Bitcoin: una descrizione architetturale

Marco Di Nicola

Abstract—Proposta nel 2008 da Satoshi Nakamoto, Bitcoin è nota come la prima valuta elettronica che non richieda l'appoggio o mediazione di alcuna istituzione finanziaria per condurre transazioni e produrre nuove monete.
Tutto questo è possibile sfruttando un'architettura peer-to-peer totalmente decentralizzata, basata su principi di crittografia asimmetrica e schemi proof-of-work per gestire lo scambio e creazione di moneta, garantendo protezione contro tentativi di contraffazione delle transazioni effettuate e attacchi double-spending.
Questo documento si propone di fornire una descrizione di Bitcoin da un punto di vista architetturale, strutturandosi come segue.
Dopo averne introdotto i principi operativi, sarà descritto il contesto di utilizzo di Bitcoin e i principali casi d'uso, discutendo quindi le proprietà di maggiore interesse (prime fra tutte la sicurezza, mantenimento di privacy e anonimato e scalabilità del sistema).
Prendendo in analisi un prototipo di software client (l'originale, concepito come proof of concept del sistema), sarà descritta la sua struttura, con componenti principali e relative funzioni e interazioni, rappresentando in maniera più dettagliata il comportamento del sistema; verranno anche descritte alcune derivazioni del modello di client peer-to-peer originale, più vicine ad uno stile client-server.
Sarà quindi discusso il razionale che sta dietro alla progettazione di una tale architettura, mostrando il risultato di analisi svolte su alcuni dei suoi punti critici e presentando note alternative a Bitcoin o sue estensioni, fra i sistemi di valuta elettronica (tra i quali spiccano Ripple ed Ethereum).

1 Introduzione

Bitcoin è forse una delle valute elettroniche maggiormente adottate al giorno d'oggi: un migliaio di compagnie e singoli individui sono elencati nella sezione Trade del wiki ufficiale [12], fra gli enti che la accettano come metodo di pagamento per beni materiali o servizi online (l'Università di Nicosia, capitale di Cipro, permette il pagamento delle tasse universitarie tramite bitcoins); in alcuni paesi sono persino disponibili sportelli ATM per il cambio con denaro materiale.
I bitcoins (BTC) in circolazione al tempo della stesura di questo documento sono più di 12,605,150; il valore medio di una singola moneta è pari a 448.60 dollari americani (327.40 euro), contro i 13.30 stimati a Gennaio del 2013.

La ragione di questa diffusione va cercata nella natura decentralizzata dei meccanismi alla base di produzione e scambio di monete, nonché nelle forti garanzie di affidabilità del sistema: non è possibile spendere due volte, contraffare o rubare bitcoins.

I principi e metodi alla base di questi meccanismi sono descritti nel paper originale pubblicato su The Cryptography Mailing List presso il sito metzdowd.com nel 2008 [Nak08], da un individuo noto sotto lo pseudonimo di Satoshi Nakamoto, la cui identità rimane tuttora ignota.
Nakamoto propose l'uso di una rete peer-to-peer pura per realizzare un sistema di transazioni elettroniche, libero da requisiti di *trust* nelle entità intermedie che sono alla base degli odierni sistemi di pagamento (banche, società di carte di credito) e basato su una valuta generata autonomamente dai partecipanti al sistema stesso; come proof of concept di questo modello, Nakamoto avviò lo sviluppo del client originale (Bitcoin Core), di natura open source e correntemente mantenuto da un vasto gruppo di sviluppatori dediti alla causa.
La prima transazione fu effettuata nel Gennaio del 2009 e, per il Giugno 2011, si osservò la presenza di ben 6.5 milioni di bitcoins, in circolo fra più di 10,000 utenti stimati.
In merito alla terminologia utilizzata in questo documento: il termine "Bitcoin" farà riferimen-

to alla valuta in se, al sistema che ne è alla base o alla rete peer-to-peer che ne fa uso e relativo protocollo di comunicazione, mentre con "bitcoin" sarà denotato un quantitativo di denaro virtuale trasferibile dagli utenti.
Nella sezione successiva sarà fornito un breve sunto dei principi operativi alla base di Bitcoin, discussi ed esplicati maggiormente nel resto del documento.

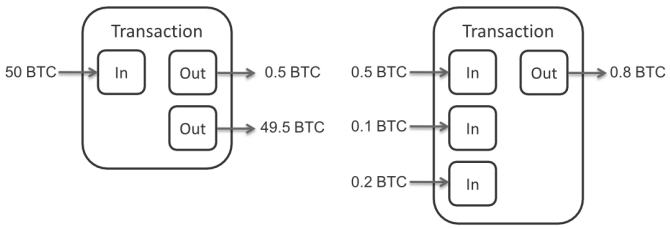

Figura 1. Esempi di transazioni Bitcoin con multipli input/output

2 PRINCIPI OPERATIVI

Al fine di eliminare il requisito di trust di un utente in altre entità, Bitcoin fonda i suoi principi operativi sull'uso di crittografia asimmetrica per firme digitali e funzioni di hashing *one-way* (a senso unico, non invertibili); questo ne fa la prima *criptovaluta* effettivamente implementata.
Nella rete Bitcoin ogni utente P può possedere una o più identità (o "account"), definite come coppie (k_{pub}, k_{priv}) di chiavi pubbliche e private per crittografia asimmetrica: se da k_{pub} viene derivato l'indirizzo Bitcoin pubblico di P, k_{priv} è conservata nel suo portafoglio virtuale (wallet) ed è utilizzata da P per spendere i bitcoins ricevuti.
Una moneta è definita come il risultato di una catena di transazioni, firmate digitalmente da chi le emette, tra i diversi partecipanti alla rete peer-to-peer.
Ogni nuova transazione da P a Q fa riferimento ad una o più transazioni precedenti, contenenti il quantitativo di bitcoins precedentemente trasferiti a P da altri utenti e conservate localmente da ogni partecipante alla rete peer-to-peer.
Un messaggio contenente i riferimenti in questione (*inputs*), l'ammontare di bitcoins da trasferire e il digest della chiave pubblica (indirizzo) di Q (*output*), viene emesso in broadcast sulla rete e propagato dai nodi; è opportuno menzionare che l'unità atomica in cui è possibile scomporre un bitcoin è il *satoshi*, tale che $1\ BTC = 100,000,000\ satoshi$.
Tutte le transazioni da/a P sono conservate da tutti i nodi della rete, permettendo a Q di validare pubblicamente quella ricevuta attraverso un consenso collettivo e a P di osservarne l'accettazione: in un certo senso, le monete possedute da ogni utente sono mantenute da tutti gli altri.
In ogni caso sarà solo Q a poter fare riferimento alla specifica transazione a suo beneficio, nell'atto di trasferire bitcoins ad altri utenti, mediante uso della sua chiave privata e quindi firma digitale, verificabile grazie alla corrispettiva chiave pubblica precedentemente allegata da P alla transazione.
In sostanza, l'unico riferimento ai bitcoins posseduti da Q è la somma degli importi delle transazioni a lui destinate e non ancora utilizzate come input da altre.

Resta il problema di evitare che P conduca quasi simultaneamente una seconda transazione diretta a R, facendo riferimento alle stesse transazioni precedenti (stessi inputs) usate per Q.
Questo attacco è una falla di molti sistemi di moneta elettronica, ed è denominato *double-spending*.
Un tipico mezzo di protezione consiste nell'introdurre un'autorità certificata che attesti la validità di ogni transazione controllando che non faccia riferimento ai medesimi inputs di un'altra, ma naturalmente questo vanificherebbe l'intero paradigma di decentralizzazione adottato da Bitcoin.
La soluzione utilizzata consiste invece nella sopracitata disponibilità pubblica di tutte le transazioni e in un meccanismo che permetta ai nodi della rete di concordare (eventualmente) su quale delle due transazioni sia effettiva.
Tale meccanismo risiede in un timestamp server distribuito che possa gestire una catena di blocchi di transazioni, tali che ogni blocco k conservi nella propria intestazione un

riferimento alle transazioni contenute in esso e un valore $hash_{k-1}$ derivato da hashing del blocco precedente, il quale a sua volta conterrà $hash_{k-2}$, etc.; questa lista concatenata di blocchi è denominata *block chain*.
Una volta costruita questa catena di blocchi, della quale ogni nodo della rete mantiene la propria versione locale, la semplice inclusione di un blocco al suo interno potrebbe rappresentare l'accettazione, da parte del singolo nodo, di tutte le transazioni contenute nel blocco. Nuove transazioni sono ammesse all'interno di un blocco solo se valide: sintatticamente corrette, non duplicate e che non facciano riferimento ad inputs inesistenti o già indicati da altre.
In fase di verifica dell'accettazione di una specifica transazione, si potrebbe controllare che un numero sufficiente di nodi la possegga all'interno di un blocco nella propria versione della block chain, la quale si suppone essere coerente con quelle mantenute dal resto della rete Bitcoin.
Rimane il problema di evitare che un attaccante capace di allocare molti nodi (indirizzi IP) distinti sia in grado di imporre il suo voto su quello di nodi onesti, fornendo una sua versione alterata della chain (e quindi delle transazioni al suo interno).
Questo problema può essere risolto rendendo computazionalmente "costosa" la creazione di un blocco valido da aggiungere alla chain.
Lo schema utilizzato da Bitcoin è il seguente: determinati nodi della rete (*miners*) raccolgono tutte le transazioni non ancora accettate nel proprio blocco k e cercano una cosiddetta *proof-of-work*, ossia compiono uno sforzo computazionale che consiste nel cercare un nonce s tale che la funzione di hashing utilizzata (SHA-256), applicata all'intestazione del blocco comprensiva del nonce, risulti in un digest la cui rappresentazione binaria a 256 bits abbia un prefisso composto da un certo numero di bits 0.
Il numero di bits 0 viene determinato in maniera uniforme da tutti i nodi della rete e calcolato in base al numero di blocchi aggiunti alla chain in un intervallo di 2 settimane di tempo.
Questo problema può essere riformulato come quello di trovare un nonce tale che l'hashing dell'intestazione del blocco sia minore di un

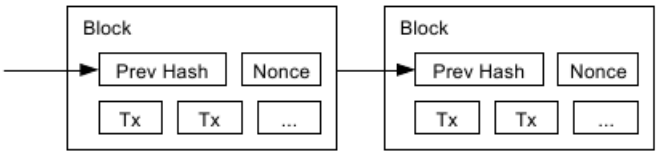

Figura 2. Block chain

certo numero, denominato *target*, modificato in modo dinamico ed uniforme (secondo uno schema predefinito) dai nodi della rete.
Basti pensare che, in un istante del Settembre 2011, il target richiese una media di 7,539,609,386,691,347 tentativi di hashing per trovare la proof-of-work.
Una volta ottenuto il giusto valore di s, il miner annuncia la sua soluzione al resto della rete emettendo in broadcast il blocco k, comprensivo di $hash_{k-1}$ e s nella propria intestazione. I nodi che ricevono k lo accettano e inseriscono nella propria versione della block chain solo se tutte le transazioni contenute in esso sono valide, il valore di $hash_{k-1}$ corrisponde effettivamente al digest dell'ultimo blocco inserito e l'applicazione di SHA-256 all'intestazione con il nonce dato risulta in un prefisso del giusto numero di bits 0; il vantaggio delle funzioni di hashing one-way risiede proprio nel fatto che, al contrario della ricerca della soluzione, la verifica di quest'ultima sia pressoché istantanea.
L'accettazione prevede anche la ritrasmissione del blocco in questione ad altri nodi (i quali ripeteranno il medesimo processo di validazione).
Il punto di forza del meccanismo di proof-of-work è che per contraffare un blocco k, per esempio modificando le transazioni in esso, occorre modificare anche tutti i successivi (in quanto le loro intestazioni dipenderanno dal digest di k) e trovare una nuova proof-of-work per ognuno di essi: questo risulta in uno sforzo computazionale non banale, esponenziale nel numero di bits 0 del prefisso ricercato e nel numero di blocchi prodotti allo stesso tempo da miners onesti.
Nel probabile caso di fork (diverse biforcazioni di blocchi a partire da $k-1$) nella chain, i nodi sceglieranno eventualmente di estendere

