(process.env["
    LND_ADMINMACAROON_HUB"],
8         process.env["LND_TLSCERT_HUB"],
9         process.env["LND_LOCALHOST_HUB"]
10    );
11
12    newChannelAcceptor(client_HUB);
13
14    let client_LND_CP = [];
15
16
17
18    for (let i = 0; i < 3; i++) {
19        let client = await lndClient(process.env["
    LND_ADMINMACAROON_CHARGINGPOINT_" + (i + 1)],
20            process.env["LND_TLSCERT_CHARGINGPOINT_" + (i + 1)],
21            process.env["LND_LOCALHOST_CHARGINGPOINT_" + (i + 1)]
22        );
23        client_LND_CP.push(client);
24    }
25
26    await createChannelHubCP(client_HUB);
```

Listing 4.1: Collegamento Quorum e creazione canale Lightning Network

Prima di iniziare l'analisi vera e propria abbiamo bisogno di stabilire una connessione con la blockchain Quorum e con i nodi Lightning. Per semplificare la gestione del codice, gli indirizzi e altre informazioni utili vengono salvate come variabili d'ambiente. Ad esempio, process.env["LND_LOCALHOST_CHARGINGPOINT_" + (i + 1)] è un espressione per memorizzare la porta del nodo Lightning. Per stabilire una connessione con la blockchain Quorum utilizziamo la funzione **CreateConnection**, Listing 4.1.

```
1
2 export function createConnection(n_node) {
3     let provider = new Web3.providers.WebsocketProvider(
4       process.env["ETHEREUM_NODE_WS_URL"]
5     );
6     const instance = new Web3(provider);
7
8     for (let i = 1; i <= n_node ; i++) {
```

```
9       if (process.env["PUBLIC_ADDRESS_" + i] == chargingPoint) {
10        continue;
11      } else {
12
13        instance.eth.accounts.wallet.add({
14          privateKey: process.env["WALLET_MNEMONIC_" + i],
15          address: process.env["PUBLIC_ADDRESS_" + i],
16        });
17      }
18    }
19
20
21    const contract = new instance.eth.Contract(
22      Contract.abi,
23      process.env["CONTRACT_ADDRESS"]
24    );
25
26
27    return { web3: instance, contract };
28  }
```

Listing 4.2: Collegamento Quorum

La funzione createConnection, Listing 4.2, ci consente di connetre il backend ad un nodo e di creare un istanza Web3. Per semplificare la gestione delle chiavi private e per firmare le transazioni usiamo il modulo Wallet, che gestisce queste operazioni in modo automatico. L'ultimo step è quello di istanziare un oggetto contratto per interagire con il contratto salvato sulla blockchain. Prima di aprire un canale di pagamento dobbiamo stabilire una connessione con i nodi LND. Come mostrato in 4.1, chiamiamo la funzione **lndClient** che ha bisogno di una connessione **TLS/SSL** e di un **macaroon** che è utilizzato per l'autenticazione RPC. Stabilita la connessione è possibile interagire con il nodo LND. Inizialmente si stabilisce una connessione con l'hub e con le colonnine, successivamente, durante la fase di pagamento, si eseguirà lo stesso processo anche per i veicoli.

```
1     async function newChannelAcceptor(client) {
2     const call = await client.channelAcceptor({});
3
4     channelAcceptorPromise(client, call)
5     .then(()=>{
6         console.log('Tutti i dati sono stati elaborati e la
    chiamata è completata ACCEPT.');
7     })
8     .catch((error)=> {
9         console.error('Si è verificato un errore durante la
    chiamata ACCEPT:', error);
10    });
11    console.log("FINE");
```

```
12      }
13
14      async function channelAcceptorPromise(client, call) {
15      return new Promise((resolve, reject) => {
16
17          call.on('data', function (response) {
18
19              console.log("CHANNEL ACCEPTOR: ", response);
20              let request = {
21                  accept: true,
22                  pending_chan_id: response.pending_chan_id,
23
24                  reserve_sat : 550,
25                  min_accept_depth : 1,
26                  zero_conf: false
27              };
28              call.write(request);
29          });
30          call.on('status', function (status) {
31
32              console.log("STATUS CHANNEL ACCEPTOR: ", status)
33          });
34          call.on('end', function () {
35
36              console.log('Chiamata completata.ACCEPTOR');
37              resolve();
38          });
39
40          call.on('error', function (error) {
41              console.error('Errore nella chiamata CA:', error);
42          });
43
44      });
45      }
46
```

Listing 4.3: channelAcceptor

L'hub, stabilita la connessione, definisce con quali canali stabilire la connessione e con quali no attraverso la funzione **newChannelAcceptor** che richiama l'RPC channelAcceptor di LND che consiste in uno streaming bidirezionale, Listing 4.3. Quando arriva una richiesta di connessione del canale, il client avvisa il nodo della richiesta, attraverso la callback **call.on**. Il nodo, presa visione della richiesta, decide se accettare o meno. Per creare un canele tra le colonnine e l'Hub viene richiamata la funzione **createChannelHubCP**, Listing 4.1. Questa funzione ciclicamente, in base al numero delle colonnine, richiama la funzione **newChannel** per la creazione del canale. La funzione fa un controllo se colonnina e hub hanno già un canale. Nel

caso di risposta positiva non ne crea uno nuovo, altrimenti si richiama la funzione **createChannel**, Listing 4.4.

```
1     async function createChannel(client, to_client, is_hub) {
2     let request = {
3         node_pubkey: Buffer.from(to_client, 'hex'),
4         local_funding_amount: is_hub ? 16000000 : 8000000,
5         push_sat: is_hub ? 3000 : 1000,
6         target_conf: 5,
7         private: false,
8         min_confs: 1,
9         spend_unconfirmed: false,
10        zero_conf: false,
11        scid_alias: false,
12        remote_chan_reserve_sat: 500,
13        fund_max: false,
14    }
15
16    let call = await client.openChannel(request);
17
18 }
```

Listing 4.4: openChannel

La funzione **createChannel** ha il compito di creare e successivamente inviare una richiesta di apertura canale attravere l'RPC openChannel di LND. Per inviare la richiesta bisogna specificare il nodo con cui si desidera aprire un canale, i fondi che si vogliono portare dalla blockchain bitcoin, il numero di conferme per la **Funding Transaction** e altro. Dopo aver stabilito una connessione con la blockchain Quorum e aver creato i canali tra hub e collonina, i veicoli iniziano a scambiare i messaggi per calcolare il percorso migliore.

```
1     for (let i = 0; i < data.length; i++) {
2         listenMessage(acc, i).then(()=>{
3             console.log("Fine Ascolto")
4         }).catch((error) => {
5             console.error('Errore durante l\'ascolto dei messaggi:
    ', error);
6         })
7     }
8
9     for (let i = 0; i < data.length; i++) {
10        await eth_start_message(acc, i);
11
12    }
13
14    for (let i = 0; i < data.length; i++) {
15        await eth_message(acc, data[i]);
16
```

```
17      }
```

Listing 4.5: Scambio di messaggi

La funzione **listenMessage** permette ad un nodo di mettersi in ascolto su un evento di interesse, nella simulazione di un messaggio, in modo essere notificato.

```
1       async function listenMessage(acc, id){
2       return new Promise((resolve, reject) => {
3           acc.contract.events.FirstMessageEvent({filter : {recipient
    : process.env["PUBLIC_ADDRESS_" + (id + 1)]}})
4               .on('data', (event) => {
5                   console.log('Messaggio ricevuto:', event);
6               })
7               .on('error', (error) => {
8                   reject(error);
9               });
10      });
11  }
```

Listing 4.6: listenMessage

Per fare ciò utilizziamo l'oggetto **event** e specifichiamo l'evento di interesse, Listing 4.6. Con la funzione **eth_start_message** si invia il messaggio agli altri automobilisti che verrà salvato sulla blockchain.

