UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in Scienze e Tecnologie Informatiche

STUDIO SU SISTEMI DISTRIBUITI E DECENTRALIZZATI BASATI SU BLOCKCHAIN

Relatore:
Andrea TRENTINI

Tesi di:
Studente: VECCHI Luca
Matricola: 830718

Anno Accademico 2015/2016

Contenuti

1 Inquadramento generale

La blockchain è un database immutabile, distribuito, protetto da crittografia e accettato tramite consenso da un sistema decentralizzato di peer. La natura crittografica della blockchain permette di costruire particolari meccanismi e automazioni, tra cui gli smart contract, che consentono di trasferire da una persona ad un' altra digital asset[1]. Questi ultimi sono beni su cui è possibile esercitare un diritto di utilizzo come lo si può fare nel caso di canzoni, film, segni distintivi, e-mail e tutto ciò che può essere digitalizzato. I digital asset sono trasferibili tramite l'esecuzione di una transazione da parte di un nodo che comunica col sistema. La transazione non avviene tramite una terza parte fidata (come può essere una banca), ma viene presa in carico, anche nello stesso istante, da uno o più nodi, indipendenti l'uno dall'altro, chiamati miner che verificano la sua validità tramite precise regole condivise chiamate regole di consenso. Le transazioni vengono verificate ed accorpate con altre transazioni e dati informativi nel cosiddetto blocco. Il processo di costruzione del blocco viene detto mining ed ogni nuovo blocco creato contiene l'hash di quello direttamente precedentemente così che tutti i blocchi validati dipendano da quello precedente. Il mining è computazionalmente dispendioso e la sua complessità dipendente in modo diretto dalla potenza di calcolo di tutti i nodi della rete così che, la ricostruzione di tutta la catena di blocchi (blockchain) con l'obiettivo di modificare una transazione, divenga un processo troppo lungo e dispendioso da attuare. Il registro pubblico di tutti i blocchi è condiviso tramite consenso da tutti i miner superando i limiti di sicurezza imposti dai sistemi centralizzati e l' utilizzo dello stato dell'arte di algoritmi crittografici, di digital signature ECDSA e di hashing come SHA permette di operare in modo trustless.

Da sempre, nel mercato i rapporti commerciali tra due individui si basano sul rispetto di alcune regole e leggi, la cui tutela viene garantita da terze parti, ossia i giudici. Per quanto riguarda la blockchain, invece, emerge il concetto di trustless, che consiste nell'assenza di un garante fidato, come può essere un giudice o una Certification Autority (CA), che assicuri l'identità delle parti. Infatti è l'architettura della blockchain stessa, che attraverso protocolli crittografici, permette la gestione sicura delle transazioni[1]. Bitcoin e tutte le altcoin[2] derivate dalla prima permettono di costruire una vasta tipologia di transazioni programmabili: si possono trasferire criptomonete in base al destinatario (o meglio la sua chiave pubblica), regalare il denaro al primo che lo reclama, bruciare i coin e persino eseguire del codice all'interno di esse. Questi sono solo alcune funzionalità di base della blockchain che verranno studiate nel corso di questa trattazione ma molte altre possibilità saranno esplorate.

1.1 Cenni teorici

1.1.1 Paxos e il consenso

Uno dei problemi generali dei sistemi distribuiti o decentralizzati basati sullo scambio di messaggi è la generazione del consenso, senza di esso guasti, modifica e perdita di messaggi possono portare il sistema ad uno stato inconsistente. Nella blockchain senza il consenso non è possibile raggiungere la centralizzazione logica perseguendo

[1] `https://rctom.hbs.org/submission/coding-trust-blockchain-contracts-and-the-law-of-the-future/`

[2] `https://www.cryptocoinsnews.com/altcoin/`

la decentralizzazione architetturale, cioè un unico database replicato in più nodi. Il protocollo *Paxos* venne pubblicato nel 1990 da Leslie Lamport e rappresenta uno dei primi protocolli di consenso distribuito di sistemi soggetti a guasti(fault-tolerant)[2]. Altre soluzioni sono stato ideate nel passato, ma Paxos rappresenta un punto fondamentale per capire le problematiche e le soluzioni da adottare per costruire un algoritmo di consenso distribuito. In parte la tecnologia blockchain si basa sulle problematiche esposte in questo sottocapitolo, ma introduce soluzioni aggiuntive per implementare le stesse funzionalità di Paxos agendo in un ambiente *trustless* grazie alla crittografia.

In un sistema asincrono basato sullo scambio di messaggi non esiste alcun algoritmo deterministico in grado di garantire il raggiungimento del consenso anche nel caso del fallimento di un solo unico nodo[3]. Nonostante l'impossibilità di garantire il consenso, Paxos provvede a raggiungere la consistenza delle informazioni, a fronte del blocco del progresso del sistema vista l'impossibilità di tutelare l'operatività dei nodi.Introduciamo un modello semplificato rispetto alla trattazione originale di Paxos e andiamo a definire i meccanismi generali che permettono l'implementazione dell'algoritmo.

Definiamo un modello client-server in cui abbiamo dei nodi che possono svolgere la funzione di client (proposer) o server(acceptor) e la comunicazione tra di essi è basata sullo scambio di messaggi. La rete inoltre può essere soggetta a fault dei singoli nodi nonchè modifica, ritardi e perdita dei messaggi scambiati dai partecipanti. Se l'integrità dei messaggi può essere garantita attraverso meccanismi (crc, parity-bit) il fault dei singoli nodi e la perdita dei messaggi non può essere evitata se non tramite gli acknowledgement dei singoli messaggi.Purtroppo anche in quest'ultimo caso non è possibile garantire per un messaggio spedito m che il relativo acknowledgement, che è un messaggio a sua volta, non sia soggetto a perdita.

Quindi si presuppone che, nel seguente modello, sia presente un protocollo di trasporto affidabile come lo è *TCP*, formato da numeri di sequenza e timer, che permette di costruire la base di un protocollo di trasmissione affidabile tra nodo e nodo senza poter però garantire l'invio di messaggi a causa della rete sottostante.

Consideriamo un sistema composto da N client e M server, nel quale gli N client vogliono eseguire dei comandi c su tutti gli M server attraverso l'invio di un messaggio $execute(c)$. Se assumiamo il caso più piccolo nel quale due clienti u_1 ed u_2, che inviano rispettivamente $execute(c)$ ed $execute(c')$ dove $c \neq c'$, basta che uno solo tra gli M server riceva i comandi nel'ordine sbagliato per produrre stati inconsistenti.

Infatti condizione fondamentale perchè il sistema sia in uno stato consistente è che venga rispettata la *replicazione di stato*. La replicazione di stato di un insieme di nodi avviene quando tutti i nodi del set, preso in considerazione, sono nel medesimo stato S ed eseguendo, nello stesso ordine, una sequenza (potenzialmente infinita) di comandi $c_1, ..., c_n$ raggiungono tutti il medesimo nuovo stato S'. Questa proprietà è fondamentale nei sistemi distribuiti e sia Paxos che la blockchain propongono diversi metodi che permettono il raggiungimento della replicazione dello stato.

Non è possibile ideare un algoritmo che permetta il raggiungimento del consenso dei nodi in un sistema fault-tolerant che implementi una forma di lock o una soluzione centralizzata che gestisca il proseguimento del processo.

Nel primo caso possiamo immaginare che il client che possiede il lock, qualunque sia il modo in cui l'abbia ottenuto, vada in crash e se il sistema non prevede altre

forme di protezione è in deadlock. Il secondo invece introduce un nodo proxy che gestisce i messaggi per conto dei client introducendo un single-poit-of-failure.

Per presentare l'algoritmo di Paxos introduciamo una nuova definizione: *ticket*. Un ticket è una forma debole di lock dotato di un identificativo crescente e delle seguenti funzionalità:

Ristampabile : Un server può fornire un ticket, anche nel momento in cui un ticket sia già stato fornito e non restituito.

Scadenza : Se un client invia un messaggio ad un server utilizzando un ticket t precedentemente acquisito, il server accetterà t se e solo se esso è il ticket generato più recentemente.

Il crash di un client ora, rispetto alla semplice mutua esclusione mediante lock, non priva gli altri nodi della possibilità di riservare risorse, visto che ognuno può richiedere un nuovo ticket valido.

Introduciamo una nuova funzionalità che consente di determinare se la maggioranza dei nostri ticket sia valida.

Il server oltre a tenere in memoria il ticket più recente notifica ad ogni client l'eventuale comando che deve essere confermato per l'esecuzione. Il client u_2 che ha ottenuto un nuovo ticket t_2 e vuole fare lo store del proprio comando c_2, prima interroga il server e se esso segnala che è in attesa del messaggio d'esecuzione per c_1 associato al ticket t_1, dove $t_2 \geq t_1$, allora u_2 piuttosto che tentare di far eseguire c_2 contribuirà all'esecuzione di c_1. Entrambi i nodi, quindi, provano ad eseguire prima c_1 e successivamente c_2 ottenendo così la replicazione dello stato. I client quindi supporteranno sempre il comando associato al ticket più recente salvato in modo da conseguire la maggioranza.

Un client può tornare alla fase 1 in qualsiasi punto dell'algoritmo rendendo indipendente dai tempi o timeout la correttezza dell'algoritmo, ovviamente può essere aggiunto un timer random per ridurre la contesa di due tentativi consecutivi da parte dei client. E' importante notare che nessun nodo nella fase 2 ha modo di decidere se il proprio comando o quello più recente inviato dal server debbano essere proposti in modo tale da imporre il client a contribuire a portare a termine l'esecuzione del comando in tutti i nodi.

Il teorema fondamentale che sta alla base dell'algoritmo di Paxos è il seguente: se un comando c è preso in esecuzione da alcuni server, tutti i server eseguono c.

Dato il messaggio $propose(t, c)$ inviato dal client, definito come la proposta (t,c) per l'esecuzione del comando c identificato univocamente dal ticket t. Assumendo un nodo che vuole eseguire un $c' \neq c$, alla fine dell'algoritmo si troverà ad inviare $propose(t', c')$ con $t' > t$ e $c' = c$ ed il server aggiornerà quindi la proposta iniziale (t,c) ad un ticket più recente memorizzando (t',c).

Una volta che una proposta per c è stata scelta ogni seguente proposta conterrà il comando c fino alla sua esecuzione. Visto che esattamente il primo messaggio $propose(t, c)$ ha generato la proposta (t,c), ogni altro messaggio seguente conterrà il comando c. Quindi visto che verranno spedite solo proposte per il singolo comando c e dato che i client, spediranno il messaggio $execute(c)$ solo quando vedranno che la maggioranza dei server è coerente con il comando, ogni server eseguirà il comando c.

Inizializzazione:

Client	**Server**
$t = 0$	$C = \varnothing$
$c = currentCommand$	$Tmax = 0$

Fase 1:

creo $t = t + 1$	
chiedo conferma al server per t	
	if $t > Tmax$ **then**
	$Tmax = t$
	Invia a client ok($Tmax$,C)
	end if

Fase2:

if la maggioranza ha risposto *ok* **then**	**if** if $t = Tmax$ **then**
Prendi (Tstore,C) con Tstore più alto	$C = c$
if $Tstore > 0$ **then**	$Tstore = t$
$c = C$	Rispondi *success*
end if	**end if**
invia propose(t,c) alla maggioranza	
end if	

Fase 3:

if la maggioranza ha risposto *success* **then**
invia $execute(c)$ a tutti i server
end if

Algorithm 1: Paxos

Può essere dimostrato che la fase 2 dell'algoritmo ha il minor costo possibile per un algoritmo di consenso in presenza di fallimenti. In altri termini, Paxos può essere considerato l'ottimo.[4]

Paxos non permette di perseguire la decisione maggioritaria dei client se la metà o più dei server va in crash.

Il consenso rappresenta il più grande denominatore dei problemi di accordo come il broadcast atomico, l'elezione del leader, la consistenza dei dati e il raggiungimento di uno stato globale.

Il sistema di Lamport però si basa sulla fiducia tra le entità coinvolte nella proposta ed esecuzioni di comandi che può portare all'adozione di soluzioni centralizzate.

La blockchain, basandosi sulla crittografia, introduce una soluzione elegante per il problema del trust nei sistemi rimuovendo completamente l'uso di terze parti per l'autenticazione. Essa porta a nuove possibilità potendo specificare dinamicamente nelle transazioni nuove regole piuttosto. La blockchain permette alle entità di raggiungere il consenso e, sulla base del consenso e delle transazioni precedenti, attuare decisioni autonome sulle nuove transazioni che verrano.

1.1.2 Blockchain, un sistema distribuito e decentralizzato

Prima di tutto chiariamo il concetto di sistema:

Sistema : è un insieme di componenti interconnesse che presentano un comportamento noto attraverso un'interfaccia con l'ambiente esterno[3].

Possiamo trovare sistemi centralizzati, decentralizzati e distribuiti, ma spesso non riusciamo, come nel caso dei sistemi basati su blockchain, a racchiudere una tecnologia in una sola di queste categorie.

Le architetture dei sistemi possono avere dei punti in comune ed essere visti o sotto l'aspetto della topologia del sistema o ragionando sulle relazioni che intercorrono tra gli elementi del sistema stesso.

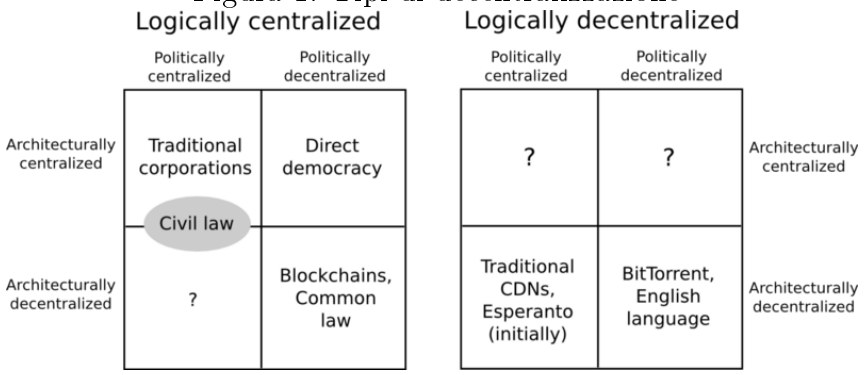
A seconda della topologia di questi sistemi, quindi della coppia composta dagli insiemi dei nodi e delle interconnessioni tra di essi, possiamo andare a distinguere diverse architetture: centralizzata, decentralizzata e distribuita.

Nell' architettura centralizzata, come nel caso specifico di un'implementazione di un'applicazione client-server, un unico nodo è il centro del sistema al quale tutti gli altri nodi si connettono e scambiano messaggi attraverso esso. In questa situazione l'architettura ha seri problemi riguardanti la disponibilità: aggiornamenti, attacchi e malfunzionamenti dell' unico nodo centrale, comprometterebbero l'applicazione per via del *One-Point-of-Failure*. Inoltre il server diventa la principale fonte di bottleneck per l'applicazione a causa della frequente impossibilità di aumentare linearmente le prestazioni. I sistemi centralizzati non verranno quindi presi in considerazione non sono idonei a ospitare un'applicazione fault-tolerante come la blockchain.

Un sistema decentralizzato è realizzato mediante una gerarchia dei nodi tra i quali troviamo tanti sottoinsiemi connessi tra di loro dove si trovano nodi periferici connessi ad un nodo centrale e quest'ultimo, a sua volta, comunica con altri nodi

[3]http://web.mit.edu/6.033/www/lec/s01.pdf

Figura 1: Tipi di decentralizzazione



centrali. Oltre alla diversa topologia, la caratteristica che distingue i sistemi decentralizzati rispetto a quelli centralizzati è che i nodi periferici utilizzano informazioni locali, parziali, con lo scopo di perseguire un obiettivo.

Un'applicazione distribuita invece elimina la gerarchia e distribuisce il controllo a tutti i nodi, nessuno detiene il potere.