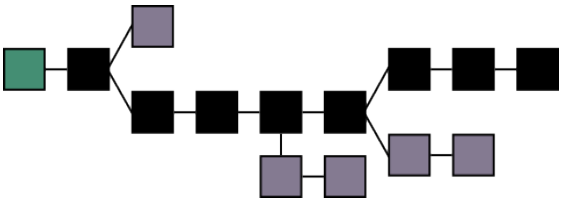

Figura 3. Più forks nella block chain, eventualmente scartate (la nera rimane la principale)

il ramo con maggiore lunghezza, scartando l'altro.
La lunghezza in questione non dipende dal numero di blocchi in una biforcazione, ma dalla difficoltà complessiva (*difficulty*) nel risolvere la proof-of-work per ognuno di essi; poiché il target è inserito nell'intestazione di un blocco, in base ad esso è possibile calcolare la difficulty della soluzione.
In sostanza, fintanto che la potenza computazionale complessiva dei nodi onesti supera quella degli eventuali disonesti, solo i blocchi contenenti transazioni non contraffatte entreranno a far parte della block chain, venendo quindi riconosciute come valide.
Le biforcazioni, causate da blocchi prodotti concorrentemente da nodi diversi a partire dalla medesima estensione della chain, saranno eventualmente scartate.
Ne consegue che un tentativo di double-spending da parte di P potrebbe, nel peggiore dei casi, risultare in due probabili biforcazioni della chain: una contenente un blocco con la transazione diretta a Q, l'altra contenente un blocco con quella diretta ad R. Eventualmente i nodi che concatenano blocchi ad una delle due prevarrebbero, provocando lo scarto definitivo dell'altra (eliminandone quindi ogni traccia e vanificando il tentativo di double-spending).
Il procedimento appena descritto include anche la coniazione di nuove monete: la prima transazione contenuta nel blocco k appena creato è di un tipo speciale che assegna all'indirizzo del miner un determinato quantitativo fisso di bitcoins; oltre a questo, sono aggiunte le eventuali *transaction fees* specificate dalle altre transazioni incluse nel blocco, utilizzate come incentivo per i miners.
Il primo blocco della block chain, soprannominato *genesis block*, è stato creato il 3 Gennaio 2009 e ha prodotto i primi 50 bitcoins, messi in circolo dall'indirizzo a cui erano destinati (probabilmente generato da Satoshi Nakamoto).
Eventualmente il processo di mining può essere svolto in gruppo: più miners possono cooperare nella creazione di un blocco valido utilizzando un mining pool, dividendo poi i bitcoins prodotti dal blocco creato.

Riassumendo: il bilancio di un utente è dato dall'importo delle transazioni a lui dirette e non spese (quindi non referenziate come input da altre transazioni valide).
La conferma di un avvenuto pagamento è data dal processo collettivo di mining, il quale garantisce, attraverso l'utilizzo di funzioni di hashing one-way e ricerca di proof-of-work, la validità di un blocco accettato da un numero sufficiente di nodi all'interno della propria block chain.
I miners sono ricompensati dello sforzo computazionale e del consumo di energia elettrica attraverso la creazione di nuovi bitcoins intestati ad essi, specificati da una transazione speciale destinata ad essi ed inclusa nel blocco appena prodotto.

3 ARCHITETTURA

In questa sezione sarà presentata l'architettura del sistema Bitcoin, fornendo una visione più dettagliata delle sue ragioni d'essere e contesto di utilizzo.
Sarà fornita la descrizione della struttura di un client (rappresentante un peer all'interno della rete) ed i suoi componenti fondamentali, evidenziando le differenze concettuali fra le implementazioni maggiormente diffuse.
Infine, sarà descritto il comportamento e le funzioni di ogni singolo componente dell'architettura, ponendo enfasi sui più rilevanti aspetti progettuali e discutendo alcune proprietà interessanti, quali sicurezza e garanzia di anonimato.

3.1 Contesto

Molti utenti utilizzano Bitcoin per ragioni ideologiche, politiche e soprattutto pratiche.

Per quanto riguarda le prime, basti pensare al concetto di proprietà popolare della moneta: lo scambio di bitcoins non può essere regolamentato o tassato in maniera effettiva, poiché non esiste un sistema di produzione centrale o un operatore di rete attraverso il quale transitino le transazioni.
Venendo alle ragioni pratiche, sicuramente Bitcoin non costituisce l'unica alternativa, fra i sistemi che permettono l'utilizzo di valute elettroniche (la descrizione di alcune di esse, comprensiva di vantaggi, problemi e differenze con Bitcoin, è rimandata alla sezione 5), ma la ragione che ha portato al suo successo e diffusione, come precedentemente menzionato, è il fatto che funzioni correttamente ormai da più di 5 anni, fornendo ad un utente medio numerose motivazioni per il suo utilizzo:

- Creare un "account" consiste nella semplice installazione di un software.
- Permette di effettuare pagamenti con rapidità: dal momento in cui la transazione è emessa, più nodi saranno impegnati a lavorare sulla sua validazione, così che possa essere inserita nella block chain, in un procedimento che è stimato durare 10 minuti circa.
  Esistono anche le cosiddette transazioni *zero-confirmation*: accettate istantaneamente dal ricevente, senza attenderne l'inserimento definitivo nella block chain.
  In entrambi i casi è meno tempo di quanto richieda un qualunque trasferimento fra banche.
- É economico: non è necessario pagare per il privilegio di avere transazioni istantanee (come con quelle via carta di credito) e le transaction fees sono minime o nulle.
- É impossibile che un qualunque governo possa attaccarlo, impedirne l'utilizzo o cancellare le transazioni, in modo simile a quanto accade con altre risorse condivise attraverso la rete (accesso ai record DNS, per esempio), poiché il sistema è totalmente decentralizzato.
- Non si possono annullare le transazioni, ovvero non esiste lo storno di addebito. Questo impedisce ai venditori di praticare truffe richiedendo l'annullamento delle transazioni, ma d'altro canto non tutela contro compratori disonesti.
- Non è necessario fornire alcuna informazione personale, per effettuare pagamenti.
- Le monete sono prodotte dagli utenti stessi, i quali investono tempo di calcolo dei loro processori e corrente elettrica nel procedimento; la ricompensa monetaria è determinata dal sistema ed è utilizzata come incentivo al contributo individuale al mantenimento della coerenza globale.
- É privo di inflazione: il protocollo specifica che il numero di bitcoins prodotti dalla creazione di un blocco diminuiranno fino a lasciare le sole transaction fees come guadagno per i miners, nel 2140.
  Tutt'al più vi è rischio di deflazione, quando questo limite sarà raggiunto.
- Garantisce una forma di anonimato (maggiori dettagli a riguardo nella sezione 3.2.2) per gli utenti.
- Ultimo ma non meno importante: non richiede alcuna forma di trust in entità intermedie (banche, società di carte di credito) e l'"account" di ogni utente (inteso come coppie di chiavi) è conservato unicamente da lui stesso.

Il diagramma in figura 4 mostra i normali casi d'uso per Bitcoin; ogni nodo della rete può essere un miner, sebbene ultimamente questa distinzione con l'utente normale (il quale si limita ad effettuare transazioni) sia più netta (vedi 3.3.5).
La creazione di nuove identità, intese come coppie di chiavi e relativo indirizzo Bitcoin, è spesso condotta in maniera del tutto trasparente dal software client utilizzato.
Esistono anche alcuni providers di servizi relativi a Bitcoin (i più famosi sono MyBitcoin e Mt.Gox), i quali conducono transazioni per conto di più utenti, sollevandoli dalla gestione delle proprie chiavi private e utilizzo di risorse computazionali; le implicazioni nell'utilizzo di questi servizi sarà discusso nelle sezioni successive.
Nonostante i fattori che promuovono l'utilizzo di Bitcoin all'interno del mercato, ne esistono numerosi detrattori, specialmente in merito all'ultimo punto della lista precedente, quindi

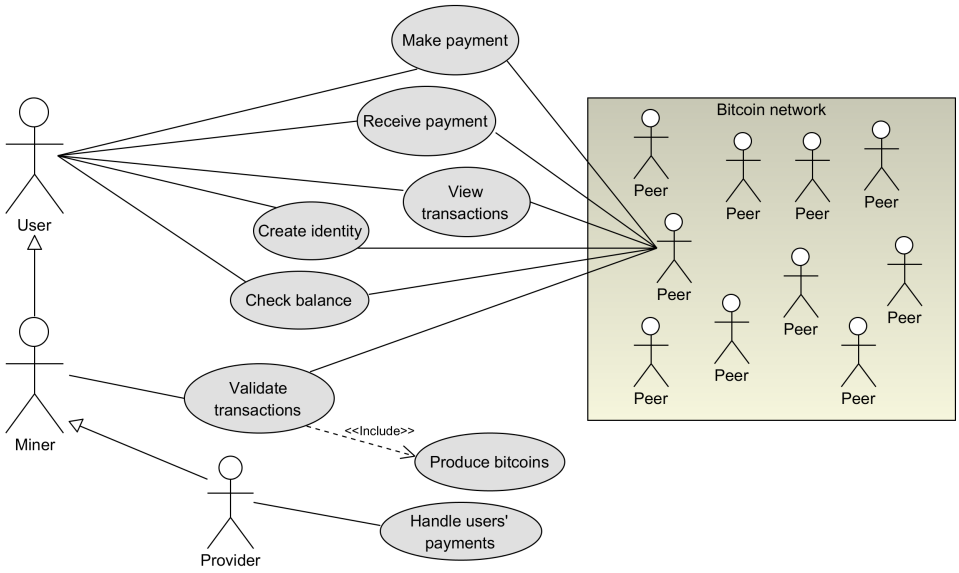

Figura 4. Casi d'uso per Bitcoin

questioni relative all'impossibilità per una istituzione finanziaria di esercitarvi alcuna forma di controllo.
É infatti proprio per merito della sua natura decentralizzata e garanzie di anonimato che Bitcoin rappresenta il mezzo ideale per acquistare droga o riciclare denaro la cui provenienza è illecita.
Citando le parole dello scrittore Charlie Stross, noto detrattore di Bitcoin, in un post online: "*Bitcoin sembra progettato come un'arma destinata a colpire le banche centrali che emettono moneta, in omaggio a obiettivi politici ispirati all'ultraliberismo: lo scopo è limitare la capacità degli Stati di riscuotere le tasse e monitorare le transizioni finanziarie dei loro cittadini*" [14].