```
1       async function eth_start_message(acc, id) {
2       let numeroStazione = generaNumeroCasuale(1, 3);
3
4       let capacitaBatteria = generaNumeroCasuale(170, 350);
5
6       let statoCaricaVeicolo = generaNumeroCasuale(10, 80);
7
8       await acc.contract.methods
9           .firstMessageF(recipients.filter(item => item.address !==
    process.env["PUBLIC_ADDRESS_" + (id + 1)]).map(x => x.address),
10          id, numeroStazione, capacitaBatteria.toString(),
    statoCaricaVeicolo.toString())
11          .send({
12              from: process.env["PUBLIC_ADDRESS_" + (id + 1)],
13              gas: 500000000
14          })
15          .on("transactionHash", async (hash) => {
16              console.log("hash : ", hash);
17          })
18          .on('receipt', async (receipt) => {
19              //SAVE_DATA_IN_DB_MESSAGE
20              console.log("risp : ", receipt);
21          })
22          .on("error", async (error) => {
```

```
23              console.log("ERROR UPDATE CHARGING: ", error)
24          });
25  }
```

Listing 4.7: firstMessageF

Come mostrato in Listing 4.7, inizialmente vengono generati casualmante dei dati con lo scopo di assistere il veicolo nel calcolo della stazione a cui dirigersi. I dati verranno salvati sulla blockchain Quorum richiamando la funzione **firstMessageF**.

```
1
2     event FirstMessageEvent(address indexed sender, address[]
       recipients, FirstMessage message);
3
4
5       function firstMessageF(address[] memory recipients, uint8 _id,
        uint8 _stationId, string calldata _batteryCapacity,
6       string calldata _soc) external{
7
8           FirstMessage memory message = FirstMessage({
9             id : _id,
10            station_id : _stationId,
11            battery_capacity : _batteryCapacity,
12            soc : _soc
13          });
14          allMessages.push(message);
15          emit FirstMessageEvent(msg.sender, recipients, message);
16        }
```

Listing 4.8: openChannel

La funzione firstMessageF 4.8 in Solidity, riceve una seria di parametri che rappresentano:

- **recipients**: i destinatari del messaggio;
- **__id**: l'ide del veicolo che invia il messaggio;
- **__stationId**: la stazione verso cui il veicolo vuole dirigersi;
- **__batteryCapacity**: la capacità della batteria;
- **__soc**: e lo stato di carica del veicolo.

Le informazioni verranno salvate sulla blockchain Quorum attraverso la variabile di stato **allMessage**. Successivamente si emette un evento che contiene le informazioni appena discusse. Questo evento è progettato per essere osservato dagli altri veicoli come mostrato in Listing 4.6. Le altre funzioni Quorum utilizzate nella tesi non verrano presentate poichè simili a quest'ultima, Listing 4.8. La differenza

è nelle informazioni che verranno salvate e nell'evento emesso. I dati registrati vengono forniti a un algoritmo specifico che esegue una simulazione completa. Dopo aver completato la simulazione, l'algoritmo fornisce i dati ottenuti e si procede ad eseguire nuovamente la simulazione integrando la blockchain. La funzione **eth_message**, Listing 4.5, ha l'obiettivo di scambiare le informazioni ottenute, attraverso l'algoritmo, tra i veicoli e memorizzarle sulla blockchain.

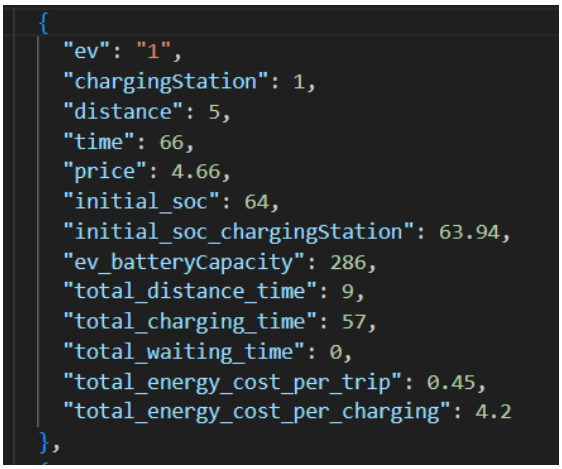

Figura 4.3: Informazioni ricevute

Le informazioni sono quelle mostrate in Figura 4.3.

```
1
2
3       while (true) {
4
5       for (let i = 0; i < data.length; i++) {
6
7           if (j < data[i].time) {
8               if (j >= data[i].total_distance_time + data[i].
    total_waiting_time) {
9
10
11                  if (j == data[i].total_distance_time + data[i].
    total_waiting_time) {
12
13                      startChargingMessage(data[i], j);
14                  }
15
16
17                  await eth_updateCharging(acc, data[i], j);
18
19              } else {
20
```

```
21                  if (data[i].total_waiting_time != 0 && j == data[i
    ].total_distance_time + 1) {
22
23
24                      waitCharging(data[i], j);
25                  }
26
27
28                  if (j == 0) {
29
30                      startMessage(data[i], j);
31                  }
32              }
33
34          } else {
35              if (!is_finish(data[i].ev)) {
36                  await payments(data[i], client_LND_CP);
37                  endMessage(data[i], j);
38                  k++;
39              }
40          }
41
42
43      }
44      j++;
45      if (k > (data.length - 1)) {
46          break;
47      }
48
49  }
50
51
```

Listing 4.9: Riproduzione simulazione

Per riprodurre la simulazione, eseguiamo un loop simultaneo per ogni veicolo elettrico. Ad ogni ciclo, che corrisponde ad un intervallo di tempo di un minuto, controlliamo, che il tempo totale di ricarica **Time** nella Figura 4.3, non venga superato. In base alla distanza dalla colonnina e al tempo di attesa, il veicolo si troverà in una determinata fase, Listing 4.9. Ad sempio, se la somma del tempo totale di attesa e il tempo necessario per arrivare alla colonnina è uguale al tempo registrato tramite la variabile j, allora il veicolo si trova nella fase di ricarica. In questo caso viene chiamata la funzione **startChargingMessage** che ha il compito di salvare lo stato del veicolo corrente su un file Json. Questo processo è necessario per aggiornare l'interfaccia. Quando il veicolo è in fase di carica, viene anche chiamata la funzione **eth_updateCharging** per emettere una nuova transazione sulla blockchain Quorum. Questa transazione serve per memorizzare

l'aggiornamento della carica del veicolo sulla blockchain. Quando il tempo totale è maggiore rispetto al tempo registrato, in veicolo effettuerà il pagamento attraverso la funzione **payments** utilizzando il protocollo Lightning Networks.

```
1
2       async function payments(data, client_LND_CP) {
3       let client = await lndClient(process.env["
        LND_ADMINMACAROON_EV_" + data.ev],
4           process.env["LND_TLSCERT_EV_" + data.ev],
5           process.env["LND_LOCALHOST_EV_" + data.ev]
6       );
7
8       let value_to_pay_IN_SATOSHI = changeValue(data.price);
9
10      await newChannel(client, process.env["LND_PUBKEY_HUB"], false)
        ;
11
12
13      let invoice = await newInvoice(client_LND_CP[data.
        chargingStation],
14          value_to_pay_IN_SATOSHI);
15
16
17      newSubscribeInvoice(client_LND_CP[data.chargingStation]);
18
19
20      let payment = await newPayment(client,
21          process.env["LND_PUBKEY_CHARGINGPOINT_" + (data.
        chargingStation + 1)],
22          value_to_pay_IN_SATOSHI, invoice.payment_request,
23          invoice.payment_addr, invoice.r_hash);
24  }
```

Listing 4.10: Payments

Come mostrato in Listing 4.10, si verifica che il veicolo abbia un canale di pagamento con l'hub. Se così non è, bisogna creare un canale. Successivamente si converte il valore che il veicolo deve pagare da euro a satoshi e la colonnina crea una nuova fattura.

```
1       async function newInvoice(client, val) {
2       let gen = generate();
3       let request = {
4           memo: "Invoice for ev. Pay : " + val,
5           r_preimage: gen.r_preimage,
6           r_hash: Buffer.from(gen.r_hash, 'hex') ,
7           value: val,
8       }
9
10
```

```
11      let invoice = await addInvoice(client, request);
12
13      return invoice;
14  }
```

Listing 4.11: Riproduzione simulazione

Per creare la nuova fattura bisogna creare prima la preimmagine e poi ricavare da questa l'hash 4.11. Fatto ciò si utilizza l'RPC **addInvoice** di LND per crearla. Una volta creata, il veicolo può procedere al pagamento dopo aver ricevuto i parametri **payment_request** e **payment_addr**, chiamando la funzione **newPayment**.