Un sistema distribuito ha varie definizioni:

> "Un sistema distribuito è una collezione di computer indipendenti che appare ai propri utenti come un singolo sistema coerente [5]. "

oppure

> " Un sistema distribuito è un sistema nel quale le componenti hardware o software si trovano all'interno di una rete di computer collegati tra loro, e comunicano e coordinano le loro azioni solitamente per mezzo dello scambio di messaggi [4]. "

La blockchain ricade sia nei sistemi decentralizzati in quanto non c'è un singolo punto di controllo, ma anche nei sistemi distribuiti in quanto la computazione avviene in ogni nodo della rete. Ci troviamo quindi nella situazione nella quale non è possibile fare una scelta netta sulla tipologia di sistema implementato, quindi si valuta la blockchain in base a il grado di decentralizzazione di determinati aspetti.

Il concetto di decentralizzazione si può dividere in tre dimensioni diverse che, nonostante siano indipendenti l'una dall'altra, hanno dei punti di contatto(come visibili in figura 1).

- Architectural decentralization: indica la numerosità di quanti dispositivi è costituito il sistema e quanto è tollerata la perdita di alcuni di essi.
- Political decentralization: quanti individui o organizzazioni controllano i dispositivi che costituiscono collettivamente il sistema.
- Logical decentralization: descrive quanto sono inscindibili le interfacce e le strutture dati del sistema. Una semplice euristica è che c'è forte decentralizzazione logica se, isolate due parti del sistema esse continuano ad operare indipendentemente.

[4] S. Mullender Distributed Systems, Addison-Wesley 1993

La proprietà della blockchain di presentare un'unico database consistente si riferisce alla dimensione della centralizzazione logica, ma se può essere un bene per i sistemi basati su blockchain, per sistemi come IPFS diventa una caratteristica da evitare.

La centralizzazione architetturale unita alla decentralizzazione politica nei sistemi computerizzati può essere vista come la scelta da una parte di alcuni utenti di schierarsi per l'utilizzo di certi servizi piuttosto che altri.

La centralizzazione logica implica che la decentralizzazione architetturale sia più difficile da raggiungere, è stato visto con successo che le blockchain, reti di consenso decentralizzato, funzionano, ma sono più impegnative da implementare rispetto ai sistemi completamente decentralizzati architetturalmente ed indipendenti come bittorrent.

Ci sono tre ragioni principali per preferire la decentralizzazione di un sistema:

- Fault tolerance: i sistemi decentralizzati sono meno facilmente soggetti a problemi perchè fanno affidamento su componenti separati che non dipendono l'uno dall'altro.

- Attack resistance : è più dispendioso attaccare un sistema decentralizzato perchè non possiede un nodo centrale facilmente attaccabile ad un costo minore.

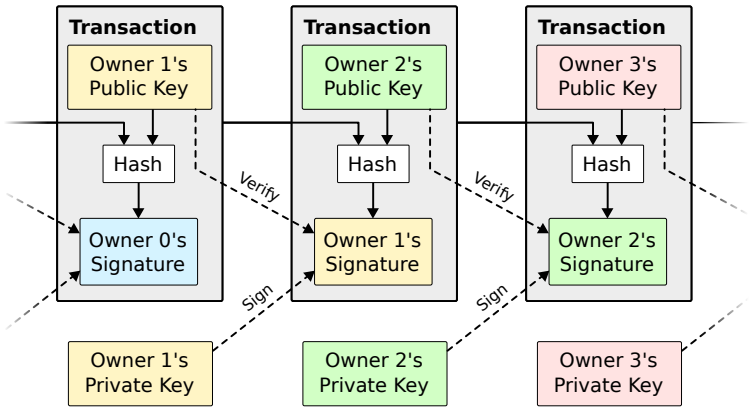
- Collusion resistance: è molto più difficile per i partecipanti in un sistema decentralizzato beneficiare in qualche modo se stesso a discapito degli altri partecipanti.

1.2 La nascita della blockchain: Bitcoin

Il 3 gennaio 2009, un programmatore anonimo, sotto il nome di Satoshi Nakamoto, mina con successo il primo blocco della blockchain Bitcoin dando il via alla criptovaluta più celebre di tutte e a quel processo di innovazione che oggi sta creando sempre più possibilità. Il suo articolo intitolato *Bitcoin: A peer-to-peer electronic cash system*, è uno studio sulla possibilità di creare una moneta virtuale completamente autonoma e distribuita basata sulla crittografia, la proof-of-work e una rete peer-to-peer che la sostiene[6]. Questo lavoro, pubblicato il 31 ottobre 2008, è il condensato di tanti concetti, alcuni dei quali semplici ipotesi altri veri e propri trattati teorici o progetti che nel passato erano stati diffusi. Il paper di Nakamoto è il primo documento formale contenente i concetti fondamentali che hanno preceduto la messa in funzione, poco tempo dopo, di Bitcoin, un sistema di scambio di denaro elettronico tramite una rete peer-to-peer. Nakamoto introduce la sua innovazione partendo da un problema noto che affligge il commercio in internet: l'obbligo di una terza parte fidata, come una banca, la quale con costi impraticabili non permette lo scambio di piccole somme di denaro tra due utenti. Riconosce l'assenza di un sistema di pagamento che non coinvolga terze parti e che non sconfini nell'acquisire informazioni che invadano la privacy. La sua soluzione è costruire un sistema basato non più sulla fiducia verso una third-trusted-party, bensì sulla crittografia.

Prende spunto dal concetto di Bmoney[7] espresso da Wei Dai il quale idealizza un nuovo modo, indipendente ed anonimo (in quanto le persone si presentano mediante chiave pubblica e non più con le loro identità), di scambio di denaro decentralizzato utilizzando la crittografia asimmetrica a chiave pubblica e privata. Nel

Figura 2: Catena delle transazioni o firme digitali



sistema Bmoney ogni utente dispone di un database contenente il bilancio, indicizzato tramite chiave pubblica, degli altri utenti e tutte le forme di comunicazione tra di essi avvengono tramite messaggi broadcast. Idealizzò l'utilizzo di contratti tra più parti: un offerente, un compratore, un escrow [5], che sottoscrivevano un contratto con la possibilità di automatizzare delle funzioni nel caso non venisse rispettato. Purtroppo un sistema decentralizzato di scambio di monete virtuali, per definizione, non può fare più affidamento ad un ente centrale per il conio di nuove monete, come invece accade nel caso delle monete a corso legale[6]. Oltre a ciò bisogna prevedere un meccanismo per assegnare un valore alla criptovaluta. Wei pensò alla possibilità di coniare monete ripagando gli sforzi utilizzati per risolvere dei quesiti proposti da altri, sia intellettuali che pratici. E' proprio con Wei Dai che per la prima volta si può parlare di criptovaluta: il denaro migra da un bilancio all'altro mediante transazioni, esse contengono la chiave pubblica del beneficiario e l'ammontare del denaro ed il tutto viene firmato con la chiave privata del mittente così da permettere a chiunque di verificarne l'autenticità. Il problema che si presenta nella trattazione di Wei Dai è la consistenza e la condivisione di una risorsa comune, un database univoco che possa essere utilizzato per tener traccia dello storico delle transazioni allo scopo di evitare il consumo di risorse già trasferite, il c.d. *double spending*.

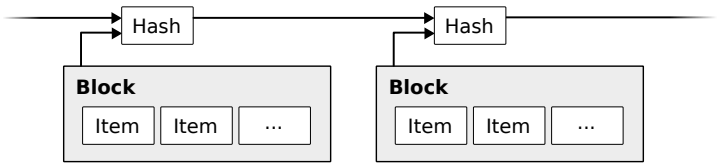
Nel nuovo sistema Bitcoin vengono rielaborati i concetti di Bmoney e utilizzati per creare la catena di transazioni. Come si vede dalla figura 2 se un utente, identificato mediante una coppia di chiavi pubblica/privata, ha ricevuto a suo carico una proprietà si troverà una transazione che lo indicherà. Nakamoto aggiunge delle varianti alle transazioni di Bmoney: viene applicata la funzione hash tra la chiave pubblica del destinatario e l'hash della transazione precedente (quella che ha visto l'attuale mittente essere il destinatario della moneta) e, successivamente, firmati dalla chiave privata del mittente. Questo porta a dare una definizione di electronic coin come una sequenza di firme digitali. Si pone, a questo punto della trattazione, il problema di evitare il double-spending di una singola transazione, che non può essere evitato mediante la sola concatenazione di transazioni.

Una possibile soluzione è un'autorità centrale che sia a conoscenza e controlli

[5]Escrow: un escrow è una terza parte del contratto e garante che vigila sul rispettato delle clausole delle parti.

[6]Sottocapitolo 1.5.1

Figura 3: Timestamp di elementi mediante hash



tutte le transazioni, ma ricadiamo nel problema fondamentale della presenza di una terza parte fidata. Quindi una proposta può essere quella di rendere pubbliche le transazioni, in modo che chiunque possa controllare l'assenza di double-spending e si deve disporre di un sistema che permetta di stabilire un consenso sulle transazioni effettuate. Nakamoto pensò che l'utilizzo di una rete distribuita di timestamp server peer-to-peer sia la soluzione al problema del double-spending. Lo scopo principale dei server di timestamp è proprio quella di raccogliere, registrare e mostrare una prova pubblica che un certo dato, dopo il momento della sua registrazione, esiste ed è precisamente quello. Nakamoto ancora si basa sulla letteratura passata per ideare un semplice sistema di timestamping come si può notare in figura 3.

Il concetto generale è quello di formare una catena,sempre crescente, composta dai valori hash ottenuti tramite l'hash dei documenti e l'hash del blocco precedente. I valori così trovati devono essere pubblicati per permettere il confronto.

Ora che si dispongono gli strumenti necessari per risolvere il double-spending bisogna risolvere da una parte il problema dell'esistenza di più nodi che condividono catene diverse, dall' altra bisogna impedire che si possa facilmente modificare la catena di timestamp. Per risolvere quest'ultimo problema il sistema Bitcoin fa uso dei concetti che sono alla base di Hashcash. Originariamente Hashcash è stato un meccanismo di protezione contro l'abuso di e-mail da parte di bot o spammer [8]. Il concetto che sta alla base di Hashchash, usato per prevenire attacchi di Spam a caselle postali, è quello di impedire che un attaccante possa trarre guadagno dall'uso indistinto di un servizio obbligandolo a consumare tempo e risorse per una determinata azione.

Quello che si richiede è la dimostrazione del lavoro impiegato sia in termini di tempo che di risorse: la cosiddetta *proof-of-work* [9]. La proof-of-work di Hashcash è la prova della creazione di un *token* mediante una CPU cost-function. Quest'ultima deve soddisfare varie proprietà, ma principalmente deve garantire che, da una parte l'esecuzione della funzione per la creazione del token deve essere un procedimento dispendioso, dall'altra la verifica della validità deve essere efficiente.

La proof of work di Bitcoin o PoW usa come CPU cost-function l'applicazione della funzione di hash SHA-256 per calcolare l'hash di un blocco in modo tale da soddisfare una determinata proprietà chiamata *difficoltà*. La difficoltà di un blocco indica il numero di zeri consecutivi in posizione dei bit più significativi e il lavoro medio richiesto, per soddisfare la PoW, è esponenziale nel numero di bit a zero che sono richiesti, mentre la verifica può essere fatta efficientemente mediante l'esecuzione di un numero ristretto di hash. Nakamoto ritiene in oltre che, in previsione dell'aumento delle prestazioni e dell'aumento dei partecipanti, la difficoltà della PoW varia in funzione del numero medio di blocchi creati in ora e, per mantenere questo valore costante, la difficoltà aumenta e diminuisce con esso.

In figura 4 è mostrato il contenuto di un blocco: contiene le transazioni, l'hash

Figura 4: Nonce e l'hash del blocco

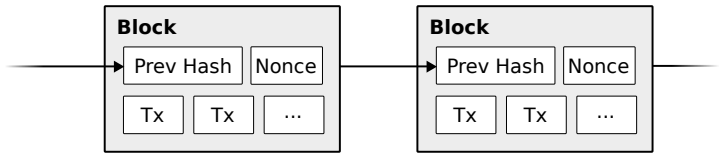
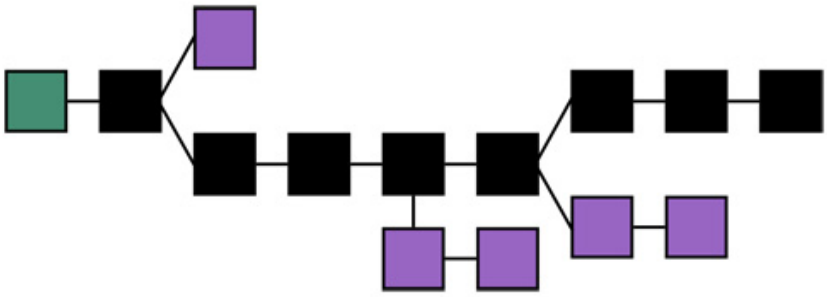


Figura 5: Blockchain soggetta ad fork, si notano colori diversi per differenti branch.



del blocco precedente ed un campo numerico detto *nonce*. Quest'ultimo viene inizializzato a zero e, ogni volta che l'hash del blocco non soddisfa la difficoltà, viene incrementato di uno e ripetuto il calcolo della PoW finchè non viene trovato un hash valido.

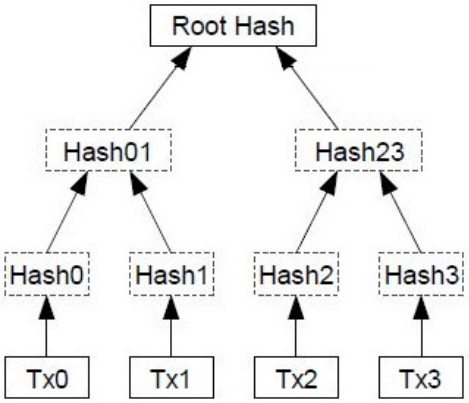
Una volta che gli sforzi della CPU sono stati spesi per trovare un hash che soddisfi la proof-of-work, il blocco non può più essere modificato senza rieseguire il lavoro. Inoltre, dato che i blocchi sono concatenati l'uno all'altro, mediante l'hash del blocco precedente, modificare un blocco passato implica calcolare non solo nuovamente l'hash di quel preciso blocco, ma bensì tutte le PoW successive ad esso.

Anche il consenso dei nodi su quale sia, tra le possibili catene quella *migliore*, viene risolto mediante la PoW. Verrà infatti estesa solo la catena migliore, solo quella con il numero più elevato di blocchi validi.

La catena più lunga gode di un lavoro e una spesa maggiore impiegata e, quindi, il modificare un blocco passato è più dispendioso da attuare rispetto alle altre. Fintanto che i nodi onesti e la loro potenza computazionale rappresenteranno la maggioranza, non sarà possibile per un attaccante la modifica di un blocco passato. Nakamoto definisce il comportamento di un nodo che viene informato di due nuovi blocchi validi, ma differenti. Prima della soluzione viene introdotto il concetto di altezza del blocco e branching. L'altezza di un blocco coincide con la profondità di un nodo in una topologia ad albero dove la radice è il primo nodo minato. Come si vede in figura 5, il branching o biforcazione avviene quando un nodo riceve due blocchi validi della stessa altezza.

La soluzione del problema precedente è un branching della catena: un nodo nel momento in cui riceverà due nuovi blocchi validi per la catena su cui stava lavorando, mantiene i due blocchi e costruisce una biforcazione della catena. Il nodo a seguito della biforcazione o *soft fork* ha due possibili catene, entrambe valide, su cui poter lavorare per estenderle. Si tornerà a lavorare su di una sola catena nel momento in cui la lunghezza di un branch supererà quella degli altri. Una transazione per poter essere registrata deve prima essere creata, come abbiamo visto precedentemente, e

Figura 6: Costruzione del Merkle Tree



diffusa verso tutti i nodi della rete. Ogni nodo collezionerà le transazioni inserendole in un blocco e lavorerà per trovare la proof-of-work. Una volta trovata, il blocco valido sarà condiviso con tutti gli altri nodi che potranno verificare la validità e l'assenza di double-spending.

La funzionalità che introdurremo nasce sempre nell'ambiente del timestamp di documenti e avrà l'obiettivo di assegnare un nome ad un documento digitale che, anche a scapito della perdita di un potere espressivo nel linguaggio umano, conferisce la capacità di identificare univocamente il documento e la sua verifica mediante crittografia[10].