3.2 Proprietà

Segue un'analisi più approfondita delle proprietà rilevanti del sistema Bitcoin, alcune delle quali già menzionate in 3.1.
L'**affidabilità** del sistema è garantita dalla sua natura peer-to-peer decentralizzata: fintanto che rimarrà attiva una porzione della rete Bitcoin (stimata a più di 3,342,922 nodi da blockchain.info) sarà possibile condurre transazioni e vederle confermate.
L'**usabilità** dei software disponibili per interfacciarsi con il sistema consente di condurre qualunque operazione, dalla creazione di una nuova identità (coppia di chiavi) all'emissione di una transazione, con estrema semplicità. Molti di questi software godono di **portabilità** multi-piattaforma, grazie all'utilizzo di librerie specializzate (Boost e Qt per il client originale Bitcoin Core, scritto in C++) o esecuzione su Java Virtual Machine (Bitcoinj).
Il tempo medio necessario a portare a termine una transazione, ossia a vederla confermata dal sistema, rappresenta il "tempo di risposta" del sistema per il caso d'uso principale; questo può essere quantificato mediante osservazione della block chain: si rimanda alla sezione 4.
Una proprietà più critica è la **scalabilità**. Ci si potrebbe chiedere se in futuro, con un notevole aumento nella frequenza delle transazioni e blocchi in transito, nonché con un ulteriore aumento nelle dimensioni della block chain, rimanga possibile gestire il carico del protocollo per dispositivi personal computer.
Purtroppo un'ulteriore aggravante è data dai cosiddetti *dormant coins*: output di transazioni nella chain non spesi e non spendibili, in quanto i loro possessori hanno perso le chiavi private o sono morti; non c'è modo di eliminare questo potenzialmente enorme "peso morto" dai dati che i milioni di nodi della rete Bitcoin processano quotidianamente.
Ad oggi ci sono numerose proposte per migliorare questa situazione: congelare parte della

block chain, utilizzare metodi semplificati per la verifica del pagamento (vedi 3.3.2) o semplicemente rimuovere tutte le transazioni con output già spesi.

3.2.1 Sicurezza

La sicurezza nella gestione dei propri soldi costituisce sicuramente il requisito non funzionale più importante, nell'architettura di un sistema per valute virtuali.
Come precedentemente menzionato, in Bitcoin ogni utente conserva i soldi di ogni altro utente, in quanto questi sono rappresentati dalle transazioni con output non spesi, conservate all'interno della block chain e mantenute dall'intera rete.
Ciò che consente ad uno specifico utente di reclamare il possesso di un determinato quantitativo di bitcoins, spendendoli nell'emissione di nuove transazioni, è una firma digitale per mezzo della sua chiave privata.
Poiché l'alterazione di una o più transazioni di un utente risulterebbe impossibile per un eventuale attaccante, grazie a quanto appena detto e al meccanismo di proof-of-work, ne deriva che l'unica potenziale falla di sicurezza risieda unicamente nella gestione delle chiavi private.
Tale gestione è stata notevolmente migliorata negli ultimi anni, al punto di implementare tecniche più sofisticate di manipolazione del wallet (vedi 3.4.2).
Inoltre, Bitcoin prevede la possibilità di effettuare transazioni che richiedano più firme (*multi-signature*) per poter essere poi utilizzate dai beneficiari.
Un utente potrebbe conservare le proprie n chiavi private su diversi dispositivi, in modo tale che se solo $n-1$ sono compromesse non sia possibile spendere i suoi bitcoins. Naturalmente ciò presenta un costo aggiunto di complessità per gestire lo scambio delle firme tra i dispositivi.
Maggiori dettagli sulle modalità di utilizzo delle transazioni da parte dei beneficiari saranno forniti in 3.5.3.
A parte questo, rimane la possibilità che uno o più degli schemi crittografici o funzioni di hashing utilizzate sia infranto in futuro, rivelando nuove vulnerabilità (basti pensare al caso Heartbleed): questo rappresenta, in un certo senso, un limite intrinseco dei sistemi di sicurezza informatica che si affidano a tali meccanismi.
Un altro aspetto di sicurezza può risiedere nel come il sistema riesca a tutelare gli utenti da venditori o acquirenti disonesti. Non è possibile annullare transazioni e da ciò ne deriva che una volta effettuato un pagamento, in caso di dispute fra le due parti non sia possibile risolverle, non avendo un ente centralizzato al quale rivolgersi.
Tuttavia grazie al meccanismo di transazioni multi-signature è possibile introdurre un *broker* intermediario e suddividere l'acquisto in due fasi:

1) Una transazione dall'acquirente a due indirizzi: se stesso ed il broker.
2) Una seconda transazione emessa a partire dalla prima, sfruttando firme di acquirente e broker, destinata al venditore.

Operando questa suddivisione è possibile bloccare i bitcoins trasferiti in un limbo fra le due transazioni, in attesa che il broker (nel quale entrambe le parti devono riporre fiducia) possa effettuare un controllo sui beni venduti prima di partecipare all'inoltro del pagamento verso il venditore.
Se una delle tre parti tenta di violare l'accordo, i bitcoins trasferiti rimarranno semplicemente inaccessibili ad ognuno di essi e in attesa di almeno 2 firme digitali.

3.2.2 Privacy

Una delle proprietà più rilevanti e controverse, nell'ambito delle architetture di sistemi per valute virtuali, è la garanzia di anonimato agli utenti che effettuano transazioni: la difficoltà (o impossibilità) nell'associare queste ultime alle identità "reali" (intese come generalità, indirizzi IP e quindi posizioni geografiche o altri dati) degli utenti coinvolti.
Il meccanismo con cui Bitcoin garantisce questa proprietà è molto analizzato in letteratura: a partire da alcuni cenni nel paper originale ([Nak08]), fino a studi più recenti ([RH13], [AKR+13], [OKH13]).

Nel modello bancario tradizionale si garantisce un certo livello di privacy limitando l'accesso alle informazioni sui pagamenti alle parti coinvolte ed eventualmente a terze parti fidate. In Bitcoin tutte le transazioni sono pubbliche, ma ciò non preclude la possibilità di mantenere anonime le chiavi pubbliche utilizzate, eventualmente associando una nuova coppia di chiavi ad ogni transazione effettuata.

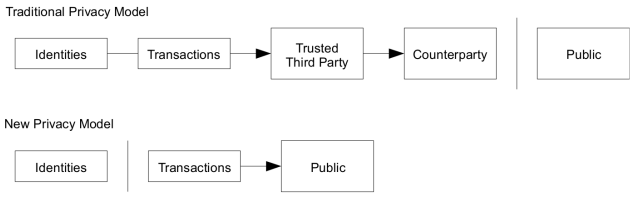

Figura 5. Modello di privacy in Bitcoin

Questa separazione fra identità degli utenti e chiavi pubbliche corrisponde al concetto di *Pseudonymity*: mantenere anonimato utilizzando identità fittizie per svolgere le operazioni necessarie.
Nonostante questa separazione, in letteratura è suggerita la possibilità di associare diverse chiavi pubbliche a singoli utenti, costruendo un grafo delle transazioni a partire dalla block chain e analizzandone la struttura ([RH13], [OKH13]).
Considerando i nodi di questo grafo come transazioni e gli archi come collegamenti di tipo input/output fra di esse, è possibile collassare diversi indirizzi Bitcoin in singole entità, considerando che se una transazione fa riferimento a diversi input gli indirizzi a cui i corrispettivi outputs sono destinati apparterranno probabilmente alla medesima entità.
Viene ventilata la possibilità di dedurre questi collegamenti anche in base a diversi campi di output in una transazione e relativi importi [AKR+13], considerando che uno di essi può risultare in un "resto" che l'utente invia a se stesso (maggiori dettagli in 3.5.3); tuttavia molti clients creano un nuovo indirizzo appositamente per trasferire il resto, al fine di ridurre la possibilità di inferire collegamenti.
Una volta definito un mapping uno a molti fra singole entità e insiemi di indirizzi Bitcoin, è possibile tentare di collegarle ad identità reali, qualora uno degli indirizzi utilizzati sia pubblicamente associato ad un'organizzazione o individuo. Per fare un esempio, Wikileaks permette di fare donazioni di bitcoins al proprio indirizzo, pubblicato su web; qualora si potesse associare tale indirizzo ad altri, utilizzati come input per diverse transazioni, sarebbe possibile determinare quali transazioni siano state effettuate da Wikileaks.
Un nodo malevolo, dopo aver costruito tale mapping, potrebbe tracciare i pagamenti delle singole entità.
Una potenziale debolezza del sistema di privacy è data anche dalla rete stessa: un attaccante potrebbe stabilire il maggior numero possibile di connessioni con i nodi della rete Bitcoin e tentare di associare l'indirizzo IP con cui una transazione è stata inviata per la prima volta, considerando che le seguenti ricezioni di questa da altri nodi sarebbero probabilmente dovute al meccanismo di propagazione, al mittente di quella transazione.
In merito a quest'ultimo aspetto, molti clients implementano la possibilità di utilizzare dei servers *Tor* (The Onion Router) come relay per spedire transazioni e blocchi al resto della rete Bitcoin, al fine di vanificare l'analisi del traffico da parte dei riceventi.

3.3 Struttura

Dopo aver discusso il contesto di utilizzo di Bitcoin e i requisiti non funzionali di maggiore rilevanza, in questa sezione sarà fornita una descrizione della sua struttura.
Considerando l'omogeneità degli attori nell'universo Bitcoin (tutti i peers sono uguali e non ne esistono di "più uguali degli altri", per parafrasare Orwell), la descrizione dell'architettura interna di un singolo nodo è sufficiente per descrivere quella dell'intero Bitcoin.
Come riferimento sarà utilizzato il client originale Bitcoin Core (o *Bitcoind*, escludendone i componenti per l'interazione utente), in quanto capace di svolgere tutte le azioni descritte nella parte introduttiva; nonostante ciò, esiste una vasta gamma di prodotti software (open source e non), scritti in diversi linguaggi di programmazione e adatti ad esecuzione su piattaforme di diversa natura, che incorporano diversi sottoinsiemi delle funzioni del suddetto client.

I nomi utilizzati per i componenti principali rappresentati in figura 6 esprimono in maniera piuttosto intuitiva le funzioni svolte da questi, in accordo con la terminologia utilizzata precedentemente: il *Core* coordina le operazioni degli altri componenti, il *Wallet* gestisce le transazioni emesse dall'utente o a lui destinate, la *Chain* memorizza l'intera block chain e strutture dati annesse, il *Miner* svolge le funzioni di mining, il *Node* gestisce la comunicazione con gli altri nodi della rete Bitcoin e la *GUI* consiste nell'interfaccia grafica esposta all'utente.
Nella sezione 3.4 saranno descritte con maggiori dettagli le funzioni svolte da ciascun componente.
Il principale connettore è rappresentato dal protocollo Bitcoin stesso, atto a realizzare comunicazione fra diversi clients; altri sono HTTP (per le varianti client-server) e JSON-RPC.
Segue una overview delle tipologie di client Bitcoin e relative differenze strutturali [Sku12].

3.3.1 Full

I cosiddetti "full" clients sono quelli che implementano l'intero protocollo Bitcoin e possiedono una copia completa della block chain.
Questi clients sono in grado di scoprire nuovi nodi e comunicare con essi, inviare e ricevere transazioni e blocchi, memorizzare i blocchi validi su storage locale, verificare tutte le transazioni ricevute ed emettere in broadcast ad altri nodi quelle valide.
Molti clients di questo tipo offrono servizi addizionali all'utente, cioè non strettamente connessi alla partecipazione alla rete Bitcoin: memorizzare le proprie transazioni nel wallet e criptarlo, fornire un'interfaccia grafica o a linea di comando per facilitare l'interazione utente e altro.
Fino alla versione 0.3.22, il client originale possedeva anche le funzionalità necessarie a fare mining di bitcoins, rimosse in seguito per dare spazio a clients specializzati (vedi 3.3.5).
I clients di questo tipo sono quelli che contribuiscono al mantenimento della coerenza globale della rete Bitcoin, propagando blocchi e transazioni dopo un accurato controllo; questo aspetto li fa gravare sul sistema che li esegue, in quanto consumano una notevole quantità di bandwidth, tempo di calcolo e spazio su disco.
Il diagramma dei componenti in figura 6 mostra la struttura di un client completo.
Alcuni clients di questo tipo sono Bitcoin Core, Armory e Libbitcoin.