```
1       async function newPayment(client, _dest, _amount,
        _payment_request, _payment_addr) {
2       let request = {
3           dest: Buffer.from(_dest, 'hex'),
4           amt: _amount,
5           payment_request: _payment_request,
6           payment_addr: _payment_addr
7
8       }
9
10      return await sendPaymentSync(client, request);
11  }
```

Listing 4.12: Riproduzione simulazione

Per procedere al pagamento è sufficiente creare la richiesta con i campi precedentemente forniti e inviarla attraverso l'RPC sendPaymentSync di LND, Listing 4.12. Una volta effettuato il pagamento, il veicolo lascia spazio al successivo in coda di attesa.

4.3.3 Interfaccia Utente

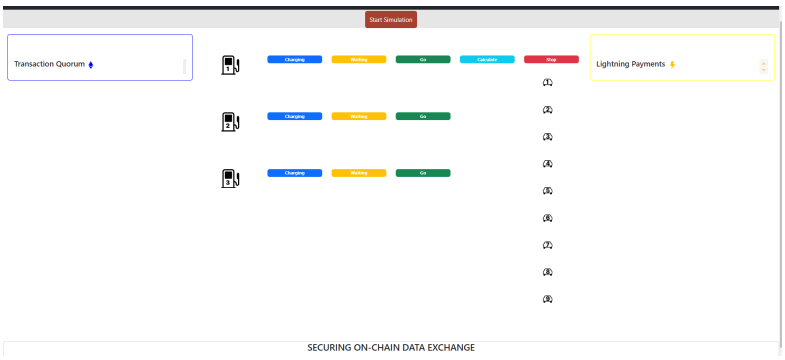

Figura 4.4: Interfaccia Utente - Prima Simulazione

L'interfaccia utente, che permette di avviare la simulazione, è rappresentata in Figura 4.4. Per avviare la simulazione è sufficieste premere il pulsante **Start Simulation**. L'interfaccia è divisa in quattro segmenti dove :

- **Transaction Quorum** :

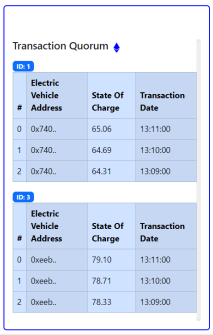

Figura 4.5: Transazioni Quorum

Situata nella sezione di sinistra, verrano visualizzate le transazioni, salvate sul blockchain Quorum, basato sullo stato di carica di un veicolo. Per ogni veicolo, verrà creata una tabella che mostra, l'indirizzo pubblico , l'avanzamento del SoC e l'ora in cui viene registrata la transazione.

- **Stato della simulazione** :

Figura 4.6: Fase di scmabio messaggio tra i veicoli

Situata nella sezione centrale, vi sono le fasi che deve seguire il veicolo prima che avvenga la ricarica. Inizialmente, quando la simulazione non è ancora iniziata, i veicoli sono in fase di **Stop**. Appena la simulazione inizia, i veicoli

entrano nella fase si **Calculating**, Figura 4.6, dove verranno scambiati dei messaggi tra i veicoli e verrà presa la decisione in quale colonnina dirigersi.

Figura 4.7: Possibile stato di un veicolo

Finita questa fase, il veicolo può trovarsi nella fase di :

- **Go** : il veicolo è diretto verso la colonnina, impiegherà del tempo prima di arrivare;
- **Waiting** : il veicolo è in attesa, poichè la colonnina è occupata da un altro veicolo;
- **Charging** : il veicolo sta ricaricando.

- **Lightning Payments** :

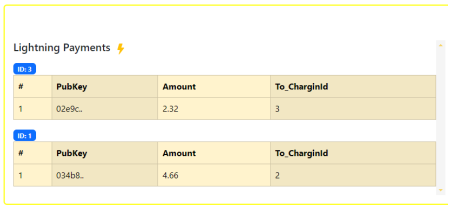

Figura 4.8: Lightning Payments

Nella sezione di destra, veranno mostrati i pagamenti effettuati da un veicolo.

- **Securing On-Chain Data Exchange** :

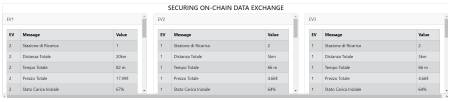

Figura 4.9: Securing On-Chain Data Exchange

Nella sezione in basso, sono presenti i messaggi scambiati da veicoli.

4.3.4 Smart Contract ChargingUpdate

Sulla blockchain Quorum è stato sviluppato e distribuito il contratto intelligente **ChargingUpdate**. ChargingUpdate ha tre compiti principali :

1. memorizzare le colonnine che faranno parte della simulazione;
2. registrare, nella prima fase della simulazione, lo scambio di messaggi tra i veicoli;
3. registrare per ogni minuto i dati relativi alla ricarica dei veicoli elettrici.

Il contratto è stato sviluppato utilizzando **Solidity** sul framework **Truffle**. Il primo step della simulazione è quello di memorizzare le colonnine nella blockchain, per poi in futuro fare riferimento ad esse. Per memorizzare le colonnine viene utilizzata la funzione **newChargingPoint**.

Figura 4.10: Nuovo messaggio scambiato

Successivamente nella fase **Calculating**, i veicoli scambiano messaggi tra di loro per calcolare il percorso migliore. Ogni qualvolta un veicolo manda un messaggio ad un altro veicolo compare il simbolo di Ethereum al suo fianco. Questo rappresenta che una nuova transazione Ethereum è stata salvata nella Blockchain Quorum, come mostrato in 4.10. Nello specifico, viene chiamata la funzione Ethereum **newFirstMessageF** che ha lo scopo di inviare un messaggio agli altri veicoli con alcune informazioni rilevanti che ci permetteranno di calcolare, una volta che ogni veicolo ha inviato il messaggio, in quale stazione il veicolo dovrà dirigersi. Il contenuto del messaggio ha le seguenti informazioni sul veicolo:

- **ID del veicolo**: numero da uno a nove per distinguere il veicolo;

- **Stazione di preferenza**: in questa simulazione sono presenti tre stazioni di ricarica, quindi una di queste;
- **Capacità Batteria** ;
- **Stato di carica del veicolo**.

I parametri sono tutti calcolati casualmente attraverso la funzione **Math.random()**.

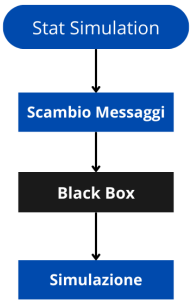

Figura 4.11: Dati per la simulazione

Una volta che lo scambio di messaggi tra i veicoli sarà ultimato, andrà in un algoritmo specifico che effettuerà una simulazione completa. L'algoritmo in questione, nella data in cui scrivo la tesi (Marzo 2024), non è ancora pubblico. Nella tesi tratteremo questo algoritmo come una black box e ci affideremo ai dati forniti per replicare la simulazione ma con l'aggiunta della Blockchain. Una volta ricevuti i dati per svolgere la simulazione, questi verranno scambiati tra i veicoli e memorizzati sulla blockchain attraverso le funzioni :

- **newMessageF**, contente informazioni relative a dove effettivamente andrà a ricaricare :
  - **ID Stazione**;
  - **Distanza dalla stazione**;
  - **Tempo totale**;
  - **Prezzo totale**;
  - **Stato di carica iniziale**;
  - **Capacità della batteria**.
- **newMessageChargingF**, contente informazioni relative alla ricarica e al consumo di energia :

- **Stato di Carica arrivato alla stazione**;
- **Tempo totale per la ricarica**;
- **Tempo totale di attesa**;
- **Stato di carica iniziale**;
- **Costo totale dell'energia per il viaggio**.

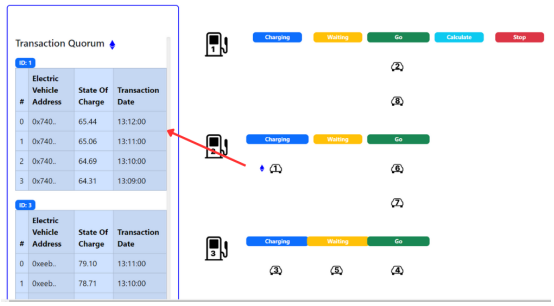

Figura 4.12: Nuova Transazione Quorum - Prima Simulazione