Come ricordiamo un timestamp è un certificato digitale che assicura l'esistenza di un certo documento in un determinato momento. Attraverso la costruzione di un albero binario di valori di hash, detto *Merkle Tree* dal nome del suo ideatore, si può organizzare un meccanismo rapido di verifica del timestamp dei documenti. Come si evince dalla figura 6 l'albero viene costruito a partire dalle foglie che contengono gli hash dei documenti, ogni altro valore assegnato ad un nodo interno viene calcolato mediante l'hash dei due valori sinistro e destro a profondità successiva. Il procedimento termina fino a che non si ottiene la radice detta *Round Root Value*, la quale viene combinata con il valore dell'albero precedentemente trovato calcolando l'hash e ottenendo una nuova radice detta *Round Value*. Ci sono due tecniche di creazione di un sistema di timestamping : trusted-third-party e distributed-trust[10]. Il primo per espletare la funzione di timestamp sfrutta un autorità centrale, pone i Round Value su di un sito fidato o su di un documento in modo tale da rendere i valori immutabili. Il secondo metodo invece distribuisce replica la serie di timestamp su di un gran numero di persone, basandosi sulla convinzione che corrompere un gran numero di utenti firmatari sia difficile. In ogni caso il timestamp di un documento sarà formato da tutti i valori necessari che permettono di ricostruire il merkle tree[7] a partire dall'hash del documento per poi confrontare il valore del Round Value trovato con il corrispettivo valore posto sul sistema trusted. In Bitcoin l'uso è duplice, permette sia la verifica immediata dell'esistenza di una transazione senza che colui che vuole effettuare la verifica debba avere tutte le transazioni, sia permette di inserire nei blocchi, piuttosto che le intere transazioni, solo il Root Round Value

[7]Merkle tree o albero di Merkle dal nome del suo ideatore, e identifica un albero in cui ogni nodo interno è identificato da un hash.

o Merkle Root permettendo di alleggerirne il peso.

Nakamoto prevede che tutti i partecipanti della rete che riescono a soddisfare la PoW estendendo la catena, debbano percepire degli incentivi per gli sforzi compiuti. Gli incentivi, in cripto-monete, sono formati dalle tasse applicate alle transazioni e dai *bounty reward* che sono il compenso per chi riesce a creare un nuovo blocco valido.

Come abbiamo già detto il problema principale di non disporre di un ente centrale, come la zecca dello stato, è che bisogna prevedere un meccanismo di introduzione di monete: per ogni nuovo blocco introdotto verranno coniate nuove monete virtuali. In Bitcoin è stato previsto anche uno strumento, detto *halving*, che consiste nel dimezzare periodicamente il bounty reward di un blocco. Ciò consente di limitare il numero massimo di monete coniabili e quindi di combattere l'inflazione1.5.1. Per far ciò la prima transazione di ogni blocco, detta *coinbase* è riservata al nodo che trova la Pow e gli garantisce l'attribuzione degli incentivi.

Grazie alla pubblicazione del white paper di Bitcoin e dal rilascio del codice sorgente sotto licenza MIT, nel corso del tempo sono nate nuove criptovalute basate sulla prima dette *altcoin*. Le differenze tra Bitcoin e altcoin possono essere sottili e riguardare solamente alcuni aspetti implementativi, ma possono anche differire negli aspetti più profondi come gli algoritmi di consenso o funzionalità aggiuntive come anonimizzare delle transazioni.

Oltre a catene indipendenti, come Tenebrix, Zetacoin, Lightcoin, Dogecoin e molte altre, possiamo trovare applicazioni decentralizzate come Zerocoin o Factom che sfruttano una blockchain costruendoci al di sopra delle applicazioni indipendenti, con funzionalità aggiuntive, che possono essere a loro volta delle nuove blockchain. In particolare ci si riferisce a quest'ultime con il termine di *sidechain* che indica ogni meccanismo che permette a delle criptomonete di una blockchain di essere usate in una catena separata, ma se necessario, possono essere riutilizzate nella catena originaria. Come nel caso di Factom nel quale si predispongono due sidechain, Factom ed Entry Credit, che permettono di operare ed aggiungere funzionaltà alla catena di Bitcoin [8].

Ogni blockchain è stata progettata per aver un preciso scopo che però limita il campo di utilizzo della blockchain da parte di nuove applicazioni, le quali sono obbligate a distribuire l'applicazione su di un sistema che usa la blockchain solo come audit. Questo fino alla nascita di Ethereum. Ethereum si è presentata come una piattaforma decentralizzata basata su blockchain, pubblica, opensource per la pubblicazione di smart contract programmati tramite un linguaggio turing-completo. Se prima un programmatore che voleva sviluppare un'applicazione di aste online era obbligato a utilizzare due sistemi distinti, il livello applicativo basato su sistema centralizzato e la blockchain come storage, ora la blockchain non diventa solo un sistema distribuito di timestamping per criptovalute, bensì una macchina virtule capace di eseguire programmi con tutti i pro e i contro della blockchain[11].

1.3 Proof-of-work e i problemi legata ad essa

Come è stato detto precedentemente l'obiettivo degli algoritmi di consenso, di cui la proof-of-work è uno di essi, è di stabilire un meccanismo attraverso il quale, in una

[8] Il sistema di consenso di Factom. `https://docs.google.com/document/d/1gsXbid3UC1AwaIgmUxjsBav0WDxZi73RXIYDETdmhR8/edit`

rete blockchain pubblica, gli utenti possono riconoscere e condividere collettivamente la stessa catena. Questo permette di minimizzare la fiducia che i nodi della rete devono mantenere tra loro, permettendo l'entrata e uscita di qualsiasi nodo dalla rete in qualsiasi momento. Le famiglie di algoritmi di consenso utilizzati nelle blockchain sono varie e si basano su concetti differenti, ma che puntano tutte nell'utilizzare funzioni che sono impegnative da attuare, facili da verificare e che portano a stabilire delle regole per la determinazione della catena da estendere. La proof-of-work è solo una di esse, ma troviamo anche altri metodi come la proof-of-stake, proof-of-burn, proof-of-capacity. All'interno della famiglia di algoritmi di consenso basati sulla proof-of-work troviamo delle differenze negli algoritmi di cifratura utilizzati e il fattore che gli accomuna è che devono soddisfare la difficoltà, trovando un numero x tale per cui l' HASH(x) ha N bit a zero. In pratica, l'algoritmo di hashing sarà ciò che contraddistinguerà un algoritmo di consenso da un altro. Nella trattazione orginale di Bitcoin viene utilizzato l'algoritmo di Hashcash che a sua volta si basa su SHA256. Il problema che Satoshi non si era posto inizialmente, che nel corso del tempo è stato sempre più nitido, è la differenza tra il contributo di un utente che dispone di hardware di fascia media rispetto ad un investitore che centralizza grande potenza di calcolo (potenza di hashing). La situazione è peggiorata con l'introduzione di hardware performanti quali GPU rig[9], FPGA[10] e ASIC[11] che hanno azzerato il contributo che un utente medio poteva fornire, decentralizzando completamente la potenza di calcolo verso chi ne dispone maggiormente, o come spesso accade, chi può farlo a basso prezzo. Il problema generale risiede nell'utilizzo di funzioni di hashing, come SHA1 o SHA256, che sono ideate per essere performanti. Inoltre l'unico bottleneck di queste funzioni è il numero di operazioni al secondo che un hardware può eseguire e, che rispetto alle CPU di fascia alta, può essere aumentato di vari ordini di grandezza mediante l'utilizzo di hardware dedicati (ASIC) al posto dei comuni processori. Questo unito al fatto che la produzioni di ASIC e il mining è in mano a poche industrie dell'est asiatico ha posto grandi interrogativi sulla qualità della PoW di Bitcoin e su quanto sia effettivamente decentralizzato.

Queste ragioni hanno spinto la blockchain *Tenebrix*, nel settembre 2011, a proporre una variante della funzione crittografica SHA256 e una nuova proof-of-work che non utilizza il numero di operazioni al secondo come bottleneck, bensì operazioni che includono grandi moli di dati e maggior memoria[12]. Il progetto Tenebrix puntava a ridare la possibilità a chiunque di minare questa criptovaluta traendone profitto, essendo progettato per essere inefficiente se eseguito su apparati che vengono usati per il calcolo parallelo o ASIC. Inoltre l'utilizzo di una nuova tecnologia poneva nuovi sbarramenti a chi volesse attaccare la rete mediante il superamento del 51% di potenza di hashing, perchè non solo un attaccante avrebbe dovuto superare il 51% di hashrate dell'intera rete, ma avrebbe dovuto investire denaro nella ricerca e sviluppo di nuovo hardware ASIC per la proof-of-work Tenebrix diminuendo la profittabilità dell'attacco. *Scrypt*, l'algoritmo alla base della PoW della nuova blockchain, non è una funzione crittografica di hashing come lo è SHA256, ma fa parte della famiglia

[9] I GPU rig, letteralmente impianti Graphics Processing Unit, sono quegli apparati di calcolo parallelo che sfruttano diverse schede grafiche come singole unità di elaborazione.

[10] Con l'acronimo FPGA o Field Programmable Gate Array è un dispositivo che consente la programmazione dei circuiti integrati.

[11] Con l'acronimo ASIC o Application Specific Integrated Circuit si indica un circuito elettronico appositamente progettato per essere efficiente nell'esecuzione di particolari compiti.

[12] `https://bitcointalk.org/index.php?topic=45667.0`

di funzioni di derivazione della chiave *KDF* (Key Derivation Function)[12]. In particolare queste funzioni vengono utilizzate per la memorizzazione di password nei database o, come nel caso dell'algoritmo HMAC, per l'invio di messaggi firmati che consente la verifica dell'integrità e dell'autenticità del messaggio.

Le funzioni KDF nascono per aggiungere maggior sicurezza rispetto alle funzioni di hashing nel salvataggio di password nei database. In generale data H, una funzione KDF, dato S, un numero detto salt, si calcola la *derivative key* h di un messaggio m applicando $H(m, s)$. Nelle normali funzioni hash abbiamo un solo parametro (il messaggio) che è ciò che vogliamo trovare se siamo degli attaccanti e vista la non invertibilità della funzione e la distribuzione costante delle preimmagini questo ci porta ad effettuare un attacco brute-force sull'hash del messaggio per risalire al contenuto originario. In particolare, a questo scopo, un attaccante si può appoggiare alle *Rainbow Table* che contengono i valori precalcolati del messaggio e il proprio hash così da fornire l'inversa dell'hash in tempo costante. Per prevenire questo tipo di attacco le funzioni di derivazione della chiave aggiungono una difficoltà, l'hash del messaggio è parametrizzato mediante l'aggiunta di un numero random detto *salt*: se un attaccante vuole conoscere l'inversa in tempo costante deve trovare l'intera rainbow table di un preciso valore salt.

Le funzioni di derivazione della chiave hanno come obiettivo quello di complicare la generazione della chiave, non troppo per essere calcolata in tempi ragionevoli, ma abbastanza da prevenire utilizzi malevoli. Quelle che in teoria possono essere applicate sono le KDF adattive: in input alla funzione oltre al messaggio, il salt, viene aggiunto un parametro che aumenta il numero di iterazioni da applicare alle funzioni interne più elementari. Le funzioni KDF adattive, sono PBKDF2, bycrypt e scrypt. Questi algoritmi permettono non solo di eseguire la ricerca di un hash lentamente, o meglio, non possono essere ottimizzate mediante circuiti appositi, ma sfruttano i limiti della memoria (in termini di capacità e velocità) come limite aggiuntivo.

L'utilizzo di Scrypt al posto degli altri algortimi è dettato da varie ragioni. La prima è che PBKDF2 può essere implementato efficacemente in piccoli componenti con poca RAM come gli ASIC. Bycript poteva essere un candidato ideale, ma è stato sorpassato da Scrypt sulla capacità di utilizzare intensivamente la memoria. Il concetto alla base di Scrypt sono le *Moderately hard, memory-bound functions* ed è stato creato da Colin Percival per ostacolare il bruteforcing nel prodotto di backup di sicurezza Tarsnap[13]. Si basa sull'assunzione che l'unico modo per attaccare una funzione KDF sia effettuare il bruteforcing di tutte le possibili combinazioni e la resistenza a ciò è solo questione di denaro-tempo. Come riporta Percival, il cracking di password è riducibile al calcolo parallelo, nel quale raddoppiare il numero di circuiti che processano le informazioni ne dimezza il tempo di scoperta. Quello che si può fare è incidere sul costo di produzione di questi dispositivi obbligando l'utilizzo di grandi quantità di memoria da parte dell'algoritmo di cifratura(Scrypt attualmente su litecoin alloca 128K). L'algoritmo Scrypt era efficente sulle CPU che disponevano di una memoria cache abbastanza grande con la quale allocare e scambiare dati velocemente. Nel corso del tempo però vari produttori di ASIC, già conosciuti per la produzione di hardware per mining su Bitcoin, hanno creato hardware dedicato che disponga di una buona potenza di hashing su Scrypt, ma gli alti costi e le scarse prestazioni in confronto ad ASIC per il calcolo di SHA256,

[13]http://www.tarsnap.com/scrypt/scrypt-slides.pdf

che erano state previste da Percival, stanno mantenendo l'hashrate delle reti delle blockchain basate su Script maggiormente distribuito.

1.4 Proof-of

E' stata mostrata la proof-of-work, le sue varianti in base all'algoritmo di cifratura utilizzato e la conseguenza di questa scelta, ma esistono altri metodi di determinazione del consenso.

In generale una proof per essere funzionale deve rispondere a determinate caratteristiche:

Target : il target è quella proprietà che deve essere soddisfatta mediante l'attuazione della proof, deve essere generato deterministicamente in modo da consentire la verifica.

Consumo di risorse : l'attuazione del della proof comporta un consumo di risorse, sia che esse siano materiali come l'energia elettrica, sia che siano immateriali come il costo di tenere congelato un asset finanziario.

Verificabilità : deve essere efficiente la verifica del soddisfacimento del target.

1.4.1 Proof-of-stake

Il termine *stake* è letteralmente tradotto in puntata, scommessa e proof-of-stake, indica la famiglia di algoritmi di consenso che si basano su quanto e come un utente possiede un'ammontare di criptovalute per garantire, allo stesso modo della proof-of-work, il consenso dei nodi della rete. Le criptovalute come Ethereum,Nxt e Bitshares utilizzano attualmente, o sono in procinto, di implementare nei loro sistemi la proof-of-stake, ma per comprendere al meglio i concetti che sono alla base di questo cambiamento è bene soffermarsi su Peercoin che è stata la prima blockchain ad introdurre questo nuovo concetto.Peercoin nasce nel 2012 ed è stata la prima criptovaluta a proporre un sistema alternativo di messa in sicurezza della blockchain e di incentivazione dei miner.

Ciò ha permesso di introdurre nuove possibilità: se prima con la PoW veniva centralizzato il potere computazionale per rendere profittabile il mining, con l'adozione della PoS ogni dispositivo può ora competere alla creazione di nuovi blocchi per via di questo sistema leggero ed efficiente dal punto vista energetico e computazionale.

Peercoint sfrutta il concetto di coin age, esso non è nuovo e in Bitcoin viene utilizzato per dare la priorità alle transazioni meno recenti e non per aggiungere sicurezza alla blockchain; viene definito come segue.
Data una transazione T verso l'account b al timestamp t di un ammontare c, indichiamo con $T[b]_c$ l'ammontare del capitale trasferito, $T[b]_t$ il timestamp della registrazione della transazione e possiamo definire la quantità $Coinage(T[i]_c)$ nel seguente modo:

$$Coinage(T[b]_c) \equiv T[b]_c \times Corr(now - T[b]_t) \tag{1}$$

Corr è una funzione di correzione che mappa il timestamp espresso in secondi in un'altra unità temporale e *now* è la funzione che restituisce il timestamp attuale.

Quando un'ammontare di denaro viene immesso in una transazione il coin age di quella quantità viene azzerato o *consumato*.

La proof-of-stake è la realizzazione dell'assunzione che per rendere sicura una rete blockchain non ci si può basare solo sulla difficoltà dell'esecuzione delle operazioni durante la fase di mining. Infatti esse spingono i miner ad investire per aumentarne la capacità di calcolo che si tramuta in transaction-fees sempre più crescenti nel momento in cui altri miner si introducono nella rete per contendersi la validazione dei blocchi.