3.3.2 Headers-only

Questi clients memorizzano l'intera block chain, ma comprensiva dei soli headers di ciascun blocco (quindi escludendo le transazioni contenute in esso).
Questo ha l'indubbio vantaggio di ridurre drasticamente il consumo di memoria (un totale 22 MB circa, attualmente), ma impedisce ai nodi di confrontare le transazioni ricevute con quelle contenute all'interno dei blocchi della chain, per verificare che siano valide (quindi che non siano tentativi di double-spending o presentino un importo mal calcolato).
Nonostante ciò, in [Nak08] viene suggerita la possibilità di utilizzare i soli headers dei blocchi per verificare la validità di una transazione (*Simplified Payment Verification*), determinando in quale blocco della chain sia contenuta la transazione e osservando un numero sufficiente di blocchi concatenati ad esso; maggiori dettagli su come ottenere un "link" alla transazione all'interno di un blocco in 3.5.4.
Un client di questo tipo è Multibit.

3.3.3 Signing-only

I clients di questo tipo non memorizzano alcuna informazione relativa alla block chain e ad altri peers: si occupano unicamente di creare transazioni ed inviarle ad un nodo designato, il quale le propagherà verso la rete Bitcoin.
Alternativamente, possono ricevere il push di transazioni di interesse (per esempio destinate a loro, o altri indirizzi Bitcoin).
La differenza sostanziale fra questa tipologia di client e le precedenti risiede nel fatto che sia basata su un modello Client-Server: il nodo designato per inviare e ricevere transazioni funge da unico intermediario verso il resto della rete Bitcoin.
Nella maggior parte dei casi questi providers forniscono accesso ai loro servizi tramite in-

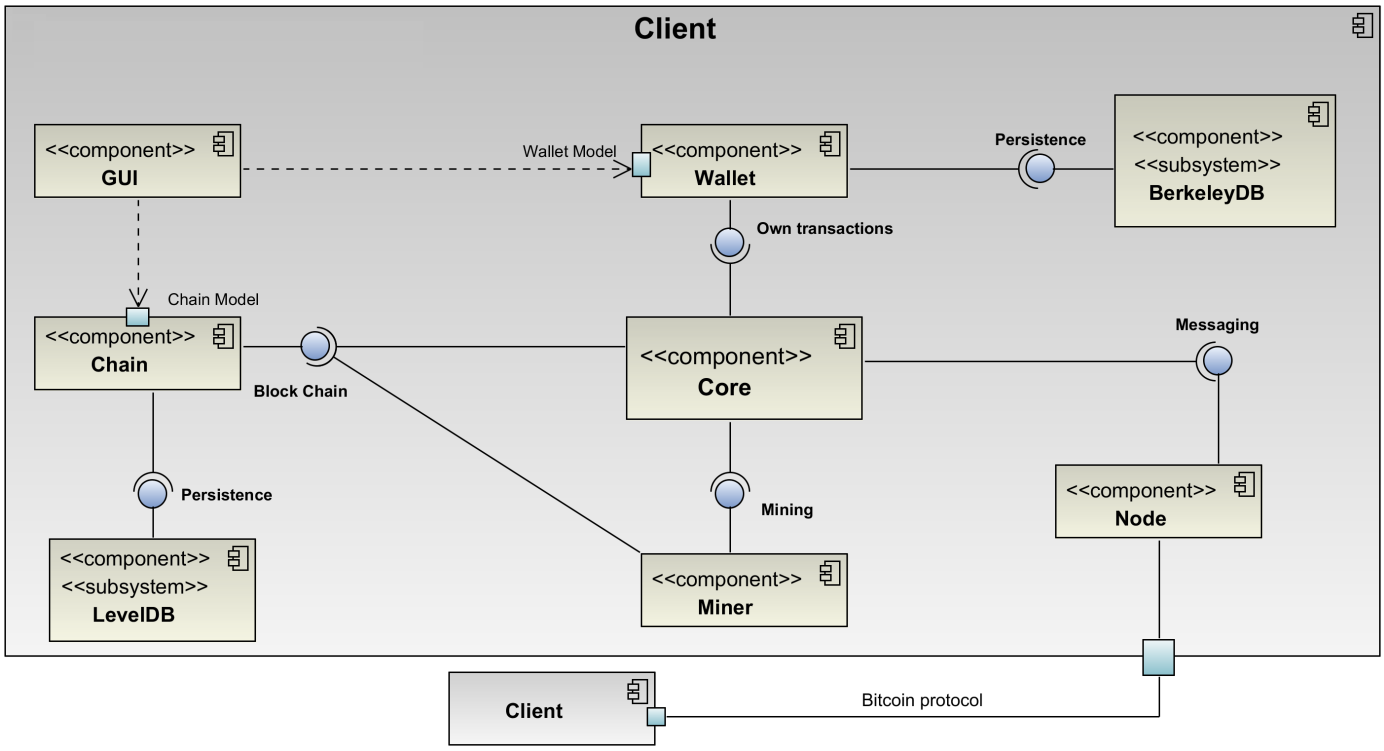

Figura 6. Diagramma dei componenti di un client Bitcoin

terfacce RESTful e le richieste sono inviate utilizzando il protocollo HTTP.

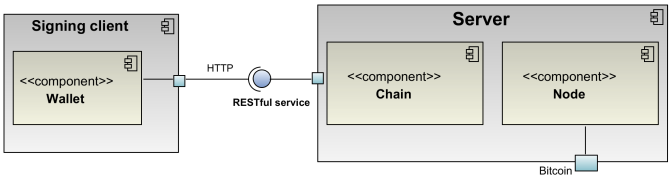

Figura 7. Signing-only client

Il pregio principale di questi clients è il consumo di bandwidth e memoria notevolmente ridotto, permettendone l'esecuzione su dispositivi desktop (Electrum) e mobili (BitcoinSpinner).
Esiste anche una web application (BlockChain.info) che fornisce un client di questo tipo, permettendo l'invio di transazioni da browser; in questo caso le chiavi private sono criptate con librerie Javascript dal dispositivo client e memorizzate successivamente dal server.
Lo svantaggio consiste nella necessità di far transitare le transazioni dirette all'utente attraverso un server, il quale potrebbe falsificarne gli importi. Questo attacco è tuttavia inutile in quanto il server non potrà invece contraffare le transazioni uscenti, le quali necessitano di firma digitale dal dispositivo client.
Nel diagramma dei componenti in figura 7 è mostrata la ripartizione dei componenti di un full client tra le due entità client e server (per alleggerire la rappresentazione sono mostrati solo i componenti concettualmente più rilevanti, al fine di evidenziare la distinzione).

3.3.4 Thin

I thin clients, anche denominati eWallets, portano ad un livello successivo i signing-only clients: si privano non solo della block chain e degli indirizzi di altri peers, ma anche del wallet (quindi chiavi private), affidandosi completamente ad un server esterno.
Questi nodi si limitano ad inviare richieste al server, il quale assume una funzione di banca: gestisce chiavi private, firma ed emette transazioni per conto loro, mostra il bilancio attuale e li notifica di eventuali accrediti; le

modalità di comunicazione sono generalmente le medesime dei signing-only clients (HTTP).

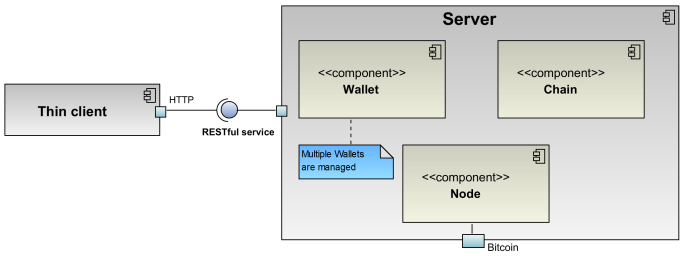

Figura 8. Thin client

Alcuni esempi di thin clients sono MyBitcoin, Coinbase, Instawallet e MtGox Wallet.
La semplicità è l'indubbio vantaggio di questi clients, i quali eliminano tutte le problematiche di sicurezza relative alla gestione delle chiavi private.
Il problema di trust dei signing-only clients è tuttavia notevolmente aggravato: il server ha pieno controllo dei bitcoins posseduti dal client, da cui deriva che una perdita o furto di bitcoins graverebbe anche sui clients interessati.
Un esempio di fiducia mal risposta in uno di questi provider è MyBitcoin, il quale divenne inaccessibile il 29 Luglio del 2011 e, tornando operativo, annunciò il furto da parte di ignoti di metà dei bitcoins posseduti dagli utenti.
Più recente è l'episodio di Mt.Gox, provider di trading giapponese andato in bancarotta a inizio del Marzo 2014, dichiarando la scomparsa di più di 850,000 bitcoins e ricevendo numerose cause legali dai propri clienti.
Questi clients sono quelli che maggiormente stravolgono la natura decentralizzata di bitcoin, annullando lo scopo primario del sistema di evitare il requisito di trust in un'entità gestita da terze parti.

3.3.5 Mining

I mining clients non inviano o ricevono transazioni, ma utilizzano tutte le loro risorse computazionali per costruire blocchi validi da aggiungere alla block chain: si limitano ad iterare applicazioni di SHA-256 per trovare il nonce corretto e soddisfare la proof-of-work del blocco attuale.
Inizialmente questa feature era implementata solo nell'originale full client Bitcoin Core, che la espletava utilizzando la CPU.
In seguito si è passati all'utilizzo di un hardware dedicato e con architettura a pipeline, quindi nettamente più veloce: la GPU.
Per questa ragione sono stati implementati alcuni mining clients specializzati (tra i quali Phoenix e CGMiner), che hanno portato all'eliminazione della feature di mining da Bitcoin Core.
Per dedicare la maggior quantità possibile di risorse al mining, questi clients delegano la responsabilità di reperire blocchi e transazioni dalla rete a dei full clients separati, con i quali comunicano (nel caso di Bitcoind) utilizzando il protocollo di chiamata di procedura remota *JSON-RPC*.
L'estremo di questa separazione fra mining-client e full-client consiste in diverse tipologie di hardware assemblati allo scopo:

- I *mining rigs*, che sfruttano array di GPU ed eliminano tutta la computazione e I/O non necessari.
- *Application specific integrated circuit* (ASIC): circuiti integrati specific purpose, dedicati unicamente alla computazione di SHA-256. Esistono numerosi prodotti di questo tipo in commercio, alcuni dei quali (es.: CoinTerra) raggiungono una frequenza di 2 TH/s (2,000,000,000,000 computazioni di hash al secondo).

Poiché, anche utilizzando una GPU, il mining risulta essere troppo lento per un singolo client (anche in base al valore attuale del target), è nata l'idea del *pooled mining*: combinare lo sforzo computazionale di più miners per risolvere l'hashing di un unico blocco (con diverse politiche di divisione del lavoro, ossia dei nonce da provare).
Questa divisione del lavoro produce ricompense più piccole (ripartite fra i miners che partecipano alla pool), ma più frequenti, risultando in una notevole popolarità del sistema.