Ricevuti questi dati si riproduce la simulazione salvando transazioni sull'avanzamento dello stato di carica, e effettuando il pagamento attraverso il protocollo Lightning. Il veicolo sta ricaricando quando è nella fase **Charging**. In questa fase si vuole tener traccia della carica del veicolo. Per fare questo, dopo ogni minuto di ricarica si richiama la funzione **newChargingUpdate**. Lo scopo della funzione è memorizzare, dopo ogni minuto di ricarica, la nuova carica del veicolo (**SoC**). Appena viene memorizzata la transazione, questa viene inserita nel segemtno transaction Quorum, nella tabella relativa al veicolo in questione. Per calcolare l'aumento del SoC del veicolo viene utilizzata la seguente equazione :

$$\text{soc}(t+1) = \text{soc}(t) + \eta \int \left(\frac{\Delta I \cdot T}{E} \right) dt \tag{4.1}$$

L'equazione 4.1 rappresenta il modo in cui lo stato di carica di un veicolo cambia nel tempo a causa della corrente che fluisce in esso, considerando l'intervallo di tempo e l'energia totale del veicolo. Il tutto viene moltiplicato per un parametro di efficenza che tiene conto della perdite di energia di una batteria durante il processo di carica.

4.3.5 Pagamenti su Lightning Network

Creazione del canale in modo automatico

Quando un veicolo conclude il suo processo di carica dovrà effettuare il pagamento. Per effettuare un pagamento il veicolo ha bisogno di un canale Lightning. Il nostro

Figura 4.13: Rete iniziale Lightning Network

punto di partenza è Polar che fornisce un'interfaccia facile da utilizzare per configurare il nostro ambiente Lightning. Come è possibile vedere nella 4.13, la nostra rete iniziale è formata da 13 nodi LND. Tre di questi (Frank, Kathye e Lucy) rappresentano le infrastrutture elettriche. Uno di questi (Mike) rappresenta l'hub, il punto di connessione che va a connnettere le infrastrutture e i veicoli elettrici. I restanti nodi LND rappresentano i veicoli. Nello scenario fornito, i nodi non sono connessi tra di loro. Per effettuare una richiesta **gRPC** ad un **lnd(Lightning Network Daemon)** sono necessarie: una connessione **TLS/SSL** e un **macaroon** che è utilizzato per l'autenticazione RPC. Quando creiamo un'istanza LND, attraverso Polar, automaticamente si generano due file. Questi sono :

- **tls.cert**: che contiene un certificato TLS. Viene usato per garantire la siccurezza delle comunicazioni su internet.
- **admin.macaroon**: è un file utilizzato per il protocollo gRPC. Questo è un token che include informazioni riguardanti permessi e restrizioni di accesso. Molto simile ad un Cookie, ma a differenza di questo può essere usato per convalidare crittofraficamente l'emittente senza l'accesso a un database centrale.

Inoltre avremo bisogno del file **lightning.proto**, che permette di scambiare messaggi da un client e server in modo che entrambi possono comunicare pur basandosi su linguaggi di programmazione diversi. Avendo questi è possibile effettuare una richiesta gRPC a un'istanza LND. La creazione del canale Lightning Network, tra le colonnine e l'hub, avviene all'inizio della simulazione attraverso la chiamata gRPC **openChannel** [55]. Questa ha l'obiettivo di creare un canale tra due nodi. Nello

specifico, nella simulazione, l'hub invia la richiesta alle colonnine. Si è scelta questa soluzione poichè l'hub ha tutto l'interesse nel creare nuovi canali con le colonnine poichè può guadagnare attraverso le commissioni fungendo da canle intermedio. Le colonnine, d'altro canto, hanno interesse ad essere facilmente raggiungibili. Per inviare il messaggio **openChannel**, devono essere specificati alcuni campi. Alcuni di questi sono :

- **node_pubkey**: indirizzo pubblico del nodo con cui si vuole creare un canale;
- **local_funding_amount**: la quantità di satoshi che vogliamo portare sul canale Lighting. Nella simulazione, l'hub pusherà 16000000sat per ognuno dei tre canali che andrà ad aprire con le colonnine.
- **private**: se il valore è true il canale è privato. Significa che non è possibile instradare pagamenti sul canale. Nel nostro contesto, non ha senso impostare il valore a true, quindi è false.
- **target_conf**: il numero di blocchi con cui la transazione di finanziamento dovrebbe essere confermata. Nella simulazione abbiamo impostato il valore a 5 blocchi.
- **zero_conf**: valore booleano. Se vero si proverà ad aprire il canale senza numero di conferme. Nel nostro caso è false.

È possibile visionare tutti i campi in OpenChannel - Lightning Labs [55].

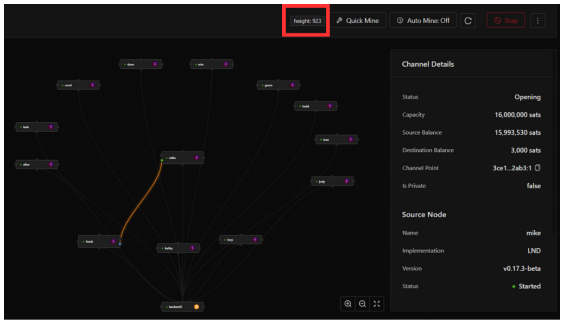

Figura 4.14: Richiesta di apertura canale inviata

Appena inviata la richiesta di apertura del canale, come illustrato nella Figura 4.14, il canale è in uno stato **Opening**. Significa che il canale non è ancora aperto. Affinchè lo stato diventi **Open**, bisogna aspettare il numero di conferme specificate in **target_conf**, cioè cinque. È possibile vedere il numero di blocchi, in alto a destra nella Figura 4.14 accanto al nome **height**. Attualmente il blocco è 910.

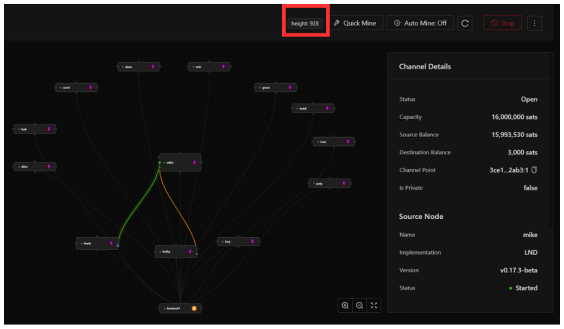

Figura 4.15: Apertura Canale dopo 5 conferme

Dopo che il numero di conferme è raggiunto, come mostrato in Figura 4.15, il canale si trova in uno stato **Open** e può essere utilizzato per instradare pagamenti. Lo stesso accade quando i veicoli vogliono aprire un canale con l'hub. L'unica differenza è che l'hub ha una logica che permette di definire quali canali accettare e quali no. Questa logica viene implementata tramite l'API **ChannelAcceptor** [56] di Lightning Labs. L'API consiste di uno streaming bidirezionale con il client. Ricevuta la richiesta **OpenChannel** il nodo invia al client questa. Il client l'analizza e risponde con un valore booleano se accettarla o meno.

Pagamento