Come ogni sistema monetario la creazione di nuova valuta deve essere impedita o rallentata, se da una parte il costo della PoW è il consumo di energia-tempo, dall'altra nella PoS è il consumo di coin age e quindi la prova della proprietà di un certo ammontare che viene consumata.

Generazione dei blocchi

Nel sistema Peercoin è presente un nuovo tipo di blocco detto blocco PoS.

La PoS nei nuovi tipi blocchi è una speciale transazione detta *coinstake* (in riferimento anche alla *coinbase* di Bitcoin) nella quale il possessore b della transazione del blocco trasferisce a se stesso $T[b]_c$ consumando la quantità $Coinage(T[b]_c)$ e guadagnando il diritto di generare un blocco.

In dettaglio la transazione *coinstake* sarà formata da:

$$coinstake \equiv (Kernel, I) \tag{2}$$

Una transazione *coinstake* viene firmata da b e sarà formata da un record di due campi:

I : è un vettore di input non spesi, o UTXO riferendoci sempre al sistema Bitcoin, definisce il coinage di quali transazioni passate deve essere consumato. L'indirizzo di destinazione è ovviamente lo stesso di colui che esegue la transazione.

Kernel : è un numero che consente di modificare l'hash della transazione coinstake per soddisfare il target.

Nella PoW di Bitcoin è presente il campo nonce che viene incluso nel calcolo dell'hash del bocco e che permette, stabilito un target, inteso come difficoltà di mining, di ricercare l'hash che lo soddisfi. Anche nella proof-of-stake di Peercoin vengono effetuati gli hash degli input e del $Kernel$ ma il target imposto dal protocollo è dipendente dalla *coinage* degli input.

Quando un blocco viene rilasciato, i nodi prima di accettarlo, verificano che la firma sia autentica, il valore del kernel sia corretto e che soddisfi il target imposto per colui che ha minato quel blocco.

In questo sistema virtualmente più veloce, per via dell'assenza dell'enorme carico computazionale dato dalla difficoltà della PoW, il criterio di scelta della catena più lunga da estendere tra una serie di fork, è basato non più sulla quantità maggiore di lavoro impiegata nella costruzione della blockchain, ma sarà basato sulla quantità maggiore di *coinage* accumulata in un ramo.

In base al protocollo Peercoin durante la nascita della blockchain, quando il totale della *coinage* è posseduta da pochi account, viene impiegata la PoW, mentre a lungo termine si utilizza la proof-of-stake.

Uno dei problemi riguardanti la proof-of-stake è la facilità con la quale un nodo può lavorare su più fork della blockchain andando a competere per estenderle tutte. Peercoin e le altre blockchain basate sulla PoS risolvono questo problema in vari modi, in Peercoin viene firmato (da un'autorità trusted) un ramo che dai nodi verrà considerato il migliore. La soluzione proposta da Peercoin non è quella di evitare la proof-of-work, è sempre presente la possibilità di minare nuovi blocchi tramite la PoW, ma risulta più profittevole ed efficiente sfruttare ricevere gli incentivi minando tramite lo staking e quindi la PoS.

1.4.2 Proof-of-burn, proof-of-capacity e proof-of-authority

Nel 2014 viene implementata una blockchain, *Slimcoin*, la criptovaluta introduce una nuova forma di proof: la *proof-of-burn*. Questa da una parte è simile alla PoS in quanto da la possibilità di oltrepassare l'uso della PoW, ma differisce in quanto il consumo di risorse viene inteso come consumo permanente di un certo ammontare, che ne conferisce però il diritto di minare il nuovo blocco[13]. In pratica il consumo permanente viene attuato creando una transazione che non può essere riscattata da nessuno,ma che risulti facilmente verificabile.

Nello stesso anno venne introdotta, con la moneta digitale *Burstcoin*, la *proof-of-capacity* che si presenta come mezzo per perseguire una più alta decentralizzazione. Il suo funzionamento si basa sulla caratteristica di essere ASIC-resistente in modo da massimizzare la profittabilità che un utente medio, con il proprio computer di casa, possa trarre dal mining, contribuendo a distribuire la potenza di calcolo. Anche se sfrutta il memory-bound è diversa dalla PoS di Litecoin perchè entrambe utilizzano quantità di memoria ben diverse e algoritmi per la determinazione del consenso diverse. Burstcoin alloca durante la sua esecuzione grandi quantità di dati, dell'ordine di diversi megabyte, denominati *plot* e richiede la ricerca di particolari informazioni. Burstocoin è la prima blockchain ha presentare la capacità di eseguire codice autonomo turing-completo all'interno delle proprie transazioni.

Infine viene considerata una proof che non richiede l'uso di risorse per minare nuovi blocchi, la *proof-of-authority*, che semplicemente codifica in fase di settaggio iniziale dei nodi le chiavi pubbliche dei firmatari da cui deve essere considerata valida la creazione di un blocco e quindi trarre da essi la blockchain migliore[14].

1.5 Aspetti economici

Un aspetto fondamentale introdotto da Nakamoto nella sua trattazione e in tutte le criptovalute attualmente in rete è la presenza di incentivi per i miner che consentono di guadagnare valuta digitale. In teoria questo viene attuato per consentire la decentralizzazione dei miner e della loro potenza di calcolo con lo scopo di conseguire un alto livello di sicurezza e imparzialità. Nel mercato composto da miner, utilizzatori di criptovalute, fornitori di servizi quali wallet online e side-chain considerano la criptovaluta come una vera e propria moneta al pari delle monete *fiat*[15].

[14]Proof-of-Authority: `https://github.com/paritytech/parity/wiki/Proof-of-Authority-Chains`

[15]Con monete fiat si intende un mezzo di pagamento legale riconosciuto dallo stato ed emesso da un'autorità, il mezzo non possiede un valore intrinseco come una riserva aurea.

Questa peculiarità permette ai cosiddetti *exchange* di imporsi come banche decentralizzate, le quali scambiano criptovalue o valute fiat applicando un tasso di cambio. Ciò è reso possibile dal largo impiego di criptomonete nel mondo reale come mezzo di scambio tra le persone, spingendo nuovi utenti, interessati a utilizzare questi metodi di pagamento alternativi, a cercare di ottenere un certo ammontare di criptovalute accettando di scambiarlo con beni e servizi.

A sua volta questo fatto permette ai miner che spendono, anche in minima parte, tempo o denaro per rendere sicura la rete, di ricevere delle criptovalute che possono essere tramutate in denaro per ripagare i propri sforzi.

Se così non fosse le reti pubbliche basate su blockchain non potrebbero giovare più del contributo autonomo e volontario dei miner i quali non potendo più effettuare il cambio tra criptovaluta in valuta reale si troverebbero in una situazione di perdita.

Per trarre una stima del consumo di elettricità, quindi del costo associato all'operazione di mining, possiamo moltiplicare l'hashrate (la quantità di hash al secondo della rete) per la potenza assorbita dagli apparati che effettuato l'operazione. Attualmente, la criptovaluta più diffusa della rete (Bitcoin) viene sostenuta da un hashrate di oltre $3,400,000,000 GH/s$[16]. Il costo energetico di un GigaHash, associato ai dispositivi che vengono sfruttati dai miner, è più complicato da stimare, in quanto essi impiegano una grande varietà di mezzi e tecnologie per validare i blocchi: Nonostante questo si può pensare che per via della profittabilità e dell'economicità dell'impiego di hardware ASIC e GPU, solo queste due tecnologie vengano usate. I consumi di questi dispositivi variano da 0.10 W/GH a 10 W/GH, quindi, prendendo una valore centrale come 5 W/GH, possiamo ben accorgerci di quanta potenza ed energia venga impiegata al secondo per minare la criptovaluta Bitcoin. Considerando la somma di tutte le blockchain possiamo considerarlo un valore allarmante.

L'enorme inefficienza del processo di mining (basato su PoW) porta coloro che vogliono trarne guadagno ad investire grandi capitali e spostare i propri stabilimenti dove il costo dell'energia è minore. Non a caso il maggior numero di blocchi minati viene trovato da miner e pool cinesi o dell'est europa [17]. Se nel breve termine questo non può dare delle conseguenze, gli alti costi di gestione possono tagliare fuori qualunque altro utente indipendente dall'operazione di mining portando quei pochi miner a centralizzare sempre di più la potenza di calcolo e, di conseguenza, la decisione su quali transazioni includere o no nei blocchi.

1.5.1 Chi decide il prezzo

Come ad ogni altra valuta reale, la collettività ha assegnato alle criptovalute un prezzo, inteso come costo di cambio tra una valuta e l'altra. La confusione però è molta, ma a livello europeo si è delineato cosa siano le blockchain intese in senso di valute virtuali:

Valute virtuali : si intende una rappresentazione digitale di valore che non viene emesso da una banca centrale o da un'autorità pubblica e non necessariamente collegato a una moneta a corso legale, ma è accettato da persone fisiche o

[16] Stima derivata dal tempo di risoluzione della PoW, che a sua volta varia in base alla difficoltà (`https://blockchain.info/it/stats`)

[17] `http://www.businessinsider.com/bitcoin-pools-miners-ranked-2016-6?IR=T/#2-antpool--1682-17`

giuridiche come mezzo di pagamento e può essere trasferita, immagazzinata o scambiata elettronicamente.[18]

La logica che sta dietro all'assegnazione del valore di un singolo "coin", sia che la criptovaluta sia riconosciuta o meno come moneta a corso legale, è la stessa delle valute fiat. Essa si riduce, ad esempio, al chiedersi quanto vale un RMB in Euro: tutte le valute hanno un proprio valore di cambio rispetto a tutte le altre.

Il mercato costituito dagli scambi delle valute è il Forex[19] e Bitcoin e le altre valute virtuali non sono incluse in esso.

Nonostante le criptovalute siano riconosciute solo come mezzo di pagamento e non come bene, esistono gli scambia valute o *exchange*, i quali sono il punto di contatto tra l'ecosistema delle valute virtuali e delle monete a corso legale.

Poloniex, Kraken, BTC-e sono solo alcuni exchange che offrono questi servizi di compravendita di criptovalute. I servizi offerti da questi enti sono vari, dalla compra-vendita e trading di qualsiasi valuta, al fornire wallet con cui gestire i propri bilanci online.

E' dal fenomeno di trading che si stabilisce il prezzo, in particolare si fa riferimento al modello *Price taker e maker*. Esso definisce le operazioni che due entità, maker e taker, eseguono al di sopra di una piattaforma di trading interna all'exchange, entrambe operano in duplice modo:

Maker : notifica al *libro degli ordini* che è intenzionato a comprare una quantità V di un bene per un determinato prezzo P, oppure a vendere un totale V per un compenso di P.

Taker : osservando gli ordini, segnalerà la volontà di comprare V al prezzo di P (dove P è maggiore o uguale del minimo prezzo dei Maker per vendere V) o di vendere V al prezzo di P (dove é minore o uguale del massimo prezzo dei maker per comprare V).

L'incontro tra una richiesta dei maker e la corrispettiva richiesta del taker che la soddisfa produrrà lo scambio tra le due entità. L'exchange potrà applicare delle tasse alle richieste dei maker e taker per incentivare o disincentivare certe operazioni o volumi scambiati[20].

L'incontro tra le offerte tra maker e taker determinerà il valore delle criptovalute, e quindi sarà molto volatile e potrà cambiare molte volte nel giro di pochi minuti. Il valore di un coin non sarà univoco, ma andrà contestualizzato a seconda di quale exchange si sta usando, in quale valuta vuole essere convertito e in che preciso istante lo si sta osservando [14].

1.5.2 Halving

In alcune reti blockchain è presente il concetto di halving che consiste nel periodico dimezzamento del bounty reward. Quest'ultimo è il compenso, in termini di criptovaluta, che il miner riceve per aver aggiunto un blocco alla catena. Nei sistemi

[18] ttps://www.ecb.europa.eu/pub/pdf/other/virtualcurrencyschemesen.pdf

[19] Foreign exchange market o mercato valutario

[20] `https://www.kraken.com/help/fees`

che implementano questa funzionalità, l'halving avviene in periodi fissati (es. ogni n blocchi) che portano, nel corso del tempo, ad una situazione nella quale il numero di monete prodotte per blocco sarà zero e il numero di monete prodotte avrà raggiunto il tetto massimo. Se l'implementazione potrebbe sembrare una feauture con scopi prettamente tecnici, in realtà è inserita in molte blockchain per gestire unicamente aspetti economici. Lo scopo di questa procedura è diminuire l'inflazione che l'immissione di nuova valuta può dare a lungo termine. L'inflazione determina la diminuzione del potere d'acquisto della criptovaluta, osservabile tramite il cambio rispetto a monete fiat, causato dall'introduzione nella rete blockchain di nuove monete. In Bitcoin il fatto che oltre il 75%,rispetto al tetto massimo di 21 milioni, delle monete sia già stato introdotto in circolazione e l'halving ne riduca sempre di più l'introduzione porta ad una continua crescita del prezzo legato ad una domanda sempre più crescente per accaparrarsi il maggior numero di valuta prima che l'emissione si fermi.

Un'altro aspetto importante è che i miner basano la sostenibilità dei loro sforzi dagli incentivi che la blockchain li assegna: oltre a il reward del blocco e le fee applicate alle transazioni non ci sono altri metodi interni alla blockchain per ricavare un compenso[21][22].

Se il reward del blocco si dimezza e i valori del cambio o delle tasse rimangono invariati molti miner non riusciranno più a sostenere i costi e si sposteranno a minare, se possibile, altre monete più proficue. Se, come in Bitcoin, le transazioni richieste superano quelle incluse nei blocchi, solo le tx con fee più alte verranno incluse portando l'aumento del valore delle fee complessivo.

Alcune blockchain non implementano l'halving perchè, come Ethereum, pensano che la somma totale delle criptovalute perse dagli utenti (attraverso la perdita delle loro chiavi o della propria password, l'invio di valuta in transazioni inspendibili) svolgerebbe in parte la stessa funzione dell'halving.

[21] `http://smartmoney.startupitalia.eu/bitcoin-blockchain/51665-20160202-halving-bitcoin`

[22] In altri sistemi, come Ethereum, è presente anche il concetto di Uncle che sarà analizzato nei capitoli seguenti

2 Ethereum

2.1 Ethereum e la sua storia

Ethereum è un progetto open source divenuto operativo il 30 Luglio 2015 con la versione Frontier. Bisogna retrocedere al 2013, anno in cui si incominciava ad ideare il progetto e a scrivere le prime implementazioni del codice dei nodi in Go e C++. Come di consueto l'apertura al pubblico della blockchain è stata preceduta,nel settembre 2014, dalla pre-vendita della criptovaluta Ethereum, l'Ether, permettendo così al progetto di raccogliere circa 18 milioni di dollari. Raccolti i fondi questo ha permesso di aprire fondazioni e di proseguire con lo sviluppo. Attualmente lo stato del progetto è visionabile sulla piattaforma *github* all'indirizzo `https://github.com/ethereum` e come si può notare lo sviluppo è su più fronti: si progetta il porting del codice dei nodi in più linguaggi, si sviluppano nuovi linguaggi orientati agli smart contract come Solidity, Viper e molto altro. Questo progetto è il risultato di idee precedenti dove alcune, come Bitcoin, sono state le innovazioni che sono alla base di Ethereum.

Prima fra tutte la creazioni di Satoshi Nakamoto il quale ha introdotto concetti nuovi e risolto grandi ostacoli che non avevano permesso la proliferazioni di determinati sistemi. La prima innovazione è stata quella di creare una moneta basata su un sistema peer-to-peer decentralizzato. La seconda introduzione che ha dato il via allo sviluppo è stato l'utilizzo in Bitcoin della proof-of-work per garantire il consenso pubblico dei nodi sul sistema di transazioni. Nessuno prima di quel momento era riuscito a superare il problema di risolvere la concorrenza di due transazioni ed evitare il double-spending, in Bitcoin si adotta il concetto di *first-to-file*: la prima transazione validata diventerà parte dello stato delle transazioni a cui la seconda transazione dovrà sottostare per essere valida[6]. Nel capitolo successivo verra mostrato formalmente il concetto di stato e come la proof-of-work risolve il problema.