3.4 Funzioni

Saranno descritte adesso le funzioni svolte da ognuno degli elementi principali che

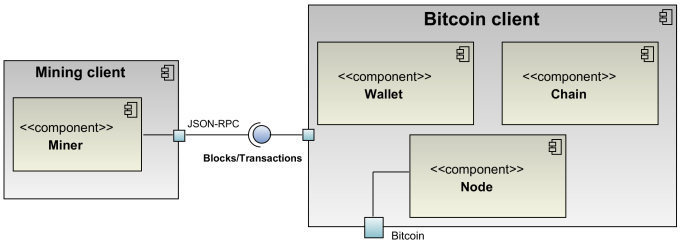

Figura 9. Mining client

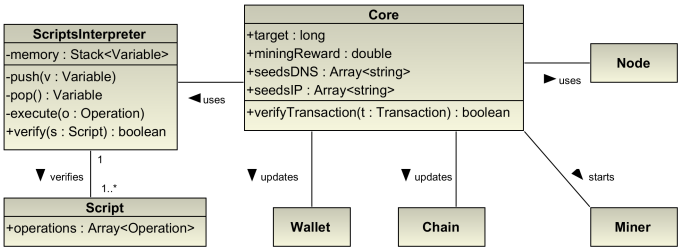

Figura 10. Diagramma di classi del Core

compongono l'architettura di un client Bitcoin. La spiegazione di alcuni dettagli inclusi nei diagrammi UML che accompagnano la descrizione testuale è rimandata alla sezione 3.5.

3.4.1 Core

Il Core rappresenta il centro della logica interna del client: inizializza gli altri componenti e scambia informazioni con loro, offrendo una raccolta di variabili e procedure che include i parametri attuali della rete Bitcoin e varie funzioni di base.
Le variabili in questione sono il target corrente per la proof-of-work, il quantitativo di bitcoins prodotti dalla creazione di un blocco, una raccolta di indirizzi IP e nomi di dominio di nodi ritenuti "stabili" (aggiornati a ogni nuova sub-release del client, vedi 3.5.2) e altro.
Le funzioni di base offerte dal Core consistono principalmente nella verifica sintattica di una transazione o blocco, esecuzione di scripts per controllare che una transazione possa effettivamente spendere l'output di un'altra (vedi 3.5.3), verifica di una firma digitale o digest e altro.
Il Core funge anche da "centro di smistamento" per transazioni e blocchi in transito da e verso l'esterno, dopo opportuni controlli: inoltra al Wallet le transazioni dirette all'utente e alla Chain transazioni non confermate e blocchi ricevuti, o al Node quelli uscenti.

3.4.2 Wallet

Componente fondamentale è il Wallet: contiene coppie di chiavi pubblica-privata per ognuno degli indirizzi Bitcoin dell'utente, una lista ordinata cronologicamente delle transazioni effettuate/ricevute, una pool di generazione di nuove chiavi, una rubrica degli indirizzi Bitcoin con i quali si sono scambiate transazioni e un set di preferenze utente.
Il Wallet è inoltre responsabile della creazione di nuove transazioni e controllo del bilancio di un utente, eventualmente reperendo le informazioni necessarie dalla block chain.
Il formato con cui è memorizzato su disco può variare a seconda del client utilizzato: Bitcoin Core, ad esempio, memorizza il Wallet su file system utilizzando il database management system **Berkeley DB** (non relazionale, basato su array di coppie chiave/valore).
Altri clients utilizzano formati binari ad hoc (Armory) o semplici strutture JSON (Blockchain.info).
Essendo il Wallet ciò che permette ad un utente di spendere i bitcoins precedentemente ricevuti, la sua sicurezza è di primaria importanza.
Per questa ragione esistono molti strumenti, spesso incorporati all'interno dei clients stessi, per cifrare il Wallet con crittografia simmetrica (la cui chiave è costituita da una passphrase).
Molti di questi strumenti offrono persino la possibilità di stampare le chiavi contenute nel Wallet su carta, sotto forma di stringhe o QR code; in questo modo è possibile stampare il proprio Wallet e conservarlo in un luogo sicuro, come la cassetta di sicurezza di una banca.
In genere i Wallet producono coppie di chiavi con valori casuali per ogni transazione effettuata dall'utente, generando fino a ≈ 100 coppie diverse, rimpiazzate di volta in volta dalle nuove secondo una politica FIFO (First In First Out); al fine di poter rimuovere una coppia di chiavi in modo permanente, si fa in modo che i bitcoins associati al suo indirizzo siano

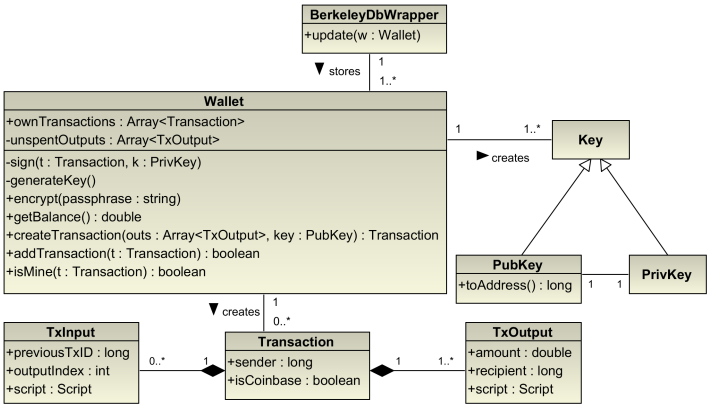

Figura 11. Diagramma di classi del Wallet

trasferiti ad uno dei nuovi.
Poiché gli indirizzi Bitcoin sono generati offline, non è impossibile che due utenti creino il medesimo indipendentemente, sebbene altamente improbabile: lo spazio di valori delle chiavi pubbliche generate è pari a 2^{512}, poiché queste hanno dimensione pari a 512 bits. Le chiavi private sono invece a 256 bits.
Poiché mantenere centinaia di chiavi generate casualmente complica notevolmente la gestione e sicurezza del Wallet, esistono alcune varianti particolari:

- *Deterministic Wallet*: utilizza un generatore di numeri pseudo-casuali per creare coppie di chiavi, a partire da un *seed* relativamente piccolo. Impiegando questo approccio puramente algoritmico, è possibile ricostruire l'intera sequenza di coppie di chiavi (e quindi di transazioni effettuate/ricevute) a partire dal singolo seed, rendendo necessario solo il backup di quest'ultimo.
- *Brain Wallet*: è, in un certo senso, una specializzazione dei Deterministic. Consente di convertire un seed da 128 bits (dimensione sufficiente ad assicurarne l'unicità) in un codice mnemonico, composto in genere da 12 parole in una qualunque lingua, facili da memorizzare.
  In questo modo è possibile cancellare il file relativo al Wallet e rigenerarlo al bisogno, utilizzando il solo codice conservato nella memoria dell'utente (per cui "Brain" Wallet).
- *Watch-only Wallet*: consiste nella completa separazione tra chiavi pubbliche (quindi identificativi degli utenti e soli mezzi necessari per tenere traccia delle transazioni associate ad essi) e private.
  In questo modo, non avendo la possibilità di effettuare pagamenti, è possibile mantenere il Watch-only Wallet su postazioni che non garantiscono totale sicurezza (in effetti, l'unica informazione utile che un attaccante potrebbe reperire è il collegamento fra diverse chiavi pubbliche ed una singola identità).

3.4.3 Node

Il Node (secondo la terminologia utilizzata dal client originale) è il componente che gestisce l'interazione del singolo nodo con la rete Bitcoin: implementa l'utilizzo del protocollo Bitcoin, comprensivo di tutti i suoi messaggi.
Le responsabilità principali di un Node sono quelle di effettuare il discovery di altri nodi nella rete (seguendo uno specifico processo di bootstrapping descritto in 3.5.2), verificarne periodicamente la connettività e, soprattutto, scambiare con essi i messaggi previsti dal protocollo.
I messaggi comprendono la richiesta di indirizzi di altri nodi, l'invio di transazioni o blocchi, la richiesta di blocchi specifici, il semplice ping, etc. Un elenco esaustivo dei messaggi o descrizione dettagliata del loro formato va al di là dello scopo di questo documento.
Il Node inoltra i messaggi ricevuti verso altri nodi della rete, seguendo determinate politiche (3.5.2).
Gli indirizzi IP di altri nodi sono memorizzati su file in un formato binario ad-hoc, al fine di poterli riutilizzare al successivo avvio del client.

3.4.4 Miner

Il Miner è il componente dell'architettura che si occupa della creazione di nuovi blocchi, a partire dalle informazioni sui precedenti e dalle transazioni ricevute da altri nodi (o prodotti da quello che esegue il client) ma non ancora validate.
Se il client permette pooled mining, questo componente incorpora al suo interno la logi-

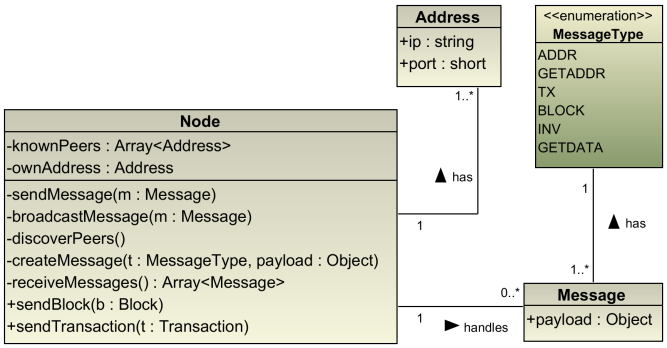

Figura 12. Diagramma di classi del Node

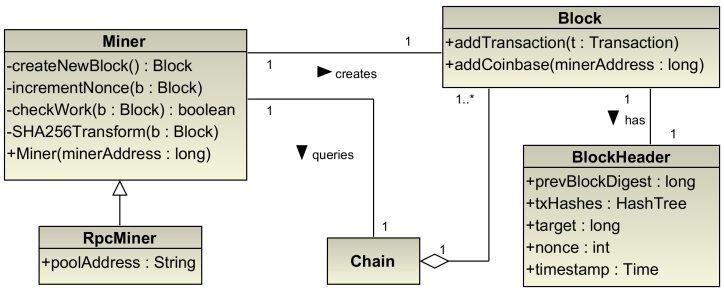

Figura 13. Diagramma di classi del Miner

ca di comunicazione e coordinamento necessaria, solitamente utilizzando il protocollo di chiamata di procedura remota JSON-RPC.
Oggigiorno è frequente che questo componente risieda in un'applicazione separata (vedi 3.3.5), ma il client Bitcoin Core ne implementa tuttora le funzionalità.

3.4.5 Chain

La Chain memorizza lo storico di tutte le transazioni effettuate a partire dalla nascita di Bitcoin, raggruppate in blocchi: la block chain. I blocchi contenuti in essa, concatenati in una lista, sono richiesti dagli altri componenti per lettura, ad esempio per controllare la validità di un blocco appena ricevuto o che l'output di una transazione non sia già stato speso, oppure ve ne sono aggiunti di nuovi in scrittura.
In sostanza, la Chain incorpora le seguenti strutture dati:

- Tutti i blocchi accettati nella block chain, comprensivi di body (transazioni) ed header, conservati su memoria secondaria.
- Una hash map contenente tutti i blocchi "orfani" (il cui predecessore non è ancora presente nella chain) ricevuti, memorizzati con chiave pari al loro digest, ottenuto mediante SHA-256.
- Una hash map che associa una transazione già accettata al blocco della chain che la contiene, utilizzando come chiavi i valori di *txid*: identificativo equivalente al digest con SHA-256 della transazione.
- Una lista ordinata (per transaction fees) di transazioni ricevute e non ancora accettate all'interno della chain, incorporata all'interno di una classe denominata *TxMemPool*. Questa lista è utilizzata dal Miner per determinare le transazioni da inserire in un blocco: quelle con transaction fees più alte, così da massimizzare il guadagno.
- Files che memorizzano stati precedenti della block chain, utilizzati qualora sia necessario un rollback (in caso di scarto di una fork).