Il pagamento avviene conclusa la ricarica del veicolo elettrico. Come spiegato in Onion Routing 3.3.4, l'Hub non conosce in nessun modo la destinazione del pagamento. Una volta tolto lo strato della cipolla, l'hub, saprà solo su quale canale è diretto il pagamento. Il pagamento da parte dei veicoli elettrici avviene nel seguente modo:

1. **Creazioze Invoice da parte dell'infrastruttura di ricarica**;
2. **Pagamento Invoice da parte del veicolo elettrico**.

La colonnina crea l'invoice, attraverso l'API **addInvoice** [57] andando a specificare alcuni campi chiave come:

- **r_preimage**: il segreto che il destinatario deve utilizzare per riscuotere il pagamento. Per creare la preimage utilizziamo la libreria **crypto** di Node.js e richiamiamo la funzione **randomBytes** [58] che accetta come argomento un number che specifica il numero di bytes da generare e restituisce dati pseudocasuali crittograficamente forti, della lunghezza specificata.

- **r_hash**: l'hash di **r_preimage**. Per creare l'hash della preimage, utilizzo sempre la libreria **crypto** di Node.js, prima creando un oggetto Hash utilizzando la funzione **createHash**. Questa accetta come argomento l'algoritmo da utilizzare per creare l'hash. Nel nostro caso l'algoritmo sarà **SHA-256** [59]. L'oggetto Hash ha a disposizione il metodo **update** [60] che permette di aggiornare il contenuto dell'hash con il valore passato, la funzione **digest** [61] che calcola l'hash da restituire attraverso l'algoritmo specificato in precedenza e i dati forniti attraverso la funzione **update**;

- **value**: il valore da pagare.

Questa come risposta ritorna **paymentRequest** e **paymentAddr** che possono essere utilizzati dal veicolo per effettuare il pagamento. Per tracciare lo stato della fattura, le colonnine si iscrivono a notifiche relative la fattura. Si utilizza l'rpc **SubscribeInvoices** [62] di Lighning Labs, che restituisce un flusso unidirezionale, dal server al client, dove il client viene notificato per determinati eventi riguardanti la fattura. Ad esempio, il pagamento di questa.

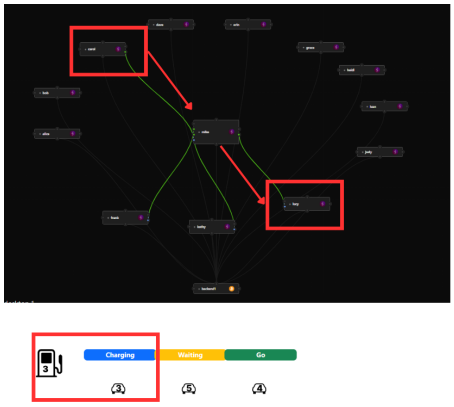

Figura 4.16: Percorso Pagamento

Il primo veicolo che dovrà pagare una fattura è carol(corrisponde al terzo veicolo). Questa fattura viene emessa da lucy (terza colonnina), Figura 4.16. il valore che carol dovrà pagare a lucy è 2,32 euro. Per prima cosa dobbiamo convertire gli euro in satoshi. Per fare questo è indispensabile il valore di Bitcoin. Nella simulazione, per semplicità, impostiamo il valore di 1 Bitcoin a 43000 euro. Il valore di satoshi da pagare è circa 5395.

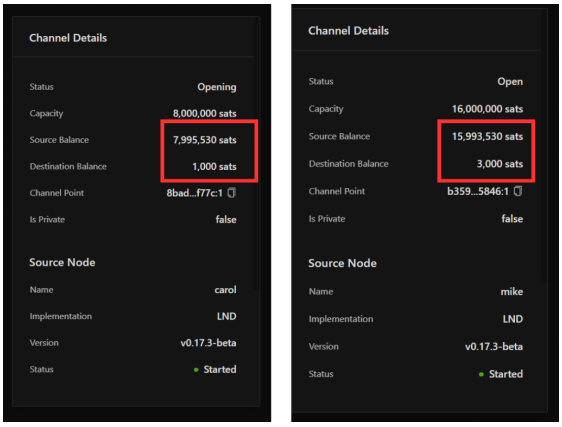

Figura 4.17: Bilancio dei canali prima del pagamento

Nella Figura 4.17, è possibile vedere il bilancio dei canali prima che avvenga il pagamento. Nel riquadro di sinistra, vi sono le informazioni sul canale tra il veicolo e l'hub (carol e mike), mentre nel riquadro di destra ci sono le informazioni del canale tra l'hub e la colonnina (mike e lucy). Per completare il pagamento, il

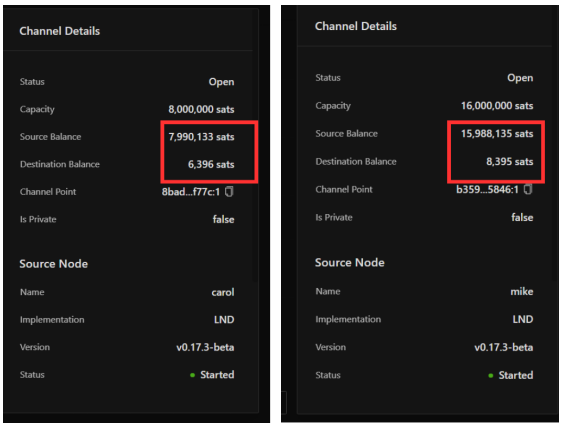

Figura 4.18: Bilancio dei canali dopo il pagamento

veicolo può utilizzare l'rpc **SendPaymentSync** [63] di Lighning Labs. Il veicolo utilizzerà i campi precedenti, forniti dalla colonnina, per completare il pagamento. Effettuato il pagamento il saldo dei partecipanti viene aggiornato come mostrato in Figura 4.18. Nel primo canale notiamo una differenza di un 1sat rispetto al valore del pagamento. Questo è dovuto alle fee che il veicolo deve pagare per passare tramite l'hub per effettuare il pagamento. Nel nostro esempio, la fee è simbolica. Tuttavia è possibile modificare il valore quando si apre un canale attraverso il campo **base_fee**.

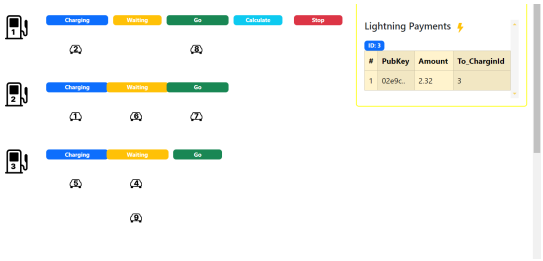

Figura 4.19: Aggiornamento Lightning Payments

Come mostrato in Figura 4.16, avvenuto il pagamento, l'interfaccia si aggiorna automaticamente. Durante questo processo il veicolo che ha completato la ricarica viene rimosso(il veicolo 3, Figura 4.12) e lascia spazio al veicolo successivo in coda. Inoltre viene mostrato il pagamento nella sezione Lightning Payments.

4.4 Design del Software - Secondo strumento di simulazione

4.4.1 Blockchain Quorum

La Blockchain Quorum utilizzata in questa simulazione è organizzata nel seguente modo :

- **Bootnode**: in questa simulazione il bootnode rappresenta la colonnina di ricarica;
- **Cinque Validatori**: in questa simulazione i validatori rappresentano i **plug** della colonnina dove i veicoli si connetteranno;
- **Sei membri**: i membri rappresentano i veicoli elettrici.

4.4.2 Analisi backend

Le interazioni viste nella prima simulazione, 4.3.2, sono sostanzialmente simili. Il cambiamento è nel modo in cui vengono memorizzati i dati sulla blockchain e i dati per svolgere la simulazione.

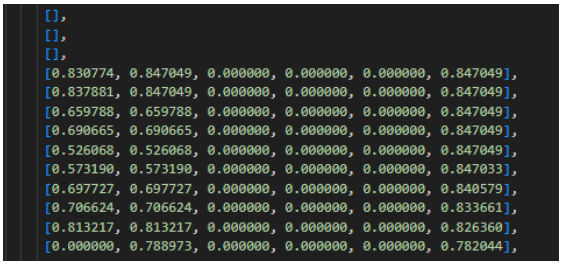

Figura 4.20: Dati Simulazione Fuzzy

I dati vengono forniti sotto forma di più file Json come mostrato in Figura 4.20. Ogni file contiene sei colonne, dove ciascuna rappresenta uno dei sei plug della colonnina di ricarica e ogni riga corrisponde a un minuto del giorno. Le informazioni contenute all'interno della colonna variano in base al file fornito. Vengono forniti file con informazioni sul peso Fuzzy, sulla potenza erogata dalla colonnina, sullo stato di carica del veicolo e sul veicolo che è collegato a quel plug, in base alla colonna in cui si trova, attraverso un id. Ad ogni veicolo è associato ad un peso Fuzzy. La simulazione si svolge iterando sulla lunghezza di uno dei file, che corrisponde a un giorno completo. Quando i veicoli sono in ricarica, per ogni minuto, si controlla il Fuzzy di ciascuno di esso. Se quest'ultimo è superiore alla media dei fuzzy dei veicoli che stanno ricaricando ed è superiore ad una soglia definita, il veicolo guadagna un profitto sotto forma di aumento di potenza erogata da parte della colonnina a cui è collegato.