L'utilizzo della blockchain per scopi che non siano la sola gestione di criptovalute ha diversi casi di successo ma presentano tutte delle difficoltà. Il primo fra tutti è un sistema decentralizzato di registrazione di nomi a dominio chiamato Namecoin[15]. Esso è un implementazione di un sistema DNS e di un *Registrar* che registra i nomi sulla base del criterio di precedenza. Namecoin purtroppo basa la sua sicurezza sul protocollo di Bitcoin, una gestione della registrazione ristretta e un network molto variabile che portano le persone a non utilizzare questo sistema. Questo si traduce in una rete meno sicura a causa della bassa potenza di hash.

In Ethereum è stato possibile introdurre due concetti che in Bitcoin non erano stati implementati appieno: le monete colorate e la verifica di codice interno alla blockchain. Quest'ultimo è un limite alla struttura classica della blockchain, quella basata su transazioni di criptovalute: la verifica di una transazione si riferisce solo allo stato del bilancio, bytecode eventualmente interno alle transazioni richiederebbe la verifica di tutto il codice della blockchain dal blocco di genesi[6, SPV - Simplified Payment Verification]. Questo ci porta a presupporre che per implementare un linguaggio di programmazione al di sopra della blockchain sono richieste strutture dati apposite. Le monete colorate invece sono criptovalute che,mediante una transazione, viene assegnata uno speciale identificativo. Questo permette di differenziarle e quindi di disporre di pagamenti configurabili in base al colore della moneta.

Questi aspetti sono implementati in Bitcoin ma presentano tutti alcuni problemi

insormontabili, che si concludono con il problema più grande dell'assenza di un linguaggio turing-completo per programmare le transazioni: in bitcoin le transazioni non hanno stati intermedi, possono o non possono essere inserite nella blockchain, non è possibile effettuare cicli (in modo efficiente), creare sistemi di autenticazione a più fasi o utilizzare come input delle transazioni dati che non siano lo script di verifica basato sulle curve ellittiche.

L'obiettivo di Ethereum è di dare la possibilità, attraverso poche righe di codice, di fornire un contratto intelligente che abbia codificato dentro di se lo statuto del contratto e permetta di gestire asset digitali in modo autonomo. Questo viene reso possibile da una blockchain che oltre a registrare dati e transazioni, condizioni base perché ci sia scambio di denaro elettronico e possibilità di memorizzazione, salva nei blocchi anche il codice degli smart contract e lo stato della loro esecuzione[16].

Nel capitolo seguente si mostrerà il funzionamento di Ethereum e il suo modello.

2.2 Ethereum e il suo funzionamento

Nei precedenti capitoli è stato visto il funzionamento di una blockchain partendo da quella Bitcoin e sono stati mostrati i principi e le scelte fatte in base ai problemi da risolvere. La storia di Ethereum è in continua evoluzione e questo si può trovare sia nella grande attività sulle repository pubbliche, sia sui blog dei principali sviluppatori. Attraverso la consultazione di questi materiali si può apprendere il funzionamento di Ethereum.

Ethereum si basa sul concetto di macchina a stati basata sulle transazioni: il primo stato contiene il blocco di genesi, una serie di transazioni racchiuse in un singolo blocco porterà il sistema in un nuovo stato.

In questo capitolo verrà utilizzata la terminologia in lingua originale per conservare chiarezza con i documenti reperibili.

La struttura di base su cui è definita la macchina di Ethereum segue il seguente paradigma.

2.2.1 Il paradigma della blockchain

Uno stato potrà includere alcuni dati come i bilanci degli account, i contratti stipulati e qualunque altro dato che può essere digitalizzato. Le transazioni rappresentano un arco valido tra due stati validi, dove una transazione T si dice valida se non porta in stati $\boldsymbol{\sigma}$ inconsistenti. Uno stato $\boldsymbol{\sigma}$ è non consistente se, per esempio, è stato ridotto un bilancio senza che ci sia stata un rispettivo aumento. Data una tupla di transazioni T_i e uno stato $\boldsymbol{\sigma}_t$ è possibile applicare la funzione di transizione di stato di Ethereum Υ per ottenere un nuovo stato arbitrario tra le transazioni :

$$\boldsymbol{\sigma}_{t+1} \equiv \Upsilon(\boldsymbol{\sigma}_t, T_i) \tag{3}$$

Data la funzione di calcolo dell'applicazione di una transazione ad uno stato della blockchain ora è possibile stabilire formalmente il risultato del calcolo di un blocco. Prima ricordiamo che un blocco B è un record formato da $(H, (T_0, T_1, ...))$ dove H è l'header del blocco e T_i è la transazione i-esima del blocco, spesso $(T_0, T_1, ...)$ verrà indicato solo con T. Inoltre consideriamo che l'utente che riuscirà a computare l'hash del blocco, in modo tale da soddisfare la proof-of-work (mining), riceverà un compenso, il bounty reward.

Possiamo ora trovare la funzione Π di transizione dei livelli dei blocchi che, dato lo stato attualeσ_t, un blocco B contenente $(H, (T_0, T_1, ...))$ calcola il nuovo stato $\sigma_t + 1$ così come segue:

$$\Pi(\boldsymbol{\sigma}, B) \equiv \Omega(B, \Upsilon(\Upsilon(\boldsymbol{\sigma}, T_0), T_1)...) \quad (4)$$
$$\boldsymbol{\sigma}_{t+1} \equiv \Pi(\boldsymbol{\sigma}_t, B) \quad (5)$$

Dove Ω è la funzione che assegna al miner il reward del blocco.

2.2.2 World state

Lo stato globale di Ethereum alla fine di una computazione di blocco viene definito come $\boldsymbol{\sigma}$ e rappresenta il mapping tra gli indirizzi, identificatori di 160 bit, e gli stati degli account serializzati. Questa struttura dati è una particolare implementazione di un albero Merkle-Patricia ottimizzato per essere efficiente [23] e non viene salvata nella blockchain in quanto può essere ottenuto ricalcolando tutte le transazioni dal blocco di genesi. Dati l'indirizzo a e $\boldsymbol{\sigma}$ lo stato globale indichiamo con $\boldsymbol{\sigma}[a]$ lo stato a-esimo definito dalla seguente tupla:

$$\boldsymbol{\sigma}[a] \equiv (\boldsymbol{\sigma}[a]_n, \boldsymbol{\sigma}[a]_b, \boldsymbol{\sigma}[a]_s, \boldsymbol{\sigma}[a]_c) \quad (6)$$

Esistono due tipologie di account in Ethereum: account posseduti da utenti esterni, quindi gestibili tramite chiave privata, e account autonomi gestiti dai contratti. Essi hanno in comune 4 campi: $\boldsymbol{\sigma}[a]_n$ o **nonce**: nel caso di account esterni è il numero di transazioni inviate da l'account a, nell'altro è il numero di contratti creati da esso. $\boldsymbol{\sigma}[a]_b$ o **balance**: è il bilancio in *Wei* dell'account a. $\boldsymbol{\sigma}[a]_s$ o **storageRoot**: è l'hash del root node dell'albero *trie* che codifica la memorizzazione di contenuti per l'account a. $\boldsymbol{\sigma}[a]_c$ o **codeHash**: è l'hash del codice per la EVM associato a questo account che verrà invocato quando questo indirizzo riceverà un messaggio di chiamata. Si distinguerà un account esterno da uno orientato ai contratti perchè nel caso di account esterni il codeHash è una stringa vuota.

2.2.3 Transazioni

Una transazione T è una singola istruzione firmata crittograficamente e nell'ambiente Ethereum possiamo trovare due tipologie:

- Transazioni usate per inviare messaggi.
- Transazioni che permettono la creazione di nuovi account.

Per prima cosa chiariamo il concetto di *Gas*. Esso rappresenta la quantità degli sforzi da effettuare per compiere una determinata operazione. Ad ogni operazione come somma, creazione contratto, scrittura dati, suicidio del contratto, viene assegnato un certo numero di Gas. Il costo delle operazioni unito al campo gasPrice determinerà le fee applicate alla transazione.

Se indichiamo con T una transazione possiamo definire formalmente le due tipologie:

[23] `https://github.com/ethereum/wiki/wiki/Patricia-Tree`

$$(T_n, T_p, T_g, T_t, T_v, T_{\mathbf{i}}, T_w, T_r, T_s) \quad \text{Se } T_t = \varnothing \tag{7}$$
$$(T_n, T_p, T_g, T_t, T_v, T_{\mathbf{d}}, T_w, T_r, T_s) \quad \text{otherwise} \tag{8}$$

Come si può osservare entrambe hanno degli elementi in comune:

nonce o T_n: lo scalare rappresenta l'n-esima transazione effettuata dal mittente della transazione .

gasPrice o T_p: uno scalare equivalente al numero di Wei che vengono pagati per unità di *Gas* per tutto il costo computazionale derivato dall'esecuzione della transazione.

gasLimit o T_g: è uno scalare e rappresenta il limite di *Gas* che si vuole utilizzare per la transazione, non può essere modificato ed è una quantità che viene versata anticipatamente.

to o T_t: è l'indirizzo (160 bit) a cui verrà recapitato il messaggio. T_t se possiede tutti i bit a zero ($T_t \equiv \varnothing$) indicherà una transazione di creazione di un account.

value o T_v: rappresenta il numero di Wei da trasferire all'indirizzo T_t o nel caso in cui quest'ultimo sia $\varnothing$ rappresenterà il bilancio di partenza di un'account creato.

v, r, s o T_w, T_r, T_s: Rappresentano 3 firme digitali e sono usate per determinare il mittente della transazione.

Nel caso della tipologia di transazioni legate alla creazione di un account è aggiunto il campo:

init o T_i: è array illimitato di byte che specifica il codice per EVM che verrà eseguito durante l'inizializzazione del contratto.

L'esecuzione del codice di *init* avviene solo in fase di inizializzazione e restituisce il *body* che è quella parte di codice che verrà invocata ogni volta che l'indirizzo del contratto in cui è salvato il body (tramite codeHash) riceverà un messaggio.

Una transazione orientata ai messaggi invece contiene una parte complementare all'*init*:

data o T_d: è array illimitato di byte che viene usato come input della chiamata contenuta nel messaggio (transazione).

Il gasLimit come abbiamo visto determina il tetto massimo che il mittente di una transazione è disposto a spendere, in particolare $gasPrice * gasLimit$ definiranno l'ammontare dei Wei che verranno spesi nell'invio di una transazione. Se nell'esecuzione di una transazione la quantità di gas spesa è pari o inferiore al gasLimit al mittente della transazione verrà restituita la differenza, in caso contrario la computazione si arresterà e la transazione non sarà conclusa, ma a causa della spesa computazionale sostenuta dal miner, non verrà restituito il gas speso.

2.2.4 Blocco

Un blocco in Ethereum è la registrazione dell'esecuzione di una serie di transazioni. Esso conterrà i risultati delle computazioni o meglio l'hash delle strutture dati che permettono di tener traccia e di rendere possibile la verifica delle esecuzioni. Definiamo un blocco B come un record:

$$B \equiv (B_H, B_{\mathbf{T}}, B_{\mathbf{U}}) \tag{9}$$

Dove B_H è l'header del blocco, B_T è la lista di transazioni e B_U la lista degli *uncle-blocks*. Questi ultimi sono anche definiti *Ommers* e sono blocchi validi che condividono con l'attuale blocco la stessa altezza. Questo permette di aggiungere una sicurezza maggiore alla blockchain perchè i blocchi passati sono stati coperti dal nuovo blocco trovato e, anche, da tutti gli Ommer-block della rete.

L'header B_H è formato da vari campi alcuni dei quali prevedono la presenza di strutture dati precalcolate come i trie dei *World State*, delle *transazioni* e del *Transaction Recipt*. I campi sono i seguenti:

parentHash o H_p: l'hash del blocco precedente.

ommersHash o H_o: l'hash della lista dei blocchi Ommer.

beneficiary o H_c: l'indirizzo dell'account a cui verranno assegnate le fee del blocco.

stateRoot o H_r: contiene l'hash della radice del *Trie State* dopo la computazione delle transazioni.

transactionsRoot o H_t: contiene l'hash della radice del Trie che contiene le Transazioni.

receiptsRoot o H_e: contiene l'hash della radice del *Transaction Recipts Trie*.

logsBloom o H_b: contiene l'hash del filtro di Bloom che contiene i log della computazione delle transazioni, esse sono contenute nel *Transaction Recipts Trie*.

difficulty o H_d: rappresenta la difficoltà dell'algoritmo di PoW e viene calcolata in base al timestamp e alla difficoltà del blocco precedente .

number o H_i: corrisponde al numero del blocco precedente incrementato di 1, tenendo conto che il blocco di genesi ha il campo $H_i = 0$.

gasLimit o H_l: rappresenta la somma delle quantità gasLimit nelle transazioni.

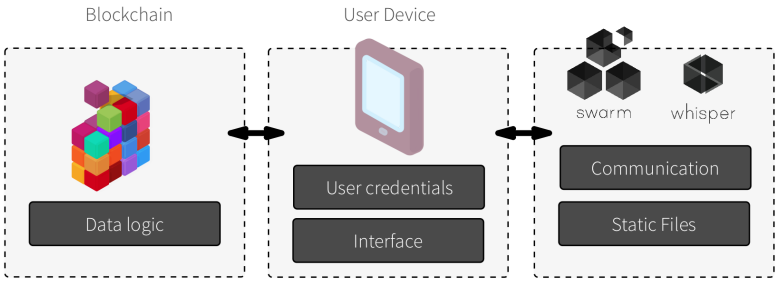
gasUsed o H_g: rappresenta la quantità di Gas usata nella computazione delle transazioni.

timestamp o H_s: rappresenta il tempo di creazione del blocco espresso mediante l' Unix timestamp.

extraData o H_x: è un array di massimo 32 byte che contiene dati aggiuntivi.

mixHash - nonce o H_m - H_n: sono due hash su cui si basa la proof-of-work e permettono di verificare che è stata spesa una determinata quantità di lavoro.

Figura 7: Modello decentralizzato



2.3 Strumenti per lo sviluppo

Ethereum vuole superare l'approccio client-server che contraddistingue gran parte delle applicazioni di rete, nelle quali troviamo un server, in cui risiedono la logica applicativa, i file statici, le credenziali dell'utente, che comunica e scambia dati con il client. Il client diviene quindi solo un interfaccia per usufruire dei servizi del server.

Un'architettura più decentralizzata, come quella di Ethereum, ci permette di disporre di un insieme di macchine e di protocolli che permettono di dividere compiti e componenti del sistema, ridistribuendone alcuni sulla macchina del client e altri in una rete peer-to-peer. Come si può osservare dalla figura 2.2.4, il layer data, il quale si occupa della gestione dei dati, è affidato in mano agli smart-contract sulla blockchain, i file statici sono forniti tramite *Swarm* e la comunicazione in tempo reale è gestita dal protocollo *Whisper*[24]. Il client interagirà con l'interfaccia per l'applicazione, ma dovrà gestire l'autenticazione. Distribuendo l'autenticazione all'utente e affidandosi alla crittografia si solleva un server centrale della responsabilità di gestire la sicurezza di tutti i client. La distribuzione dei contenuti e la fruizione di protocolli peer-to-peer permettono di ridurre i costi e migliorare l'efficienza dei servizi. La divisione del layer di data con quello di presentazione permette di dare la possibilità a chiunque di personalizzare e diffondere una propria interfaccia per la stessa applicazione.

Di seguito vengono riportati alcune componenti del sistema di Ethereum.

Mist

Mist viene presentato come l'interfaccia grafica o GUI di Ethereum o il browser per navigare nel Web 3.0 delle Dapp. In pratica è un cointaner Electron connesso a una piattaforma Meteor che permette quindi la totale programmazione degli applicativi come pagine web (javascript,html e css). Con esso è possibile interagire con Swarm e reperire le interfacce grafiche per le applicazioni distribuite.

Electrum-wallet

Il wallet in generale è un'applicazione che consente di connettersi ad una rete

[24]Whisper: è una parte del protocollo peer-to-peer di Ethereum che permette di far comunicare gli utenti attraverso lo stesso mezzo usato dalla blockchain `https://github.com/ethereum/wiki/wiki/Whisper`

blockchain ed effettuare svariate operazioni tra cui inviare, verificare pagamenti e interagire con la blockchain tramite RPC. Nel caso specifico di Ethereum-wallet esso è un wallet ma è implementato come Dapp e non come applicazione stand-alone. Essendo una Dapp è stata creata per essere eseguita con meteor, è ottenibile tramite repo [25] o via Swarm mediante Mist.