La block chain viene memorizzata su disco (19.8 Gigabytes, in data 2014-04-10) utilizzando un database management system **LevelDB**, simile a Berkeley DB (coppie chiave-valore), ma più leggero e veloce.
Considerando la frequenza con cui, potenzialmente, un nodo possa doverla leggere, parte della block chain è mantenuta in memoria, per un accesso più veloce. In particolar modo, la rappresentazione in memoria contiene le varie hash maps, al fine di consentire un rapido accesso ad una specifica transazione o blocco, per esempio in fase di verifica.

3.4.6 GUI

La GUI è l'interfaccia del client verso l'utente: consente di emettere nuove transazioni attraverso un semplice form, consultare un elenco di quelle ricevute e relativi indirizzi Bitcoin, visionare il proprio bilancio, configurare un set di opzioni e visualizzare un qualunque sottoinsieme della block chain (tramite opportuni filtri).
In realtà, il client originale Bitcoin Core (così come molti derivati) consente di interfacciarsi con gli altri componenti (Wallet, Chain) utilizzando un'interfaccia a riga di comando o grafica.

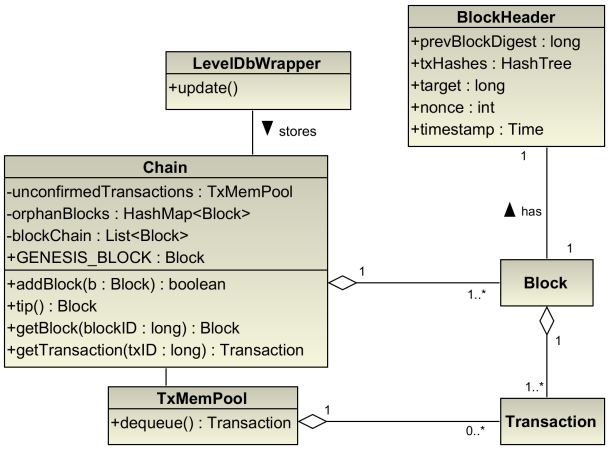

Figura 14. Diagramma di classi della Chain

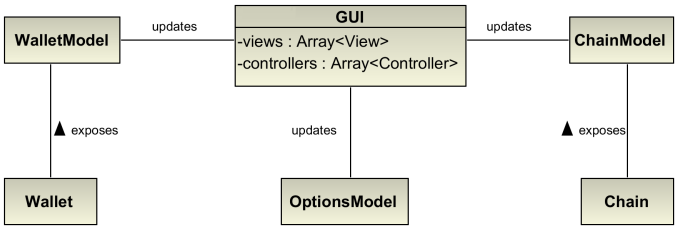

Figura 15. Diagramma di classi della GUI

Quest'ultima, nel caso di Bitcoin Core, prende il nome di *Bitcoin-Qt*, in quanto sfrutta le librerie cross-platform *Qt* per realizzare i componenti dell'interfaccia grafica; da ciò ne deriva la compatibilità con Linux, Windows e Mac.
Altre implementazioni utilizzano wxWidgets, GTK, Java Swing o i frameworks per Android o iOS.
Il connettore con cui questo componente comunica con la logica interna del sistema (rappresentata dal processo Bitcoind) è il protocollo di chiamata di procedura remota JSON-RPC.
All'interno di Bitcoin Core, la separazione fra interfaccia grafica e logica applicativa è realizzata utilizzando il pattern *Model-View-Controller* (MVC), definendo dei wrappers (models) attorno ai componenti Wallet e Chain.

3.5 Comportamento

Di seguito saranno esaminate le azioni fondamentali eseguite dal client Bitcoin Core (quindi, di riflesso, quelle eseguite da ogni nodo partecipante alla rete peer-to-peer Bitcoin), descrivendo la sequenza di operazioni prevista per ognuna di esse.

3.5.1 Boot

La prima operazione che il client esegue all'avvio è quella di popolare le strutture dati in memoria, relative a Core, Wallet, Chain e Node (indirizzi IP di altri peers precedentemente registrati).
Successivamente un socket di rete è posto in ascolto di connessioni sulla porta predefinita, restituendo errore e terminando nel caso questo non sia possibile.
A questo punto il processo genera una sorta di master thread, *StartNode*, responsabile della gestione delle comunicazioni con la rete Bitcoin (parte integrante del Node).
Il thread in questione è denominato "master" in quanto genera altri threads:

- *SocketHandler*: entra in un loop infinito nel quale accetta nuove connessioni sul socket precedentemente posto in ascolto, gestisce letture e scritture sui sockets creati e chiude quelli inattivi.
  Fa uso di tecniche di I/O Multiplexing (funzione *select*) per controllare più sockets, incluso il primario in attesa di nuove connessioni, senza bloccarsi.
- *DNSAddressSeed*: tenta di effettuare richieste DNS, al fine di risolvere i nomi di dominio incorporati nel client in indirizzi IP (vedi 3.5.2).
- *GetMyExternalIP*: tenta di determinare il proprio IP pubblico, mediante richieste HTTP a servizi di terze parti (vedi 3.5.2).
- *OpenConnections*: apre connessioni verso nuovi indirizzi.
- *MessageHandler*: controlla le code di messaggi verso e da ogni nodo con il quale si è stabilita una connessione, processando quelli che trova e inoltrandoli ad altri componenti del client (per esempio, trasferendo al Wallet eventuali transazioni dirette all'utente).
- *DumpAddresses*: scrive periodicamente gli indirizzi IP dei nodi considerati attivi su file (*peers.dat*).
- *BitcoinMiner*: svolge le operazioni di mining.

Gli accessi a sezioni critiche, quali letture e scritture delle code di messaggi per ogni nodo (effettuate da SocketHandler per inviare e ricevere dati attraverso la rete e MessageHandler, per processare i messaggi), sono gestite attraverso uso di semafori (libreria *boost*).
Una volta effettuato il discovery di un numero sufficiente di nodi, il client effettuerà un aggiornamento della propria block chain con quella mantenuta dai nodi della rete, inviando un messaggio di tipo *getblocks* contenente il digest dell'ultimo blocco noto. Se i riceventi di tale messaggio possiedono dei blocchi più recenti provvederanno a notificare il mittente, procedendo quindi all'invio dei blocchi veri e propri (vedi 3.5.4).
Il Wallet creerà una pool di transazioni ricevute dall'utente e non ancora spese, utilizzando il suo database interno o, eventualmente, quello della Chain.
Ogni full client Bitcoin è strutturato come un'applicazione multithreaded, dovendo intervallare una quantità notevole di computazione (controllo di blocchi e transazioni e/o mining) con la gestione di numerose connessioni di rete simultanee.

3.5.2 Discovery

Operazione fondamentale, in un'architettura peer-to-peer pura come quella di Bitcoin, è il discovery di nuovi nodi a inizializzazione del client.
In molte architetture peer-to-peer centralizzate o ibride si fa riferimento ad uno (o più) naming server, i quali forniscono una lista di peer a cui connettersi, con relativi indirizzi di rete.
Bitcoin, al fine di evitare l'introduzione di una componente centralizzata, utilizza altri meccanismi.
Al bootstrapping di un nodo, questo ottiene indirizzi di altri peers con le seguenti operazioni:

1) Tenta il caricamento degli eventuali indirizzi memorizzati in precedenza nel database interno.
2) Effettua richieste DNS ad alcuni nomi di dominio hardcoded all'interno del client (es: bitseed.xf2.org, dnsseed.bluematt.me, seed.bitcoin.sipa.be, dnsseed.bitcoin.dashjr.org, seed.bitcoinstats.com, . . .), memorizzando gli indirizzi nelle risposte.
3) Tenta la connessione ad alcuni indirizzi IP hardcoded all'interno del client (i cosiddetti seeds): questi sono aggiornati con ogni nuova release dell'applicazione.

I metodi 2 e 3 sono seguiti, in genere, solo al bisogno (primo avvio del nodo o incapacità di connettersi agli indirizzi memorizzati in precedenza); i nomi di dominio e i seeds harcoded sono aggiornati ad ogni sub-release del client e relativi a nodi come tutti gli altri, la cui presenza sulla rete è pressoché costante.
Dopo l'avvio del client, i mezzi principali che due nodi possono utilizzare per scambiarsi indirizzi di altri peers sono i messaggi *getaddr* e *addr*.
Se un nodo P riceve un messaggio getaddr da Q, controlla la sua pool di indirizzi e ne seleziona al più 2,500 il cui timestamp associato (relativo all'ultimo messaggio ricevuto dall'indirizzo in questione) non sia più vecchio di 3 ore, inserendoli in un messaggio addr di risposta.
Il client tenta anche di reperire il proprio indirizzo pubblico, mediante richieste HTTP a checkip.dyndns.org o www.showmyip.com, così da inviarlo periodicamente ai nodi nella sua pool, all'interno di messaggi addr spontanei (o "gratuitous", che non seguono un getaddr).
Il client memorizza periodicamente su file gli indirizzi dei peers attivi con cui ha stabilito connessioni; il criterio con cui si determina l'"attività" di un nodo è lo stesso utilizzato nel selezionare quali indirizzi accludere ad un messaggio addr.
Vale la pena menzionare che gli indirizzi dei seeds incorporati nel client vengono memorizzati (all'avvio) con un timestamp pari a 0, così da escluderli sempre dai messaggi addr.
Il numero di connessioni TCP simultaneamente attive verso altri peers è un parametro che varia con il client utilizzato: Bitcoin Core, ad esempio, cerca di mantenersi sulle 12-20 connessioni.

3.5.3 Transazioni

Una transazione consiste in un messaggio firmato ed emesso in broadcast sulla rete Bitcoin, eventualmente inserito all'interno di un blocco.
Tipicamente, una transazione fa riferimento a transazioni effettuate precedentemente da altri indirizzi Bitcoin verso quello di chi la emette e "trasferisce" un quantitativo di bitcoins ad uno o più indirizzi riceventi.
Il componente che incorpora la logica della creazione di nuove transazioni è il Wallet.
La GUI consente di emettere una transazione specificandone i parametri in un form: indirizzo Bitcoin del destinatario (eventualmente reperito dalla rubrica interna) e importo (un minimo di 546 satoshi).
Nel momento in cui l'utente conferma la transazione, è avviata una sequenza di operazioni che comprende la sua costruzione, verifica ed invio alla rete.
Come prima cosa vengono selezionate una o più transazioni da usare come input, direttamente dalla pool interna del Wallet, prese in ordine cronologico.
La pool conterrà tutte le transazioni con output diretti all'utente e non ancora spesi, essendo stata riempita a inizializzazione.
Una transazione prevede uno o più campi *input*, ognuno composto da:

- Il digest di una precedente transazione.
- L'indice dello specifico output a cui si vuol fare riferimento, all'interno della suddetta transazione.
- La prima metà di uno *script*.