```
1     let average =
2     files.fuzzy[numTransaction].reduce((a, b) => a + b, 0) /
3     files.fuzzy[numTransaction].reduce((a, b) => (b != 0 ? a + 1 :
      a), 0);
```

Listing 4.13: Calcolo media Fuzzy

La media Fuzzy viene calcolata come in Listing 4.13.

```
1     if (
2             files.fuzzy[numTransaction][ev] >= average &&
3             files.fuzzy[numTransaction][ev] > 0.5
4           ) {
5
6             let minute = (parseInt(files.startMinute) +
    numTransaction) % 60;
7             let hour = Math.floor(
8               (parseInt(files.startMinute) + numTransaction) / 60
9             );
10            let date = new Date();
11            date.setHours(hour.toString());
12            date.setMinutes(minute.toString());
13            date.setSeconds(0);
```

```
14
15              let _hash;
16
17              await acc.contract.methods
18                .updateFuzzy(
19                  ev,
20                  Math.floor(files.soc[numTransaction][ev]),
21                  files.fuzzy[numTransaction][ev].toString(),
22                  files.power[numTransaction][ev].toString(),
23                  files.ev_type[numTransaction][ev].toString(),
24                  date.toLocaleTimeString(),
25                  inputs.chargingPoint,
26                )
27                .send({
28                  from: map.get(files.targa[numTransaction][ev]).
    pub_address,
29                  gas: 500000000,
30                })
31                .on("transactionHash", async (hash) => {
32                  _hash = hash;
33                })
34                .on('receipt', async (receipt) => {
35                  await addFuzzyTransactionToDatabase(
36                    map.get(files.targa[numTransaction][ev]).id,
37                    _hash,
38                    ev,
39                    Math.floor(files.soc[numTransaction][ev]),
40                    files.fuzzy[numTransaction][ev].toString(),
41                    files.power[numTransaction][ev].toString(),
42                    files.ev_type[numTransaction][ev].toString(),
43                    date.toLocaleTimeString(),
44                    inputs.chargingPoint,
45                    map.get(files.targa[numTransaction][ev]).
    pub_address,
46                  );
47                })
48                .on("error", async (error) => {
49                  return res.status(500).json({
50                    error: "Internal Server Error",
51                    message: error.reason,
52                  });
53                });
54            }
```

Listing 4.14: Transazione Quorum

Se il peso Fuzzy del veicolo è superiore rispetto alla media dei Fuzzy degli altri veicoli, si genera una transazione attraverso la funzione **updateFuzzy**, Listing 4.14, salvata in modo permanente sulla Blockchain Quorum per garantire la piena

trasparenza e tracciabilità delle operazioni di ricarica. Dopo il salvataggio della transazione, si salvano le informazioni principali su un file JSON attraverso **addFuzzyTransactionToDatabase** per permettere la visualizzazione dei dati sull'interfaccia utente.

4.4.3 Interfaccia Utente

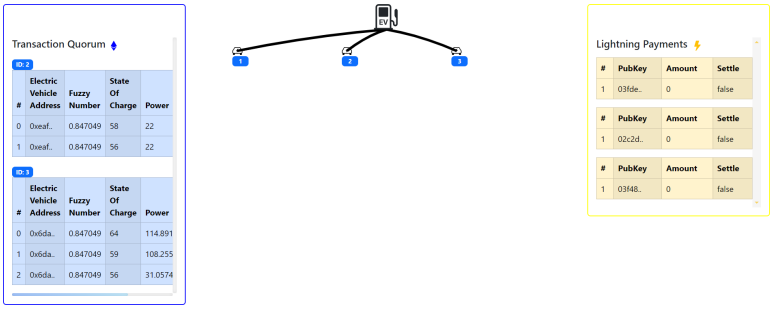

Figura 4.21: Interfaccia Utente - Seconda Simulazione

L'interfaccia nella Figura 4.21 è molto simile a quella della prima simulazione, Figura 4.4. Nella sezione centrale, vi è un cambiamento poichè, come accennato in "Secondo Simulatore" 4.1.3, si vuole simulare l'interazione di più veicoli ad un'unica colonnina per stabilire quanta potenza erogherà questa per ciascuno di essi. I veicoli sono connessi alla colonnina attraverso un Plug. Nella figura 4.21 simuliamo il plug con una linea. Sostanzialmente nelle sezioni **Transaction Quorum** e **Lightning Payments** non cambia niente, se non i dati visualizzati.

4.4.4 Charging Contract

Il cambiamento di questa simulazione è dovuta alla logica che implementa lato backend. Per eseguire la simulazione sono necessari i seguenti dati per ogni veicolo :

- **Fuzzy weights**: il fuzzy del veicolo nel momento della ricarica ;
- **Power**: la potenza erogata dalla colonnina rispetto al veicolo in questione ;
- **Stato di Carica (SoC)**: il livello di carica del veicolo.
- **Soglia minima di Fuzzy**: per poter usufruire di una potenza superiore erogata dalla colonnina, il veicolo deve avere, oltre che un peso Fuzzy maggiore rispetto la media di tutti i veicoli che stanno ricaricando, anche un pesp Fuzzy maggiore rispetto ad una soglia minima predefinita. Nella nostra simulazione, ila soglia è impostata è 0.5.

Ogni set di dati viene fornito tramite un file JSON, dove sono rappresentati come una matrice. Ogni riga rappresenta un minuto del giorno e ogni colonna uno dei cinque plug disponibili. Il nostro scopo è applicare la tecnologia Blockchain per garantire la piena tracciabilità e trasparenza delle operazione di ricarica. Con questi dati a disposizione la simulazione può essere avviata. La simulazione consiste nell'andare a controllare, per ogni minuto, se esiste un veicolo che ha un peso fuzzy maggiore rispetto la media degli altri veicoli e che sia maggiore rispetto alla soglia predefinita. Per fare questo, si calcola la media dei pesi Fuzzy di tutti i veicoli e si controlla, per ogni veicolo, se il Fuzzy è superiore alla media e alla soglia predefinita. Se questo è vero, il veicolo può registrare una transazione sulla Blockchain Quorum per salvare alcuni dati rivelanti. Nello specifico, si richiama la funzione **updateFuzzy** e si salvano sulla blockchain i seguenti paramentri :

- **indirizzo del veicolo**: rappresenta il veicolo che vuole registrare la transazione;
- **plugId**: id del plug al cui siamo attaccati ;
- **stato di carica**: l'attuale stato di carica del veicolo ;
- **peso Fuzzy**: il peso Fuzzy, grazie al quale ha potuto ottenere maggior potenza dalla colonnina;
- **potenza**: la potenza erogata dalla colonnina ;
- **ora**: l'orario in cui è avvenuta la transazione ;
- **indirizzo della colonnina di ricarica** : in quale colonnina il veicolo sta ricaricando .

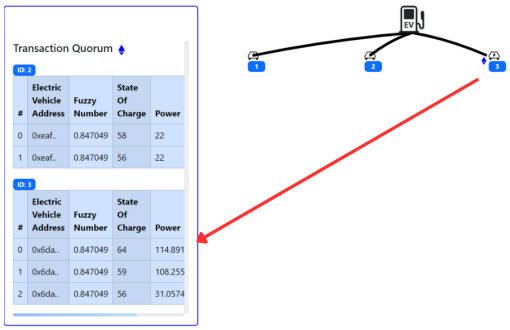

Figura 4.22: Nuova Transazione Quorum - Seconda Simulazione

Una volta registrata la transazione sulla blockchain, l'interfaccia si aggiorna automaticamente mostrando il peso Fuzzy che ha permesso al veicolo di effettuara la transazione e la conseguente potenza che la colonnina ha erogato per questo, come mostrato in Figura 4.22. Nella figura è importante notare che il veicolo uno, non ha ancora effettuato nessuna transazione. QUesto perchè il suo peso Fuzzy non soddisfa i requisiti prima citati.

Capitolo 5

Conclusioni e futuri sviluppi

5.1 Conclusioni

In questa tesi, abbiamo discusso delle problematiche relative alla privacy dell'utente e dell'acquisizione dei dati nelle infrastrutture di ricarica per i veicoli elettrici. Vista l'adozione in massa dei veicoli elettrici che si sta verificando negli ultimi anni, abbiamo bisogno di soluzioni innovative che garantiscano sia la sicurezza da parte degli utenti sia l'utilizzo efficente delle risorse da parte delle infrastrutture elettriche.