Swarm

Swarm è un piattaforma distribuita di storage oltre che un servizio di distribuzione dei contenuti, è implementato nativamente nello stack applicativo del web3 di Ethereum. Il principale obiettivo di Swarm è di provvedere una piattaforma di salvataggio distribuita e ridontante per i file relativi alle interfacce delle Dapp, che permetta l'upload di contenuti senza richiedere che il nodo debba restare connesso alla rete di peer. Dal punto di vista dell'utente, Swarm non è diverso dal WWW, eccetto che l'invio di informazioni non è verso uno specifico server, ma al sistema di peer. Dal punto di vista oggettivo è una piattaforma di storage peer-to-peer e fornisce soluzioni che sono resistenti a DDOS, censura, downtime e sono tolleranti ai guasti e implementano un sistema di incentivazione e di scambio della criptovaluta comune a tutte le blockchain. Swarm implementa il protocollo di rete di Ethereum devp2p e il sistema DNS di Ethereum.

2.4 Problemi delle applicazioni basate su blockchain

La blockchain, come già riportato, è un database immutabile e non consente la modifica di un dato registrato a meno di non implementare determinati meccanismi. Questo problema trasportato in Ethereum e nel paradigma dei contratti porta all'immutabilità del codice dei contratti che sono caricati sulla blockchain. Il seguente problema verrà risolto nel sotto capitolo 3.3.1. Un' altra problematica di ogni applicazione di Ethereum è l'assenza della possibilità di implementare un oracolo, un generatore di numeri casuali, nella EVM di Ethereum. Il problema in prima istanza è il fallimento del consenso, della rete di nodi, sull'accettare un blocco che contiene un contratto che ha generato ed effettuato un'operazione su un valore ipoteticamente random. La funzione pseudocasuale porterà ogni nodo a determinare un valore diverso portando al rifiuto del blocco. Viceversa se l'oracolo è deterministico non è possibile implementare tutti quei servizi che si basano sull'estrazione casuale di un numero.

Sulla rete si trova una soluzione [26]a questo problema che contempla una particolare implementazione di uno smart contract e l'uso di una funzione deterministica per la generazione del numero pseudocausale. La soluzione viene trattata considerando un problema reale: la vendita di biglietti(ticket) per una lotteria a premi. Definiamo con $t[i]$ il numero del ticket comprato, con n l'altezza del blocco dopo la quale l'estrazione può essere attuata e x il numero di blocchi precedenti ad n su cui è calcolato il numero pseudorandom(eseguento l'hash e l'operazione di modulo sui risultati ottenuti). La vendita di ticket avviene fino al blocco $t \equiv n - x - 1$ dopo il quale lo smart contract nega la vendita. Ogni utente disporrà quindi di ticket su cui

[25] `https://github.com/ethereum/meteor-dapp-wallet`

[26] `https://gist.github.com/alexvandesande/259b4ffb581493ec0a1c`

non potrà inferire prima dell'estrazione finale, in quanto il calcolo sarà basato sulla creazione dei blocchi successivi a t. I miner potranno influenzare l'estrazione attraverso la creazione di nuovi blocchi ma non potranno decidere quale preciso ticket verrà estratto.

La EVM non ha accesso alla rete e a funzioni dipendenti dal sistema operativo che la ospita in quanto è isolata. Questo oltre a mantenere al sicuro l'esecuzione degli smart contract rappresenta, però, un problema di tutte le blockchain. Basti pensare a un contratto che vuole stimare il costo reale di un servizio, l'Ether ha un tasso di cambio e varia in funzione dell'andamento del Forex delle criptovalute. Oltre a questo possiamo essere interessati a richiamare la funzione di un contratto a fronte di un certo evento che può essere il superamento di una certa data temporale.

Per questo ultimo scopo risiede nella rete principale un contratto, *Ethereum Alarm-clock*[27] che consente di schedulare eventi tra contratti e di permettere la verificabilità delle invocazioni. L'implementazione presenta alcune caratteristiche e funzionalità che consentono di prevenire l'abuso di questo contratto, ma ovviamente esso si affida all'esecuzione di processi esterni[28].

Un altro sistema esterno alla blockchain che permette di disporre maggiori funzionalità, rispetto al sistema Alarm-clock, è *Oraclize*[29], esso si presenta come il gateway verso Internet per le blockchain per Ethereum e Bitcoin. E' un sistema molto complesso che permette di predisporre di un contratto e di API con le quali si accede alle funzionalità infinite della rete[30]. In Internet l'autenticità e l'integrità dei dati via web viene garantità dall'utilizzo del protocollo https e terze parti fidate (CA come Verisign, GlobalSign, ecc.). Questa architettura si contrappone all'infrastruttura trust-less che risiede nelle blockchain ed Oraclize punta a sostituire l'autenticità dei dati, effettuata mediante terze-parti, con l'*authenticity proof*.

2.5 Sviluppare in Solidity

Abbiamo visto che la EVM esegue un proprio bytecode, infatti è possibile programmare in assembly-ethereum il proprio contratto. Ovviamente ciò è complicato e richiederebbe fin troppo tempo a scrivere un programma efficiente e la manutenibilità sarebbe difficile da ottenere. Per questo sono stati creati linguaggi di più alto livello, come Serpent, Viper e Solidity, che permettono la stesura di codice ad alto livello che una volta compilato può essere eseguito nella rete Ethereum.

In generale la EVM ha delle limitazioni:

1. è isolata dal sistema operativo e un altra funzione di un altro contratto e invocabile solo se si conosce il suo indirizzo e la relativa interfaccia (ABI).

[27] `http://www.ethereum-alarm-clock.com/`

[28] Come recita la documentazione: " Execution guarantees: You may have noted at this point that this service relies on external parties to initiate the execution of these transactions. This means that it is possible that your transaction will not be executed at all. In an ideal situation, there is a sufficient volume of scheduled transactions that operating a server to execute these transactions is a profitable endeavor. The reality is that I operate between 3-5 execution servers dedicated filling this role until there is sufficient volume that I am confident I can turn those servers off or until it is no longer feasible for me to continue paying their costs".

[29] `http://www.oraclize.it/`

[30] Utili api che possono essere utilizzate attraverso Oraclize sono per esempio quelle di Wolframaplha `https://docs.oraclize.it/#ethereum-integration-simple-query`

2. Ogni istruzione della EVM ha un costo e l'esecuzione può sollevare un'eccezione in base alla quantità di *GasLimit* contenuta nella transazioni, non possono essere eseguite operazioni intensive.

3. Una volta creato il contratto non è possibile modificarlo.

La scelta di quale linguaggio utilizzare è ricaduta su Solidity, che è un linguaggio ad alto livello orientato ai contratti staticamente tipizzato, che permette di scrivere smart contract che supportano l'ereditarietà, implementano funzioni di librerie esterne e utilizzano nuovi tipi definiti dall'utente. Solc è il compilatore per il linguaggio Solidity consente, dati una serie di contratti, di produrre il bytecode per la EVM, la documentazione, le interfacce abi.

Prima di discutere delle innovazioni che porta è bene far precedere dei concetti di questo linguaggio.

```
contract A{
   uint amount;
   function A() payable {
      B b = new B();
      amount=msg.value - b.bbb();
   }
}

contract B{
   function bbb() returns (uint){
      return 42;
   }
}
```

La principale entità che può essere creata è il contratto e, come i linguaggi OOL, esso presenta un costruttore, una funzione omonima che viene richiamata solo durante il deploy del contratto, le altre funzioni potranno essere invocate in ogni altro momento su un'istanza del contratto.

Eseguendo il codice nella rete di Ethereum ovviamente avremo a disposizione una sintassi e dei costrutti specifici per gestire i pagamenti ai contratti: la label *payable* informa al compilatore che la funzione $a()$ può accettare b Ether da una transazione e l'istruzione $msg.value$ restituirà l'ammontare b.

I contratti dispongono della possibilità di interagire tra di loro comunicando tramite le medesime modalità con le quali un account esterno interagisce con un contratto (call esterna) oppure mediante chiamate speciali che consentono di eseguire funzioni nell'ambiente del contratto chiamante (delegate-call)[31].

La prima modalità è una chiamata nella quale un contratto può specificare la quantità rimanente di gas nel GasLimit per la chiamata, l'ammontare di Ether da inviare e il payload. Il contratto ricevente, che può essere lo stesso chiamante, allocherà nuova memoria e potrà accedere al payload per eseguire la chiamata. Viceversa una delegate-call eseguirà le funzioni nell'ambiente del contratto chiamante e quindi l'ammontare del gas rimasto e per esempio le chiamate a libreria come $msg.value, msg.sig$ e tutte le altre proprietà di msg riporteranno lo stesso valore.

[31] Per maggiori informazioni: `http://solidity.readthedocs.io/en/develop/introduction-to-smart-contracts.html?highlight=delegatecall#delegatecall-callcode-and-libraries`

L'innovazione progettuale di Solidity si fonda proprio sull'utilizzo di librerie e sull'invocazione di delegate-call verso esse.

I contratti come tutti gli account di Ethereum vengono referenziati tramite il loro indirizzo univoco a 160 bit e l' *application binary interface* (ABI).

Interfaccia:

```
[{"constant":false
,"inputs":[]
,"name":"addr"
,"outputs":[{"name":""
           ,"type":"address"}]
,"payable":false
,"type":"function"}]
```

Codice:

```
contract A{
   function addr() returns (address){
      return msg.sender;
   }
}
```

L'ABI di un contratto è statica e viene determinata in fase di compilazione; è composta dai nomi delle chiamate a funzione, dai tipi degli argomenti e dai valori di ritorno, nonchè eventualmente da precondizioni, postcondizioni e invarianti.

Dopo quest'introduzione su alcune funzionalità del linguaggio possiamo trattare le librerie e il *librarydrivendevelopement* in Solidity.

Una libreria è un contratto che non può avere un bilancio in Ether, questo viene assicurato dal compilatore e tutte le chiamate a funzione verso esso sono delle delegate-call. Il deploy di un contratto consiste nell'effettuare una transazione speciale con il bytecode del contratto inserito nel payload verso un indirizzo speciale. Il bytecode ottenuto dalla compilazione di un contratto, che dipende da una libreria pre-caricata nella blockchain, non può essere registrato su di essa perché l'indirizzo della libreria deve essere linkata nel bytecode del contratto.

Nel caso di un contratto che utilizza una libreria chiamata Library potremmo trovare nel bytecode un place holder simile a __Untitled:Library____ che indica il punto in cui inserire l'indirizzo della libreria contenuta nella blockchain.

Il paradigma della blockchain applicato al codice è di dare la possibilità di usufruire di qualsiasi libreria scritta e registrata da altri, risparmiando il costo e lo spazio sulla catena con la conseguenza di poter disporre di librerie sicure.

Inoltre il paradigma dove ogni operazione ha un costo può essere utilizzato per disincentivare gli account (esterni e non) a non abusare della rete. In Solidity la funzione `return;` trova il suo utilizzo nelle call, nelle chiamate che non hanno side-effectC. Il comando interrompe l'esecuzione di una funzione e la EVM ritorna all'esecuzione del codice chiamante, se esistente, se no ritorna il valore alla console. La funzione `throw;` invece solleva un eccezione: nel caso di call(dove non viene eseguita una transazione) non viene consumato del gas/ether, ma nel caso di transazioni consuma tutto il gas inserito nella transazione(normalmente se avanza del gas da una computazione viene restituito il rimanente). Il comando `throw;` può essere catturato mediante catch, ma se viene propagato può essere usato come strumento per punire economicamente un account.

Solidity è un linguaggio creato recentemente, esso è in fase di sviluppo e tante operazioni non sono possibili, per esempio non è possibile restituire un'istanza di un tipo creato da una funzione e non esistono i tipi primitivi double e float. A prescindere da ciò Solidity e la blockchain introducono un interessante modo di sviluppo di applicazioni distribuite.

3 L'applicazione

Lo scopo della seguente ricerca è l'implementazione di un'applicativo finalizzato alla gestione delle opere autoriali. Esso consente la registrazione, la visione di opere autoriali e la classifica delle stesse in base all'utilizzo reale degli utenti. Il sistema sarà implementato mediante smart contract in esecuzione sulla piattaforma Ethereum, ciò consentirà di disporre delle proprietà di trasparenza, tracciabilità e immutabilità proprie della blockchain.

Verrà utilizzata la seguente terminologia:

Utente o account: è la persona che sottoscrive la registrazione di un opera (proprietario) o acquista il diritto all'utilizzo di una determinata opera(compratore).

Opera : è l'opera creativa, identificata tramite i metadati che il proprietario inserisce in fase di registrazione.

Contratto di registrazione o registrazione: rappresenta la prova verificabile della cessione del diritto di utilizzo dell'opera.

Digichain o libro mastro o ledger: è l'entità decentralizzata che gestisce l'interazione tra le componenti, è unico e da esso è possibile esercitare tutte le operazioni permesse.

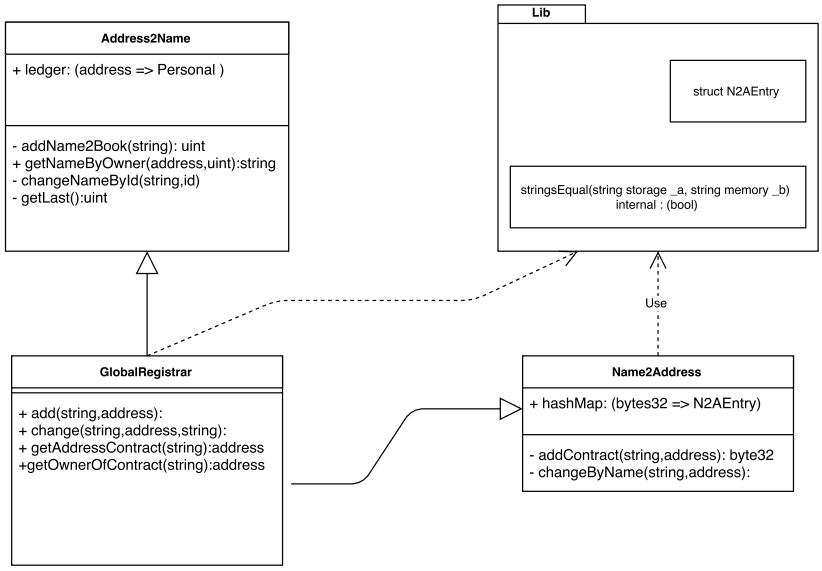
Anche se lo sviluppo di smart contract si è concentrato sull'applicazione *Digichain*, sono stati sviluppati altri contratti tra cui *GlobalRegistrar* che consente di condividere in modo immediato un indirizzo tramite un nome simbolico. Questo permette di non dover più comunicare l'indirizzo di un contratto mediante sistemi esterni alla blockchain, ma più semplicemente sfruttare il concetto di database e smart contract per implementare un ente di registrazione.

3.1 Registrar

In Ethereum ogni account, che sia uno smart contract o un utente esterno, viene identificato mediante indirizzo univoco a 160bit. Quando un utente vuole spedire un certo ammontare di Ether o vuole eseguire una chiamata ad un contratto deve recuperare in tutti e due i casi l'indirizzo del destinatario e, solo nell'ultimo, deve reperire l'abi del contratto[32]. Per l'ultimo caso, visto che è alquanto dispendioso l'inserimento nella blockchain dell'intera interfaccia delle chiamate del contratto, nella main-chain di Ethereum l'abi viene fornito tramite Swarm. Per gli indirizzi è possibile implementare un servizio efficiente di *name registrar* basato su blockchain, esso permette di:

- registrare sotto un nome simbolico un indirizzo a 160 bit.
- dato il nome simbolico reperire, se presente, l'eventuale indirizzo accoppiato.
- verificare la proprietà della registrazione di una coppia $<nome\_simbolico, indirizzo>$.

Figura 8: Registrar



In figura 8 è possibile vedere il contratto *GlobalRegistrar* che estende due contratti: *Name2Address* che implementa la correlazione degli indirizzi ai nomi e *Address2Name* che fornisce la funzionalità di raggruppare i nomi registrati da un utente.