Seguono uno o più campi *output*, composti dall'importo da trasferire in satoshi e dalla seconda metà di uno script.
Lo script consiste fondamentalmente in un set di istruzioni, definite con un domain-specific language molto simile a *Forth*, denominato *Script*, che specifica come il beneficiario (o i beneficiari) di una transazione possa utilizzarla in futuro, quindi spendere i bitcoins trasferiti con essa.
L'interprete del linguaggio utilizzato, come in Forth, utilizza due strutture a stack separate per memorizzare record di chiamate di procedura e variabili. Lo script è processato da sinistra verso destra, aggiungendo le variabili sullo stack e utilizzando comandi ad n parametri per processare le n variabili in cima ad esso, eventualmente producendone di nuove.
Esistono comandi aritmetici, logici, di controllo di flusso, orientati alla crittografia e di manipolazione dello stack; non è possibile effettuare dei cicli (volutamente al fine di evitare programmi non terminanti), da cui consegue la non Turing-completezza del linguaggio.
L'interprete del linguaggio di scripting è implementato all'interno del Core.
Lo script implementa la verifica che la transazione specificata come input sia "spendibile"; per fare ciò, viene innanzitutto eseguita la metà dello script contenuta nell'input, seguita quindi da quella contenuta nell'output a cui si fa riferimento.
In sostanza, i risultati prodotti dalla metà nell'input definiscono i parametri dei comandi contenuti nella metà dell'output.
Se lo script termina senza interruzioni (trigger di errori) e il risultante valore in cima allo stack corrisponde alla costante OP_TRUE, allora la transazione è valida.
Questa combinazione di due script viene denominata *Contract*: permette a due o più parti interessate di specificare i termini della transazione, ossia cosa è richiesto affinché il ricevente possa beneficiarne.
La tipologia di Contract più utilizzata (spesso l'unica prevista dai clients) è la cosiddetta *Pay-to-PubkeyHash*: la prima metà dello script (scriptSig) consiste in due parametri, ossia la firma della transazione precedente e la chiave pubblica del beneficiario della transazione; la seconda metà (scriptPubKey) fa hashing della chiave pubblica e controlla che sia uguale al digest della stessa precedentemente hardcoded all'interno dello script, quindi effettua la verifica della firma, producendo eventualmente OP_TRUE.
La firma viene eseguita applicando l'algoritmo *Elliptic Curve Digital Signature Algorithm* (ECDSA) con chiave privata del beneficiario della transazione; come funzione di hashing viene di norma utilizzata SHA-256.
Discutere delle potenzialità espressive del linguaggio di scripting utilizzato e di Contracts più complessi va al di là dello scopo di questo

documento; basti pensare che lo script di output possa dover verificare una password, una combinazione di più firme o non richiederne affatto.
Per semplificare il collegamento fra outputs e successivi inputs, non è possibile che un singolo output sia indicato da molteplici inputs. Di conseguenza, se il valore x dell'output prodotto in una nuova transazione è minore della somma $y = \sum_{i=1}^{n} input_i$ dei valori di tutti gli n input indicati, viene automaticamente creato un secondo campo output, con importo $y - x$, diretto al mittente stesso della transazione (il *change*, resto).
In mancanza del change, o per scelta del mittente, $y - x$ o una frazione di tale valore sono destinati alla fee per i miners che genereranno il blocco relativo alla transazione.
Le fees possono rappresentare un mezzo per velocizzare la conferma della propria transazione: più sono alte, più è probabile che i miners includano la transazione all'interno del blocco che stanno costruendo.
Dopo una verifica locale della validità della transazione prodotta, questa viene spedita ad altri peers sulla rete, così che possa essere inserita in un blocco della block chain e confermata in via definitiva; stessa cosa accade per quelle valide ricevute da altri nodi.
L'invio è ripetuto periodicamente, evitando di ritrasmettere una transazione solo quando questa compare all'interno della block chain e può essere considerata "statisticamente" valida: il blocco in cui si trova la transazione ha un numero sufficiente di successori, stimato sui 5; risulta praticamente impossibile, per un'entità che non controlla almeno il 51% delle risorse computazionali disponibili nella rete Bitcoin, falsificare un tale numero di blocchi trovando la proof-of-work e vincere quindi il consenso collettivo.
Alternativamente, una transazione non viene ritrasmessa se il timestamp relativo ad essa indica che sia troppo vecchia (la soglia varia da poche ore a poche decine di minuti, a seconda del client): sarà il nodo interessato a doverla ritrasmettere.
Al fine di ridurre il consumo di bandwidth, l'invio di una transazione è preceduto da un messaggio *inv* (inventory) che ne annuncia la presenza.

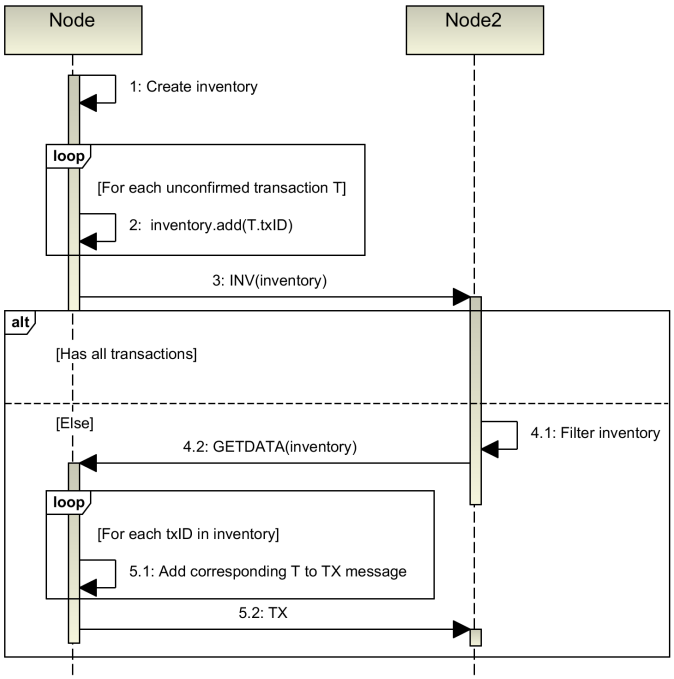

Figura 17. Sequenza di messaggi per invio di più transazioni

Il messaggio inv è composto da una o più coppie di elementi, contenente il tipo dell'elemento (transazione o blocco) e suo digest.
Il nodo che riceve un messaggio inv controlla se possiede gli elementi indicati al suo interno nella propria memoria locale (utilizzando il loro digest): tutte le transazioni o i blocchi non presenti sono richiesti al mittente del messaggio inv attraverso un messaggio *getdata*, utilizzando il medesimo formato.
A getdata segue un messaggio *tx*, contenente le transazioni vere e proprie.
Il vantaggio di questo scambio di messaggi è quello di evitare uno spreco eccessivo di bandwidth e velocizzarne la propagazione: le dimensioni tipiche di un inv sono 61 B, contro i 10 KB medi di un tx. Per uno studio approfondito di questi meccanismi di propagazione dei messaggi e relativi risultati analitici si rimanda a [DW13].
Le transazioni che compaiono infine nella block chain sono inserite nella hash map che le associa al corrispettivo blocco (3.4.5), così da poter essere controllate velocemente, in tempo $O(1)$, all'atto di dover verificare la validità di una

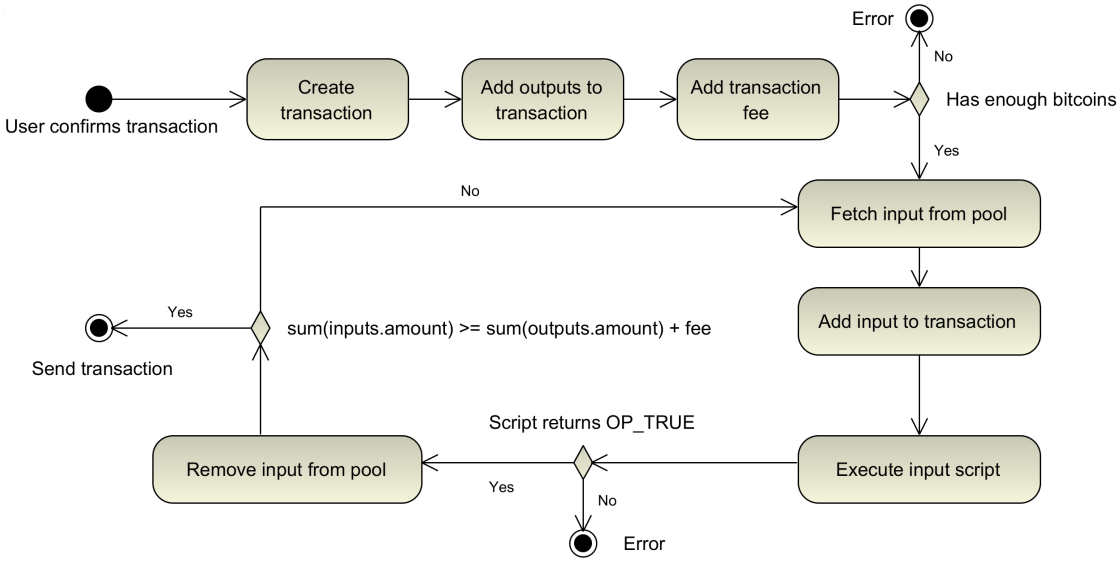

Figura 16. Emissione di una transazione

nuova transazione che le indica come input.

3.5.4 Mining

Il mining è il processo che consente di validare le transazioni emesse, costituendo il pilastro alla base dell'affidabilità di Bitcoin ed il meccanismo di creazione di nuove monete.
Nei full clients come Bitcoin Core è un componente specializzato, il Miner, che svolge questo procedimento su uno o più threads separati.
Nella sezione 2 sono stati introdotti i principi alla base di questa attività, segue adesso una descrizione più accurata di come un blocco sia prodotto, inserito nella block chain e diffuso nella rete.
Un blocco include una o più transazioni all'interno del suo *body* e un *header* composto da diversi campi:

- Un numero di versione a 4 bits, costante per tutti i blocchi.
- Digest a 256 bits dell'header del blocco precedente.
- *Merkle Root* a 256 bits, ricavato da tutte le transazioni nel blocco.
- Un timestamp UTC.
- Il target a 256 bits attuale.
- Il nonce a 32 bits attualmente considerato dal miner, nel processo di trovare la proof-of work necessaria alla validazione del blocco.

Il Merkle Root è ricavato mediante costruzione di un Hash tree (o Merkle tree).
Il tree è costruito applicando la funzione di hashing a tutte le transazioni nel blocco (foglie dell'albero) e ripetendo ricorsivamente il procedimento sui nodi intermedi, ottenuti come digest della concatenazione dei due figli.
Il processo termina restituendo il valore della radice (Root), che date n transazioni viene calcolata con un numero di applicazioni della funzione di hashing pari alla somma parziale n-esima $s_n = \sum_{k=0}^{\log_2 n} 2^k$ di una serie geometrica.
Nel caso di un numero dispari di transazioni è sufficiente che ad una di esse sia applicata due volte la funzione di hashing.
Calcolare un digest derivato da tutte le transazioni nel blocco consente di salvare spazio su disco, mantenendo un indice dell'integrità del blocco al solo costo (in termini di spazio, memoria) di 80 bytes, ossia la dimensione totale di un header.
Al fine di costruire un blocco ritenuto valido dallo schema proof-of-work impiegato da Bitcoin, il miner calcola ripetutamente il digest dell'header del blocco, applicando SHA-256; se la rappresentazione esadecimale del digest ottenuto è minore del target (ossia se ha un prefisso composto da un certo numero di bits 0) il blocco è pronto per essere inviato alla rete, altrimenti il nonce viene incrementato e il processo reiterato.