Per affrontare queste sfide, ci siamo posti l'obiettivo di utilizzare la tecnologia blockchain al fine di migliorare la privacy dei dati, di garantire la sicurezza nel memorizzare i dati e per migliorare l'affidabilità del pagamento.
Sono state proposte due simulazioni basate sull'utilizzo della Blockhain Quorum e del protocollo Lightning Network. Nella prima simulazione, abbiamo esplorato il ciclo di vita di un veicolo elettrico durante la ricarica. Abbiamo poi constatato che gli autoveicoli possono utilizzare la Blockchain per scambiare informazioni utili in modo sicuro e permanente. Tali informazioni possono essere utilizzate da specifici algoritmi, per guidare gli automobilisti verso una specifica stazione di ricarica.
Per garantire la privacy e la trasparenza nei pagamenti, è stato proposto il protocollo Lightning Network. Esso consente di superare le difficoltà di Bitcoin, proponendo dei pagamenti rapidi, sicuri e scalabili, garantendo allo stesso tempo la privacy per gli utenti. Abbiamo visto come è possibile integrare e interagire con questo protocollo in 4.3.5.
Nella seconda simulazione, abbiamo esplorato un modo alternativo per la gestione della potenza da parte delle infrastrutture di ricarica, mirando a risolvere i problemi relativi alla gestione della loro potenza. A tale approccio, abbiamo affiancato l'utilizzo della Blockchain, che consente di memorizzare i dati in modo efficiente e sicuro, garantendo trasparenza e tracciabilità. Grazie ai meccanismi di sicurezza,

dati dalla crittografia e dai protocolli di consenso, siamo in grado di evitare alcune delle problematiche di sicurezza riscontrate nelle infrastrutture tradizionali.

5.2 Futuri sviluppi

Quando si apre un canale Lightning, l'utente che inizia il processo deve specificare un saldo iniziale, che sarà poi speso per le future transazioni. Tuttavia, nel corso del tempo, potrebbe essere necessario riequilibrare i saldi, per garantire al canale la sua utilità.
In futuro, si potrebbe includere nello studio dei bilanci dei canali, l'analisi di pattern specifici e lo sviluppo di algoritmi, al fine di garantire la redistribuzione del denaro in modo automatizzato. L'implementazione di tali strategie renderebbe i canali Lightning più efficenti e affidabili, migliorando l'esperienza degli utenti e portando ad una maggiore adozione di questa tecnologia. Inoltre, per portare la nostra soluzione ad un livello più alto di funzionalità, nell'immediato futuro, si intende stabilire un collegamento più stretto tra blockchain e algoritmo di simulazione vero e proprio. Questi ci consentirà si ottimizzare e sfruttare appieno la tecnologia Blockchain. Con l'integrazione dell'algoritmo saremo in grado di raccogliere e registrare i dati in tempo reale, consentendo una maggiore precisione e controllo nella memorizzazione e nell'analisi dei risultati. L'analisi dei risultati è un altro importante obiettivo futuro, che sarà supportata da tecniche di machine learning, che ci consentiranno di avere informazioni significative dai dati raccolti in tempo reale durante le simulazioni. Attraverso queste informazioni, potremo identificare trend e anomalie nei comportamenti dei veicoli e nella gestione dei flussi di ricarica così da ottenere una comprensione più approfondita del sistema, consentendoci di ottimizzare ulteriormente le prestazioni e l'efficenza delle infrastrutture di ricarica dei veicoli elettrici.

Capitolo 6

Ringraziamenti

Inanzitutto ci tengo a ringraziare la professoressa Valentina Gatteschi e Fondazione Links per avermi dato l'opportunità di svolgere questa tesi. Ringrazio Alfredo, Alessandro e Silvio, per avermi aiutato ogni volta che ho avuto dubbi. Forza Blockchain!!

Un ringraziamento speciale va alla mia famiglia, che mi ha sempre supportato e **sopportato** sia emotivamene che finanziaramente credendo in me come nessun'altro. Questo piccolo traguardo è dedicato a voi.

Ringrazio la mia QCterme, Annalisa, che è stata sempre al mio fianco. Probabilmente, più di tutti, hai dovuto sopportarmi. Grazie per essermi vicino ogni singolo giorno, senza di te non ce l'avrei fatta. Ti meriti un bel pranzetto preparato da me.... Ti amo.

Un ringraziamento va anche a casa Barrasso, soprattuto a Michele, che è passato da essere coninquilino a essere un compagno di vita. PS se non ce la fai stai a casaaaa.

Infine, ma non per importanza, ringrazio tutti i miei amici di Grotta capitale. Descrivervi come la mia seconda famiglia è riduttivo. Ogni volta che torno è come se non fossi mai stato via. Vi amo!

Bibliografia

[1] IEA. *World Energy Outlook 2023*. Licence: CC BY 4.0 (report); CC BY NC SA 4.0 (Annex A). IEA. 2023. URL: https://www.iea.org/reports/world-energy-outlook-2023?language=it (cit. a p. 1).

[2] IEA. *Global EV Outlook 2023*. License: CC BY 4.0. IEA. 2023. URL: https://www.iea.org/reports/global-ev-outlook-2023 (cit. alle pp. 1, 2).

[3] *I rischi e le sfide dell'integrazione dei veicoli elettrici nelle città intelligenti* (cit. a p. 2).

[4] Hossam ElHusseini, Chadi Assi, Bassam Moussa, Ribal Attallah e Ali Ghrayeb. «Blockchain, AI and smart grids: The three musketeers to a decentralized EV charging infrastructure». In: *IEEE Internet of Things Magazine* 3.2 (2020), pp. 24–29 (cit. a p. 2).

[5] Zacharenia Garofalaki, Dimitrios Kosmanos, Sotiris Moschoyiannis, Dimitrios Kallergis e Christos Douligeris. «Electric vehicle charging: A survey on the security issues and challenges of the open charge point protocol (OCPP)». In: *IEEE Communications Surveys & Tutorials* 24.3 (2022), pp. 1504–1533 (cit. a p. 2).

[6] Ahmed Afif Monrat, Olov Schelén e Karl Andersson. «A survey of blockchain from the perspectives of applications, challenges, and opportunities». In: *Ieee Access* 7 (2019), pp. 117134–117151 (cit. alle pp. 5, 7, 8, 15).

[7] Marianna Belotti, Nikola Božić, Guy Pujolle e Stefano Secci. «A vademecum on blockchain technologies: When, which, and how». In: *IEEE Communications Surveys & Tutorials* 21.4 (2019), pp. 3796–3838 (cit. alle pp. 5, 8).

[8] Imran Bashir. *Mastering blockchain*. Packt Publishing Ltd, 2017 (cit. alle pp. 6, 9, 11).

[9] *Mastering Bitcoin*. 2023. URL: https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch01_intro.adoc (cit. alle pp. 12, 13).

[10] *Mastering Bitcoin*. 2023. URL: https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch11_blockchain.adoc (cit. a p. 12).

[11] *Mastering Bitcoin*. 2023. URL: https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch02_overview.adoc (cit. alle pp. 13, 14).

[12] *Mastering Bitcoin*. 2023. URL: https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch12_mining.adoc (cit. a p. 13).

[13] Andreas M Antonopoulos e David A Harding. *Mastering bitcoin, Transazione in partita doppia*. 2023. URL: https://github.com/bitcoinbook/bitcoinbook/blob/develop/images/mbc3_0202.png (cit. a p. 14).