Il container `globalRegistrar` contenuto in *GlobalRegistrar.js* fornisce dei metodi con cui inizializzare l'istanza di un globalRegistrar, il cui codice è già caricato nella blockchain e il cui indirizzo è stato pre-caricato nel file javascript.

```
> glb = globalRegistrar.getGlobalRegistrar()
> glb.add.sendTransaction("nomeSimbolico","0x438dfd4dfd26a42961",{from:
    eth.coinbase})
```

La variabile `glb` contiene l'istanza che collega l'abi e l'indirizzo del contratto Ethereum precedentemente caricato[33]:

Come si può vedere la registrazione della coppia di valori avviene tramite una transazione con side-effect[34].

L'interrogazione del contratto, per la richiesta dell'indirizzo associato ad un nome, avviene mediante il metodo `.getAddressContract("nomeSimbolico")`. Un utente, interessato alle nuove registrazioni che vengono salvate nella blockchain, può mettersi in ascolto degli eventi lanciati dal contratto stesso [35] che riportano dati informativi:

```
> watcher = globalRegistrar.watchEventNewContract(glb)

Nuova entry : < nomeSimbolico |
    0xdfd5c578c9f5a448e71bf69cd4789211b7d94494> da l'account :
    0x438dfd4dfd26a42961d878a1e27eb9f40abb0d19 .
```

[32] Riferirsi al capitolo 2.5

[33] Si suppone che l'utente stia lavorando connesso a una blockchain nella quale sia già stato fatto il deploy del contratto GlobalRegistrar e abbia pre-caricato l'ambiente contenente i file `*.js`.

[34] Consultare l'appendice B.

[35] Consultare l'appendice C e al sotto capitolo 2.2.4.

Se due utenti vogliono registrare sotto lo stesso nome due indirizzi qualsiasi, vige il criterio *first-to-file*,cioè il primo arrivato acquisisce il privilegio di poter registrare e modificare l'indirizzo del nome registrato. Il contratto impedirà ad ogni utente di modificare record che non siano di sua proprietà e applicherà delle sanzioni nel caso venga rilevato un tentativo[36].

3.2 Digichain

Digichain è lo smart contract per la gestione delle opere autoriali. Sfruttando il contratto GlobalRegistrar gli utenti possono interagire collettivamente con il medesimo smart contract digichain semplicemente accordandosi sul suo nome simbolico.

Lo smart contract consente:

- la registrazione di opere autoriali e la specifica dei metadati che la riguardano.
- la consultazione delle opere presenti, nonchè delle loro caratteristiche.
- la compravendita temporanea di opere e la verifica dello stato della cessione.
- la verifica della *reputazione* dell'opera.
- la rivalsa da parte dell'autore dell'opera di rimuovere il diritto concesso al compratore in casi specifici.

Con il termine *reputazione* di un opera si intende la somma di *ore_di_utilizzo * costo_tempo_opera* che le persone hanno speso nell'usufruire dell'opera. Questo consentirà a tutti gli utenti di visionare la reputazione di ogni opera per scegliere quella più opportuna.

Un opera è identificata tramite:

owner : l'indirizzo del proprietario dell'opera.

nome : il nome dell'opera.

costoBase : definisce in Wei l'ammontare minimo che un acquirente deve pagare per poter godere temporaneamente dell'opera.

costoTempo : definisce in Wei il costo al minuto che un utente pagherà per usufruirne.

reputazione : è il valore della reputazione acquisita nel corso della vita dell'opera.

Registrazione

Un utente, allo scopo di interagire con il contratto Digichain, è obbligato a acquisire una sua istanza. Ciò si ottiene, per esempio, tramite l'istanza GlobalRegistrar nel seguente modo:

[36]Le sanzioni sfruttano i side-effect del comando throw2.5.

```
> digichain=deployer.getIstanceFromAbiAndAddress(
   compiled.contracts["Digichain.sol:Digichain"].abi
   ,glb.getAddressContract("digichain")
   )
```

Ottenuta l'istanza `digichain`, su di essa un utente può invocare il metodo `.creaOpera()` al quale passare i metadati dell'opera(`(string titolo,uint256 costoBase,uint256 costoTempo)`) ed incapsulare il tutto in una transazione.

```
> costoBase = 200000
> costoTempo = 10000
> titolo = "Il titolo di questa opera"
> digichain.creaOpera.sendTransaction(titolo, costoBase,
    costoTempo,{from: eth.account[0]})
```

Se la transazione è spedita correttamente[37] ed inclusa in un blocco valido, il contratto Digichain avrà memorizzato sulla blockchain l'opera appena creata che sarà visibile a tutti tramite la call `digichain.opere(idOpera)`.

Cessione del diritto

Ogni utente, che non sia colui che ha registrato l'opera, ha la possibilità, versando un ammontare b maggiore o uguale al *costoBase* dell'opera, di acquisire il diritto di utilizzo della stessa.

Questo avviene attraverso i seguenti comandi:

```
> bilancio = 5000000
> idOpera = 2
> digichain.compra.sendTransaction(idOpera,{from: eth.account[0],value:
   bilancio})
```

Il pagamento per l'acquisto del diritto effettuato allo smart contract Digichain produrrà un nuovo contratto R *Registrazione* che lega l'acquirente, l'opera e l'ente di gestione delle opere Digichain. Qualsiasi operazione sul contratto R appena creato potrà passare solo dalla gestione dello smart contract Digichain. Se t_0 è il tempo di creazione del contratto R, indichiamo con c il costo accumulato associato all'uso dell'opera nell'istante t_1 ed è uguale al prodotto $(t_1 - t_0) \cdot costoTempo$.

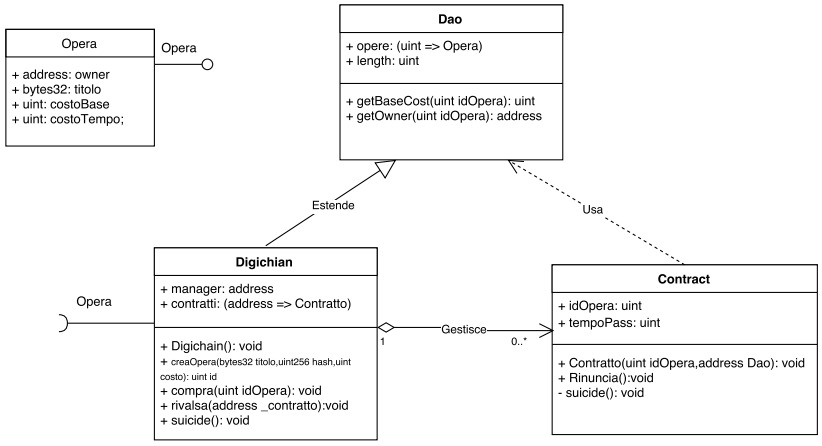
Il possesso del diritto decade in due casi: se la spesa dell'utilizzo dell'opera ha superato il bilancio inizialmente versato oppure se il compratore recede dal contratto.

Recessione

In ogni istante, un utente che ha acquisito un diritto di utilizzo di un'opera può recedere dal contratto attraverso l'invocazione l'invio di una transazione sul metodo `digichain.rinuncia()`. La recessione viene richiesta allo smart contract Digichain e ciò comporta la cancellazione del contratto R. Durante questa operazione la differenza tra il bilancio b depositato in R e il costo accumulato c viene restituita all'utente che ha recesso, mentre, la quantità c viene accreditata sul bilancio del

[37] Le transazioni

Figura 9: Class diagram



proprietario dell'opera. Nel caso in cui il costo accumulato sia maggiore del bilancio depositato, quest'ultimo verrà interamente trasferito al creatore dell'opera.

Rivalsa

Come è stato descritto precedentemente, quando il costo accumulato supera il bilancio depositato non si scatena alcun evento[38]. Solo in questo caso specifico dove, nonostante il contratto scaduto risulti ancora registrato, il proprietario dell'opera può avvalersi del diritto di rivalsa e, e richiamando il metodo `digichain.rinuncia()`, rimuovere personalmente il contratto sbloccando il pagamento verso di lui.

Come si può vedere dalla figura 9 il contratto *Digichain* estende la classe astratta *Dao*. Questo viene fatto per dare la possibilità ad ognuno di creare la propria implementazione Digichain, anche nella previsione futura di una possibile interazione tra varie istanze Digichain specializzate in opere specifiche. L'aggregazione *gestisce* mette in associazione un'istanza Digichain con 0 o n contratti Registrazione.

3.3 Possibili sviluppi

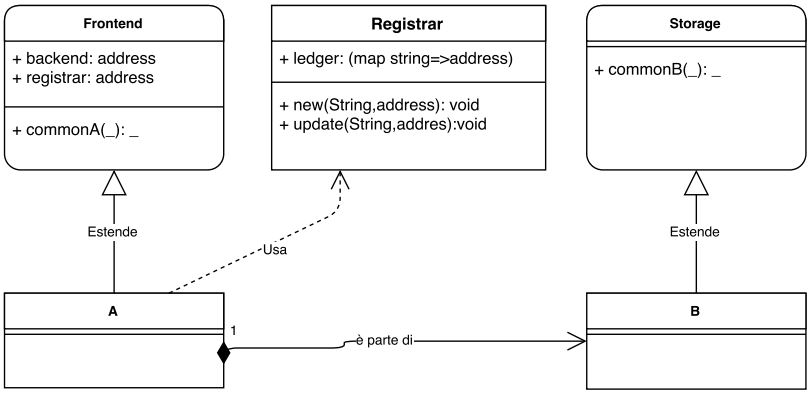
Ci sono diverse funzionalità che possono essere aggiunte nell'applicazione per includere nuove caratteristiche. E' possibile perseguirle con l'applicazione di pattern riguardanti i linguaggi orientati ai contratti, ma anche mediante l'utilizzo di software esterni.

3.3.1 System of Contracts

La prima interessante possibilità è l'impiego nell'applicazione del pattern *System of Contracts* che consente di modificare delle componenti di una Dapp mentre essa è in esecuzione. Come già riportato più volte, gli smart contract contengono codice immutabile, l'unica operazione di modifica concessa è la cancellazione (o suicidio) del contratto. Se nelle applicazioni testate in una private-net questo aspetto può

[38]In Ethereum, come è stato osservato precedentemente, non possono essere costruiti, senza l'ausilio di sistemi (centralizzati) esterni dei meccanismi che prevedano lo scatenarsi di eventi sulla base di stati dell'esecuzione dei contratti

Figura 10: System of contracts



sembrare irrilevante, nelle blockchain pubbliche il deploy di contratti ha un costo. Inoltre, se la nostra applicazione distribuita ha riscosso un notevole successo e molti utenti hanno salvato dati sul suo storage, o peggio ancora all'applicazione sono stati accreditati diversi Ether, l'esecuzione del suicidio del contratto manderebbe in fumo ogni dato. Quello che è possibile fare è dividere in moduli funzionali il proprio smart contract e disporre di un meccanismo che permetta l'aggiornamento del collegamento tra un'istanza di uno smart contract con un altro.

In figura 10 è possibile distinguere le classi o contratti:

Storage : è il contratto della gestione di un'aspetto fondamentale dell'applicazione che è stato potuto isolare dal contratto Frontend e da cui esso dipende.

Frontend : definisce la parte in comune della componente dipendente.

Registrar : è un contratto di gestione della registrazione di record composti da nomi e indirizzi, consente al proprietario del record di poter modificare l'indirizzo a piacimento.

Supponendo che il contratto Registrar è già operativo sulla blockchain, si scrive il codice per il contratto B, si esegue il deploy e l'indirizzo così ottenuto viene associato ad un nome simbolico nel contratto registrar. Viene caricato sulla blockchain il contratto A con l'indirizzo del contratto Registrar e il nome simbolico del contratto che implementa Storage. Nel contratto A ogni chiamata del metodo `A.commonA()` scatena l'interrogazione del Registrar (`registrar.ledger(backend)`) che come risultato da l'indirizzo sempre aggiornato del contratto Backend a cui deve invocare il metodo `commonB()`. In questo modo nel momento in cui il proprietario dell' applicazione vuole aggiornare solo la componente inclusa nel contratto B, caricherà un nuovo contratto sulla blockchain, salverà l'indirizzo di deploy nel registrar e da quel momento in poi A eseguirà il metodo del nuovo contratto.

3.3.2 Controlli temporizzati

sssec:controllitemporizzati)

Come affrontato nel sottocapitolo 2.4 non esiste un meccanismo, decentralizzato ed interno alla blockchain, in Ethereum per implementare un sistema temporizzato o che reagisca ad eventi basati sullo stato dell'esecuzione dei contratti. Una frequente

soluzione consiste nel fornire degli incentivi per spingere gli utenti a verificare loro stessi se si è verificato un determinato evento e impedendo che possano, allo stesso tempo, di "ingannare" il contratto per accaparrarsi l'incentivo.
La funzione che, in Digichain, avrebbe bisogno di essere automatizzata è la rivalsa del contratto legata al contratto Registrazione che può trovarsi in uno stato in cui il costo c accumulato nel tempo ha superato il bilancio b del contratto Registrazione. La chiamata `digichain.rivalsa(_intestatario)` è una funzione con side-effect il che comporta, anche in minima parte un costo. Questo scoraggia le persone a voler invocare la funzione, che scalerebbe degli Ether dai loro bilanci per sbloccare il pagamento verso qualcun'altro. Al contrario il controllo dell'ammontare della quantità di valuta che deve essere rimborsata viene eseguito tramite una call sullo stato del nodo locale, il che non comporta alcun costo in termini di transazioni (Si veda appendice B).

Per aggiungere questa funzionalità e, quindi, creare degli incentivi verso gli utenti che riscuoterebbero i bilanci tramite `.rivalsa(_intestatario)` si potrebbe aggiungere in fase di istanziazione dell'opera un nuovo valore *incentivo* che il creatore dell'opera è disposto a pagare per il servizio di rivalsa fornito. Da notare che attualmente la rivalsa è consentita solo al possessore dell'opera, ma per via della natura della funzione, incide in minima parte sul suo bilancio.

4 Conclusioni

Lo sviluppo dell'applicazione Digichain mi ha permesso di effettuare un primo approccio al linguaggio Solidity ed ai nuovi linguaggi orientati ai contratti. Studiando le possibili implementazioni già esistenti e i pattern riguardanti gli smart contract è stato chiaro che le tecnologie basate su blockchain ed in particolare Ethereum, sono realmente fruibili per applicazioni reali.

Se attualmente la rete di Ethereum ospita un carico moderatamente basso di transazioni al secondo[39], con i recenti sviluppi del nuovo sistema di consenso *Casper* e mediante lo sharding della blockchain, si stima di riuscire a moltiplicare di diverse centinaia il throughput attuale delle transazioni. Questo porterebbe la grande possibilità, nel prossimo futuro, alla sostituzione nelle applicazioni della terza parte fidata con uno smart contract autonomo, imparziale che non può essere soggetto a censura, frodi e periodi di downtime.

L'applicazione è funzionante e può essere un punto di partenza per ulteriori sviluppi riguardanti l'interazione mediante interfacce web. Una possibile interazione con i browser, attraverso plug-in, potrebbe permetterebbe la verifica automatica del diritto d'uso delle opere digitali contenute in un sito.

Se da una parte lo sviluppo del progetto Ethereum sia ancora nel pieno sviluppo dall'altra, le enormi potenzialità che introduce e la facilità con cui uno sviluppatore può creare un'applicazione decentralizzata basata su blockchain rappresentano una realtà da approfondire.

Sono certo che, nel futuro, sempre più attenzione verrà data a questa tecnologia, non solo dal punto di vista del mero aspetto speculativo delle criptovalute o di particolari *killer application*, ma le ricerche e gli sviluppi sulle proof, gli algoritmi di consenso e le nuove soluzioni getteranno le basi teoriche e pratiche dei nuovi sistemi decentralizzati.

[39] Attualmente durante l'impiego della pow dall' 1 alle 15 tx/s visionabile `https://etherchain.org/` e in accordo con `https://github.com/ethereum/wiki/wiki/Sharding-FAQ`

Appendice

Qui saranno contenute le appendici che nel corso della trattazione di questa tesi mi hanno permesso di eseguire operazioni particolari o che non hanno trovato posto nella trattazione.