[14] *Mastering Bitcoin*. 2023. URL: https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch06_transactions.adoc (cit. a p. 14).

[15] CoinLoan. *Elements of Proof of Work*. URL: https://s3-eu-west-1.amazonaws.com/coinloan-blog-assets/2022/09/Proof-of-Work-1.png (cit. a p. 15).

[16] Andreas M Antonopoulos e Gavin Wood. *Mastering Ethereum*. 2018. URL: https://github.com/ethereumbook/ethereumbook/blob/develop/01what-is.asciidoc (cit. alle pp. 16, 18).

[17] Davide Mancino, Alberto Leporati, Marco Viviani, Giovanni Denaro et al. «Exploiting ethereum after "the merge": The interplay between pos and mev strategies». In: *Proceedings of the Italian Conference on Cybersecurity (ITASEC 2023)*. 2023 (cit. alle pp. 16, 17).

[18] Andreas M Antonopoulos e Gavin Wood. *Mastering Ethereum*. 2018. URL: https://github.com/ethereumbook/ethereumbook/blob/develop/07smart-contracts-solidity.asciidoc (cit. a p. 17).

[19] *PROOF OF STAKE*. 2024. URL: https://ethereum.org/it/developers/docs/consensus-mechanisms/pos/ (cit. a p. 17).

[20] Andreas M Antonopoulos e Gavin Wood. *Mastering Ethereum*. 2018. URL: https://github.com/ethereumbook/ethereumbook/blob/develop/12dapps.asciidoc (cit. a p. 18).

[21] *Mastering Lightning Network*. 2021. URL: https://github.com/lnbook/lnbook/blob/develop/01_introduction.asciidoc (cit. alle pp. 19, 20).

[22] *Lightning Network paper*. 2016. URL: https://lightning.network/lightning-network-paper.pdf (cit. a p. 19).

[23] *Mastering Lightning Network*. 2021. URL: https://github.com/lnbook/lnbook/blob/develop/03_how_ln_works.asciidoc (cit. a p. 20).

[24] *Mastering Lightning Network, The Lightning Network protocol suite*. 2021. URL: https://github.com/lnbook/lnbook/blob/develop/images/mtln_0601.png (cit. a p. 21).

[25] Andreas M Antonopoulos, Olaoluwa Osuntokun e René Pickhardt. *Mastering the Lightning Network*. " O'Reilly Media, Inc.", 2021 (cit. a p. 21).

[26] *Mastering Lightning Network*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/07_payment_channels.asciidoc` (cit. alle pp. 22, 23, 26, 27).

[27] *Mastering Lightning Network, The channel establishment message flow*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/images/mtln_0703.png` (cit. a p. 23).

[28] *Mastering Lightning Network, Commitment and revocation message flow*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/images/mtln_0711.png` (cit. alle pp. 28, 29).

[29] *Mastering Lightning Network, The channel close message flow*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/images/mtln_0715.png` (cit. a p. 29).

[30] *Mastering Lightning Network, Atomic payment routing in the Lightning protocol suite*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/images/mtln_0801.png` (cit. a p. 30).

[31] *Mastering Lightning Network, Routing on a network of payment channels*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/08_routing_htlcs.asciidoc` (cit. alle pp. 30, 32–34).

[32] *Mastering Lightning Network, The message flow for HTLC commitment between channel partners*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/images/mtln_0903.png` (cit. a p. 35).

[33] *Mastering Lightning Network, Channel Operation and Payment Forwarding*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/09_channel_operation.asciidoc` (cit. a p. 35).

[34] *Mastering Lightning Network, The HTLC fulfillment message flow*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/images/mtln_0911.png` (cit. a p. 37).

[35] *Mastering Lightning Network, Lightning Payment Requests*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/15_payment_requests.asciidoc` (cit. a p. 38).

[36] *Mastering Lightning Network amount multipliers*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/15_payment_requests.asciidoc#table1502` (cit. a p. 38).

[37] *Mastering Lightning Network, Onion Routing*. 2021. URL: `https://github.com/lnbook/lnbook/blob/develop/10_onion_routing.asciidoc` (cit. a p. 39).

[38] *Mastering Lightning Network, Pathfinding and Payment Delivery*. 2021. URL: https://github.com/lnbook/lnbook/blob/develop/12_path_finding.asciidoc (cit. alle pp. 40, 41).

[39] Soheil Saadatmandi, Gianfranco Chicco, Alfredo Favenza, Alessandro Mozzato, Francesco Giordano e Maurizio Arnone. «Smart electric vehicle charging for reducing photovoltaic energy curtailment». In: *Electric Power Systems Research* 230 (2024), p. 110181 (cit. alle pp. 42, 44).

[40] *Node.js*. URL: https://nodejs.org/en (cit. a p. 45).

[41] *React*. URL: https://react.dev/ (cit. a p. 45).

[42] *React-bootstrap*. URL: https://nodejs.org/en (cit. a p. 46).

[43] *Next.js*. URL: https://nextjs.org/ (cit. a p. 46).

[44] *web3.js - Etherreum Javascript API*. 2016. URL: https://web3js.readthedocs.io/en/v1.10.0/ (cit. a p. 46).

[45] *Web3Provider*. 2023. URL: https://web3js.readthedocs.io/en/v1.10.0/web3.html (cit. a p. 46).

[46] *Wallets and Accounts Overview*. 2023. URL: https://docs.web3js.org/guides/wallet/ (cit. a p. 47).

[47] *web3.eth.Contract*. 2023. URL: https://web3js.readthedocs.io/en/v1.10.0/web3-eth-contract.html (cit. a p. 47).

[48] *Truffle*. URL: https://archive.trufflesuite.com/ (cit. a p. 47).

[49] *Solidity*. URL: https://soliditylang.org/ (cit. a p. 47).

[50] *Quorum Architecture*. URL: https://docs.goquorum.consensys.io/assets/images/Quorum%20Design-0b8564f1a845d87d0679b057df55561b.png (cit. a p. 47).

[51] *Quorum QBFT*. URL: https://docs.goquorum.consensys.io/configure-and-manage/configure/consensus-protocols/qbft (cit. a p. 48).

[52] *Quorum Free gas networks*. URL: https://docs.goquorum.consensys.io/concepts/free-gas-network (cit. a p. 48).

[53] *Lightning Polar*. URL: https://lightningpolar.com/ (cit. a p. 49).

[54] *LND*. 2023. URL: https://lightning.engineering/api-docs/api/lnd/ (cit. a p. 49).

[55] *OpenChannel*. 2023. URL: https://lightning.engineering/api-docs/api/lnd/lightning/open-channel (cit. alle pp. 66, 67).

[56] *ChannelAcceptor*. 2023. URL: https://lightning.engineering/api-docs/api/lnd/lightning/channel-acceptor/index.html (cit. a p. 68).

[57] *ChannelAcceptor*. 2023. URL: https://lightning.engineering/api-docs/api/lnd/lightning/add-invoice (cit. a p. 68).

[58] *Crypto.randomBytes*. 2023. URL: https://nodejs.org/api/crypto.html#cryptorandombytessize-callback (cit. a p. 68).

[59] *Crypto.createHash*. 2023. URL: https://nodejs.org/api/crypto.html#cryptocreatehashalgorithm-options (cit. a p. 69).

[60] *Hash.update*. 2023. URL: https://nodejs.org/api/crypto.html#hashupdatedata-inputencoding (cit. a p. 69).

[61] *Hash.digest*. 2023. URL: https://nodejs.org/api/crypto.html#hashdigestencoding (cit. a p. 69).

[62] *SubscribeInvoices*. 2023. URL: https://lightning.engineering/api-docs/api/lnd/lightning/subscribe-invoices (cit. a p. 69).

[63] *SendPaymentSync*. 2023. URL: https://lightning.engineering/api-docs/api/lnd/lightning/send-payment-sync (cit. a p. 70).