A Visionare i nodi di una private-net Ethereum mediante strumenti visuali

Visionare lo stato di una blockchain non è un compito semplice da effettuare da riga di comando, si perde la visione d'insieme che spesso è la cosa che interessa. Potrebbe capitare di essere interessati all'hashrate della rete o al numero di transazioni che vengono effettuate in una particolare *testnet*. Per visionare le blockchain pubbliche esistono siti costruiti ad hoc [40]per controllare graficamente lo stato dei blocchi, dei parametri riguardanti i blocchi e le transazioni, per ricevere notifiche real-time sulle transazioni immesse nella rete, però possiamo osservare che non per tutte le criptovalute questo è disponibile.
Nel seguente caso vogliamo costruire un interfaccia grafica che permetta di osservare la blockchain privata di Ethereum. In rete, attualmente, è disponibile *etherchain-light*[41] che fornisce alcune delle funzionalità sopra citate e sfrutta l'interfaccia fornita da il light-node *Parity*[42].

Nel seguente elaborato per costruire la rete privata di nodi sono state usate istanze Geth e per collegare il nodo Parity ad esse è stato richiesto di effettuare i settaggi del blocco di genesi, del network-id e l'aggiunta degli endpoint `enode` dei nodi Geth. Purtroppo per via dell'incompatibilità tra Parity e Geth è stato imposto il bisogno di tradurre il blocco di genesi per Geth in un formato compatibile per Parity[43]. Non è stato raggiunto il fine di far comunicare il nodo Parity con i nodi Geth e quindi non si è potuta analizzare la blockchain privata. Si sospetta fortemente che i nodi delle due tipologie non si sincronizzino per via del blocco di genesi diverso.

Un applicativo parzialmente funzionante (durante la stesura di questo testo), ma che consente di visionare lo stato della rete a cui è connesso il nodo sottostante è *ETHExplorer*[44]. Esso non richiede il collegamento ad un nodo Parity, bensì si collega direttamente all'interfaccia RPC di Geth e quindi può essere utilizzato, senza aggiunte, nella topologia di rete di questo elaborato.

Una volta scaricato l'applicativo e settato l'ambiente per consentirne la sua funzione, si eseguono le istanze dei nodi e l'applicativo si connette di default all'endpoint `localhost:8545`. Navigando dal browser su `localhost:3000` è possibile avere delle statistiche sul tempo medio di creazione dei blocchi, l'hashrate attuale della rete e altri parametri, nonchè gli ultimi blocchi minati e le ultime transazioni effettuate.

[40]I più celebri: `https://blockchain.info/`, `https://chain.so`, `.blockr.io`, `https://etherchain.org/`, `http://moneroblocks.info/`
[41]`https://github.com/gobitfly/etherchain-light`
[42]`https://github.com/paritytech/parity`
[43]`https://github.com/keorn/parity-spec`
[44]`https://github.com/carsenk/explorer`

Attualmente questi strumenti non sono così completi rispetto ai siti di ispezione delle blockchain pubbliche (es. `blockchain.info`) ma forniscono un ottimo e immediato resoconto generale.

B Inizializzazione di nodi Geth

Per poter minare blocchi, includendo le transazioni e rispondendo alle chiamate via RPC, si necessita di una componente software che funga da nodo, le scelte sono molteplici. E' presente in rete *Testrpc*[45], un node scritto in puro javascript, il quale consente di predisporre delle interfacce e i comportamenti che permettono di simulare l'interazione con la blockchain ottenendo una rete privata composta da un solo miner. La grande quantità di funzioni dell'interfaccia che un nodo vero dovrebbe predisporre, rispetto a quella fornita da *Testrpc*, porta questo software ad essere abbandonato durante lo sviluppo. I programmi ufficiali che comunicano con la rete p2p, minano blocchi eseguendo la proof e predispongono un'interfaccia RPC completa possono essere trovati `https://github.com/ethereum/`. In questo caso si è voluto utilizzare *Geth*. Per simulare un applicazione su reti reali possiamo partecipare alla blockchain pubblica in rete semplicemente eseguendo:

```
$ geth
```

in modo tale da scaricare tutti i blocchi e abilitare il mining sfruttando la modalità full-node. Esistono anche altre possibilità tra cui scaricare la blockchain solo per analizzarla o per connettersi ad un light-client che connettendosi ad altri nodi permette di interagire con la blockchain.

Oltre a connetterci alla rete principale (main-net) c'è la possibilità che Geth possa collegarsi alla rete di testing di Ethereum (*Ropsten*) creata appositamente per permettere ai contratti di essere in esecuzione in una rete vera e propria senza eventuali costi . L'ultima possibilità è quella di creare una propria blockchain in un'ambiente isolato: una private-net.

Prima di poter far comunicare due nodi di Ethereum e farli minare sulla stessa blockchain bisogna comprendere con che criterio un nodo può sostituire la propria blockchain, sulla quale stava lavorando, con una che reputa migliore ottenuta da un nodo a cui è connesso.

Un nodo in ascolto nella rete può trovare nuovi nodi a cui connettersi per lavorare insieme, con il fine di estendere la medesima blockchain, solo se si verificano due ipotesi:

- i nodi devono essere nella stessa rete virtuale (*–networkid*).
- i blocchi di *genesi* dei due nodi devono essere identici perché un nodo abbandoni la propria blockchain e la sostituisca con un'altra.

Prima di tutto bisogna creare un account che sarà quello la cui chiave pubblica andrà inserita nell'etherbase/coinbase. Se in geth non viene passato nessun parametro $--$ *etherbase* viene preso il primo account trovato nel *keystore*[46].

[45] `https://github.com/ethereumjs/testrpc`
[46] https://github.com/ethereum/go-ethereum/wiki/Managing-Your-Accounts

Per prima cosa installiamo Geth da repo[47].

I blocchi in Ethereum sono formattati[48] in *JSON* il che permette la lettura e scrittura in modo semplice. La creazione del blocco di genesi può essere eseguita in forma testuale, non avendo bilanci pre-allocati presenta la seguente struttura:

```
//file CustomGensis.json
{
"nonce": "0x0000000000000042",
"timestamp": "0x0",
"parentHash":
   "0x0000000000000000000000000000000000000000000000000000000000000000",
"extraData": "0x0",
"gasLimit": "0x8000000",
"difficulty": "0x400",
"mixhash":
   "0x0000000000000000000000000000000000000000000000000000000000000000",
"coinbase": "0x3333333333333333333333333333333333333333",
"alloc": {
}
}
```

La scelta del valore assegnato ai parametri è per la maggior parte arbitraria, gli unici parametri che è significativo settare sono:

parentHash : solo per il blocco di genesi è uno scalare assegnato a zero.

alloc : è un vettore di record di $< pubkey, numWei >$ e ne definisce la pre-allocazione di valute ai rispettivi account.

difficulty : definisce la difficoltà relativa al primo blocco, la difficoltà dei blocchi successivi verrà determinata dal timestamp e dalla difficoltà del blocco precedente.

coinbase : o etherbase, è l'indirizzo a 160bit che definisce a quale indirizzo le fee e il reward del blocco debbano essere trasferiti. Per il blocco di genesi può essere arbitrario in quanto verrà determinato dinamicamente dagli address dei miner.

altro : ogni altro campo è arbitrario e possono essere modificati a piacere contribuendo ad aggiungere entropia per evitare che involontariamente due nodi minino sulla stessa blockchain.

Per simulare due processi che eseguono il mining sulla stessa macchina locale, dobbiamo tenere in considerazione che ogni istanza:

- deve poter creare le proprie strutture dati, *–datadir* deve avere un path assoluto unico.
- deve essere in ascolto su porte differenti (*–port* e *–rpcport*).

[47] `https://github.com/ethereum/go-ethereum`

[48] A seconda del software la formattazione dei blocchi varia, per esempio i blocchi compatibili con Parity non lo sono per Geth e viceversa.

- l'endpoint deve essere unico o può essere disabilitato (*–ipcpath* o *–ipcdisable*).

Per comodità andiamo a definire degli script per l'inizializzazione dei nodi (*init.sh*)e degli script per l'avvio dei processi Geth.

Una volta avviati i due processi disporremo di una console in ambiente JS con dei moduli precaricati che ci permettono di interagire con il nodo a cui siamo connessi e con la blockchain. I nodi di una rete privata dovranno essere registrati l'uno all'altro in modo tale da permettere che i peer scambino informazioni riguardanti le transazioni e i blocchi minati.

Da console possiamo creare nuovi account che risiederanno nel nostro keystore locale (se il keystore viene perso o la passphrase dimenticata non è possibile il recupero).

```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"newaddress 0x..."
```

Compilare da solc a EVMbytecode

I contratti vengono eseguiti sulla EVM di ethereum sottoforma di bytecode.

Per verificare che geth conosca la locazione del compilatore per il linguaggio solidity:

```
eth.getCompilers()
\\oppure eth.compile.solidity("")
```

E' possibile installarlo mediante **apt-get**. Oltre ad esso esiste *Solc-js* che permette di compilare file `*.sol` e di produrre `*.sol.js` direttamente importabili dalla console di Geth, questo consente di automatizzare eventuale testing.

Se correttamente installato da in output:

```
> eth.getCompilers()
["Solidity"]
```

E' possibile ora compilare uno o più contratti da console:

```
> comp= eth.compile.solidity("contract A {}")
{
   <stdin>:A: {
   code:
       "0x60606040523415600b57fe5b5b60338060196000396000f3006060604052",
   info: {
      abiDefinition: [],
      compilerOptions: "--combined-json bin,abi,userdoc,devdoc --add-std
          --optimize",
      compilerVersion: "0.4.10",
      developerDoc: {
         methods: {}
      },
      language: "Solidity",
      languageVersion: "0.4.10",
```

```
      source: "contract A {}",
      userDoc: {
         methods: {}
         }
      }
   }
}
```

Dall'output [49]si può vedere immediatamente come deve essere gestita una Dapp:

- **code**: è il codice in bytecode del contratto compilato, conterrà il codice di *init* e il codice di *body*.
- **info**: contiene i metadati compreso l'abi, il nome e altre informazioni che, unite all'indirizzo di deploy, permettono alla libreria Web3 di predisporre di oggetti per l'interazione con i contratti deploiati.

B.1 Deploy ed interazione del contratto

Una volta che i nodi sono funzionanti ed è possibile compilare correttamente i contratti, può essere eseguito il deploy degli smart contract e interagirci.

La libreria Web3 svolge il processo di collegare l'indirizzo del contratto che risiede (conosciuto solo dopo la compilazione e il deploy) e l'interfaccia abi che predispone i metodi che possiamo interrogare.

Una delle tante proprierty dell'oggetto `eth` caricato nella console di Geth è `eth.contract(abi)` che consente di ottenere un oggetto fittizio linkato ad abi. Su questo oggetto a seconda delle operazioni da effettuare vengono predisposti due proprietà dell'oggetto:

.new() : consente di eseguire una transazione specificandone i campi ed inserendo nel campo `data`: il codice del contratto. Se la transazione avverrà, la callback sarà chiamata e l'istanza collegata al contratto sulla blockchain sarà restituita.

.at() : consente di specificare un indirizzo e di ritornare un istanza del contratto che risiede sulla catena.

In tutti e due i casi, ottenuta l'istanza sarà possibile invocare su di essa il nome della funzione (contenuta nell'abi) e specificare se si vuole eseguire una call o una transazioneC.

B.2 Mining

La forma più semplice di mining da effettuare con Geth è il mining via CPU e senza dover settare l'ambiente. Nella versione *Frontier* l'intervallo tra un blocco e l'altro è bilanciato sui 15-17 secondi, bilanciato dalla difficoltà settata dalla PoW Ethash.

[49]Nel caso specifico di solc versione 0.4.10 la compilazione produce un errore sulle label json utilizzando output["<stdin>:NOMEDELCONTRATTO"] `http://ethereum.stackexchange.com/questions/11912/unable-to-define-greetercontract-in-the-greeter-tutorial-breaking-change-in-sol/11915#11915`

```
> miner.start(1)
true

>eth.hashrate
4
```

Come argomento del comando `.start()` viene specificato il numero di thread che il processo usa per il mining. Nel caso un nuovo peer si unisce al mining, della medesima blockchain della stessa rete, automaticamente la difficoltà viene alzata.

C SendTransaction (side-effect call) e call

Formalmente i metodi `.sendTransaction()` e `.call()` che possiamo eseguire su un istanza `eth.contract` mandando entrambi due chiamate all'interfaccia rpc del nodo a cui siamo collegati tramite l'interfaccia *web3* (e i relativi moduli) che è caricata nella console JS. Il primo metodo esegue una transazione: viene passata all'interfaccia il mittente della transazione firmandola, il payload data, il nonce e tutti gli altri parametri di una transazione. L'esecuzione del comando ritornerà sempre l'hash della tx. Viceversa il metodo `.call()` può essere invocato su un metodo che non produce cambiament e viene eseguita dal nodo sottostante, ritornerà un valore se sarà specificato le clausole `constant returns`[50]. Questo comporta:

- la transazione prima di produrre un cambiamento di stato deve essere condivisa con gli altri nodi e inclusa in un blocco valido, non è possibile a priori, sapere ne quando ne se sarà inclusa.
- la call ispeziona solo lo stato dei contratti nella blockchain e può essere eseguita immediatamente, non produce effetti, non deve essere inclusa in un blocco e ciò comporta che la sua esecuzione non ha un costo in gas.

Ogni transazione nella piattaforma Ethereum ha un costo minino di 21000 gas (che moltiplicato per il gasPrice fornisce il costo in Wei della transazione)[51].

D Eventi, log e return

Ogni contratto ha associato una struttura dati che contiene le *entry log* emesse da un contratto. In base a [16] un log è formato dall' indirizzo del logger, una serie di campi di 32 byte e un campo data contenente un numero arbitrario di byte.

Come risultato di un invocazione di un evento `event NomeEvento(_type _param)` viene emesso e registrato un log. Ovviamente ogni singolo log e ogni byte in esso contenuto costano gas, ma se la notifica è indispensabile si ritiene obbligato la scrittura dei log[52].

[50] L'identificativo `constant` dichiara che la funzione non ha side-effect, questo rafforza l'analisi formale che può essere fatta

[51] Per la documentazione ufficiale: `https://github.com/ethereum/wiki/wiki/JavaScript-API#web3ethcall`

[52] I log costano 375 gas ognuno e 8 gas ogni byte aggiunto contro i 20.000 gas per ogni byte di storage del contratto

L'esecuzione di una transazione dalla console di Geth,visto che restituisce solo l'hash della tx inviata per le ragione dell'appendice B, richiede per la restituzione di un valore dalla sua esecuzione l'utilizzo di eventi.

La classe web3.eth.contract permette di porsi in ascolto sul log di uno specifico contratto, specificando uno o tutti gli eventi che predispone l'interfaccia abi[53].

[53] `https://github.com/ethereum/wiki/wiki/JavaScript-API#contract-events`

References

[1] Gareth W Peters and Efstathios Panayi. Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money. In *Banking Beyond Banks and Money*, pages 239–278. Springer, 2016.

[2] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 2001.

[3] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 1985.

[4] Idit Keidar and Sergio Rajsbaum. On the cost of fault-tolerant consensus when there are no faults: preliminary version. *ACM SIGACT News*, 2001.

[5] Andrew S Tanenbaum and Maarten Van Steen. *Sistemi distribuiti. Principi e paradigmi*. Pearson Italia Spa, 2007.

[6] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[7] Wei Dai. Bmoney e la crypto-anarchy. 2008.

[8] Adam Back et al. Hashcash—a denial of service counter-measure, 2002. *Available from World Wide Web: http://www. hashcash. org/papers/hashcash. pdf*, 2007.

[9] Debin Liu and L Jean Camp. Proof of work can work. In *WEIS*, 2006.

[10] H Massias, X Serret Avila, and J-J Quisquater. Design of a secure timestamping service with minimal trust requirement. In *the 20th Symposium on Information Theory in the Benelux*. Citeseer, 1999.

[11] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151, 2014.

[12] Colin Percival. Stronger key derivation via sequential memory-hard functions. *Self-published*, pages 1–16, 2009.

[13] P4Titan. Slimcoin: A peer-to-peer crypto-currency with proof-of-burn. 2014.

[14] Lesedi Seforo. Currency and forex trading in the 21st century: bitcoin: features-interest. *TaxTalk*.

[15] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. Certcoin: A namecoin based decentralized authentication system. *Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep*, 6, 2014.

[16] Gavin Wood. Ethereum yellow paper, 2014.