Il risultato dell'hashing, grazie alle proprietà di funzioni one-way, è del tutto imprevedibile: un qualunque numero compreso tra 0 e 2^{256}; più piccolo è il target, maggiore è la difficoltà (difficulty) nel trovare una delle soluzioni accettabili.
La difficulty è calcolata come il rapporto fra il massimo valore che il target può assumere e quello indicato dall'header del blocco.
Il target di riferimento per il mining viene modificato indipendentemente da ogni nodo della rete Bitcoin ogni 2016 blocchi ricevuti. A ricezione dell'ultimo blocco, un client confronta il tempo totale impiegato per produrre gli ultimi 2016 (calcolato come differenza fra il timestamp dell'ultimo e quello del primo) con quello ideale di 2 settimane, ottenuto considerando una media di un blocco ogni 10 minuti: il target viene modificato in base alla differenza percentuale fra di essi.
Da notare come aggiungere altre transazioni al blocco durante il procedimento, ricalcolando quindi la Merkle Root e modificando l'header del blocco, invaliderebbe il calcolo dei digest precedenti (uno dei nonce già usati potrebbe adesso portare al risultato).
Per questa ragione, una volta iniziata la ricerca della proof-of-work il miner non inserisce ulteriori transazioni all'interno del blocco.
Il compromesso fra il dedicare più tempo a cercare la proof-of-work o raccogliere il maggior numero di transazioni all'interno del blocco prima di iniziare, così da massimizzare la somma delle transaction fees, resta una scelta a discrezione del miner; esiste la possibilità di creare blocchi ad una sola transazione.
Generalmente il Miner estrae il maggior numero possibile di transizioni conservate nella lista ordinata per transaction fees mantenuta dalla Chain, così da massimizzare il guadagno.
Ci si potrebbe domandare come possa un elaboratore poco potente competere con un mining rig (3.3.5), ipotizzando che abbiano raccolto le stesse transazioni nel blocco e considerando il fatto che il nonce viene incrementato allo stesso modo da entrambi: otterrebbero i medesimi digests.
In realtà non è così, in quanto ogni miner aggiunge al blocco una transazione speciale (detta *coinbase*), il cui output è destinato al proprio indirizzo Bitcoin e che rappresenta il premio per la scoperta della proof-of-work; poiché questa transazione altera il Merkle Root e quindi l'header del blocco, nessun miner fa hashing della medesima sequenza di valori.
L'importo della coinbase è costante (25 BTC, al tempo della stesura di questo documento) ed è dimezzato automaticamente dai nodi della rete ogni 210,000 blocchi, al fine di evitare un'eccessiva produzione di bitcoins (3.1) e conseguente inflazione.
Nel caso in cui la somma totale degli inputs di una transazione nel blocco superi il valore emesso dagli outputs (determinando una transaction fee), la differenza viene aggiunta all'importo della coinbase dal miner.
Parametri quali il target attuale e la parte costante dell'importo della coinbase, nonché la logica di controllo delle transazioni ricevute, sono incorporate nel componente Core del client.
Rimane da spiegare come viene propagato un blocco, una volta che la proof-of-work è completata.
Il meccanismo di invio di un blocco da P a Q è analogo a quello utilizzato per le transazioni. P invia un messaggio di tipo *inv*, contenente il digest del blocco, a Q. Se Q non possiede il blocco localmente risponde con un *getdata*, al quale P replica inviando il blocco vero e proprio, in un messaggio *block*.
Anche in questo caso il vantaggio è dato dalla riduzione della bandwidth utilizzata (un messaggio di tipo block può avere una dimensione di svariati MBs).
Alla ricezione segue la verifica che un blocco sia valido: il digest del suo header sia minore del target dichiarato, tutte le transazioni all'interno del body siano valide e non duplicate (già contenute in un blocco nella block chain), via sia una sola coinbase e presenti il corretto importo.
Se il blocco è valido e non duplicato, viene aggiunto alla block chain. Sono previsti tre casi:

1) Il digest del blocco precedente corrisponde a quello attualmente in cima alla block chain. Rappresenta il caso ottimale: tutte le transazioni nel blocco sono memorizzate nella hash map che le associa ad esso

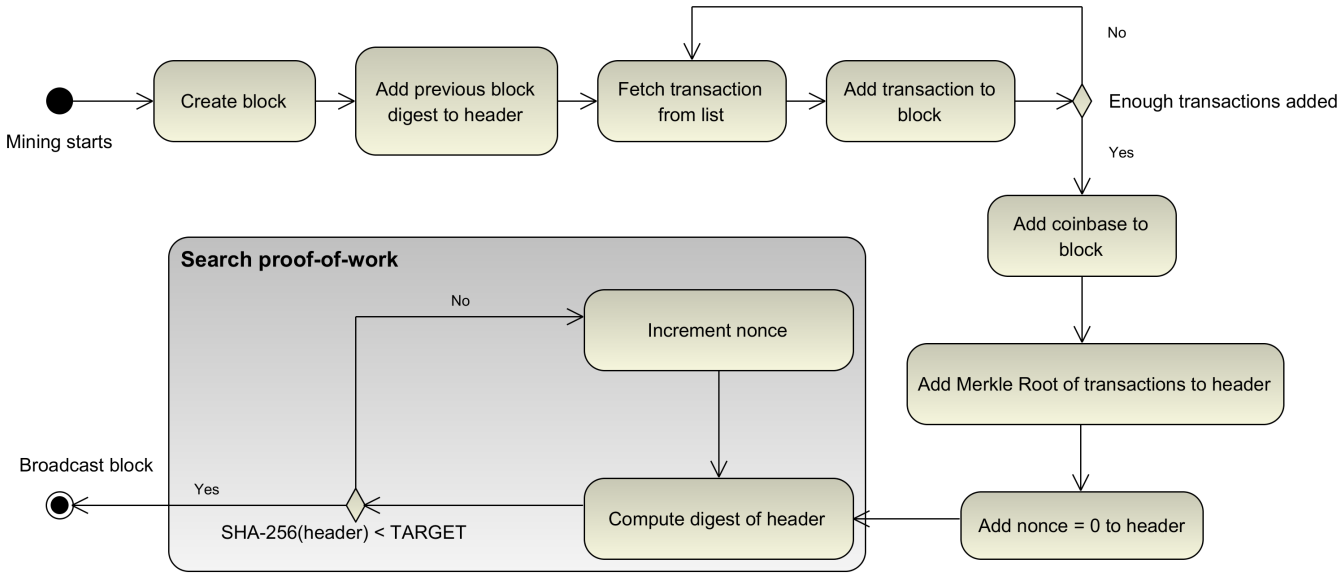

Figura 18. Mining di un blocco

(e nel Wallet se relative all'utente) e il blocco è inoltrato agli altri peers mediante lo stesso procedimento.

2) Il blocco è *orphan*: non ha un predecessore nella block chain. In questo caso è possibile che per ritardi nella propagazione dei messaggi o perdita di questi ultimi il blocco precedente non sia ancora arrivato. Il blocco orfano viene conservato in una pool di dimensione limitata, scartandolo qualora diventi troppo vecchio o la pool risultasse piena. Mediante il messaggio *getblocks* il client può richiedere il "genitore" di un blocco orfano agli altri peers.
3) Il digest del blocco precedente corrisponde a quello di un blocco intermedio nella block chain, causando una fork di quest'ultima.

Nel terzo caso, il client verifica che il "ramo" secondario abbia una difficulty complessiva dei blocchi al suo interno maggiore di quello primario, con una certa differenza minima: in caso affermativo il secondario diventa il nuovo primario e tutte le transazioni valide all'interno del vecchio primario e che non creano conflitti con quelle del secondario (double-spending) vengono aggiunte alla TxMemPool e ritrasmesse sulla rete, così da poter essere incluse in nuovi blocchi.

In caso di un pareggio, il secondario viene mantenuto nella memoria locale del client fino a che non prevalga una maggioranza.
Nel caso in cui il ramo secondario abbia una difficulty complessiva minore, tutte le transazioni al suo interno (se valide) sono aggiunte alla TxMemPool e inoltrate agli altri peers, mentre i blocchi vengono scartati.
La differenza minima nella difficulty di due biforcazioni che segna lo scarto di una di esse è un parametro dello specifico client, quindi non definito dal protocollo adottato da Bitcoin. Il client Bitcoin Core determina la cancellazione di una fork dalla memoria locale se questa è in difetto della difficulty relativa a 5 blocchi.
Non tutte le transazioni dei blocchi scartati sono re-inserite nella TxMemPool e reimmesse nella rete: fanno eccezione le coinbase, le quali sono perse per sempre (a danno dei miners che hanno prodotto i relativi blocchi).
Nell'eventualità in cui un nodo decida erroneamente che un ramo secondario vada scartato, a ricezione dei blocchi successivi di quel ramo (orfani) si ricadrà semplicemente nel caso 2.

3.6 Razionale

L'architettura di Bitcoin è basata su uno stile peer-to-peer puro, o decentralizzato: non esiste un'infrastruttura centralizzata per localizzazione dei peers o altri servizi, come non esistono

"super nodi"; tutt'al più vi sono dei nodi considerati affidabili, in termini di presenza sulla rete, i cui indirizzi sono hardcoded all'interno del client originale e che possono essere contattati al primo avvio di un nodo.
Come già discusso in 3.1 e 3.2, le ragioni dietro all'adozione di questo stile sono tanto ideologiche (nessuna istituzione finanziaria o governo a cui rispondere e nessuna esigenza di trust in organizzazioni gestite da terze parti) quanto pratiche.
In particolar modo è possibile ragionare su queste ultime in termini di compromessi (trade offs) raggiunti, valutando le possibili alternative: spiccano **affidabilità** e **indipendenza** (intesa, in questa sezione, come assenza di necessità di trust in terzi), a scapito dell'**efficienza**.
La possibilità di poter scaricare porzioni della block chain da qualunque nodo connesso alla rete Bitcoin, così come di poter ricevere conferma delle proprie transazioni connettendosi ad una minima frazione della rete, grazie al meccanismo di disseminazione utilizzato, rappresenta una forte garanzia di affidabilità.
La disseminazione delle informazioni tra i peers, salvo l'utilizzo di messaggi preliminari (inv e getdata) al fine di ridurre la mole di dati trasferiti, consiste in un puro e semplice flooding.
Il prezzo da pagare risiede nel consumo di memoria e bandwidth estremamente alto di cui soffrono i full clients; consumo che, col tempo e l'intensificarsi della frequenza di transazioni effettuate, andrà crescendo, minando la scalabilità del sistema.
Possibili soluzioni che non compromettano la decentralizzazione del sistema sono già state accennate in 3.2.
Per migliorare le performance delle operazioni sono stati introdotti nuovi tipi di clients: headers-only (3.3.2), signing-only (3.3.3) e thin (3.3.4).
Nell'ordine in cui sono stati elencati, questi derivati del client originale rappresentano una progressiva rinuncia degli utenti ai primi due requisiti (affidabilità e indipendenza) verso un goal di maggiore efficienza.
Gli headers-only rinunciano ad un controllo completo di blocchi e transazioni, ponendo una maggiore fiducia negli altri nodi (supposti full clients). I signing-only rinunciano alla possibilità di controllare lo stato delle proprie operazioni, conservando il solo accesso ai propri bitcoins mediante possesso del wallet. I thin clients rappresentano una migrazione completa verso un'architettura client-server centralizzata: non possiedono wallet o block chain, ma si limitano ad avere uno scorcio della rete Bitcoin attraverso l'interfaccia che un server mette loro a disposizione.
Questi ultimi rinunciano completamente alla loro indipendenza, con conseguenze talvolta catastrofiche (si pensi ai già citati episodi dei providers MyBitcoin e Mt.Gox).
Per concludere questa analisi, è possibile affermare che la comunità di sviluppatori Bitcoin potrebbe dover rilassare la decentralizzazione dell'architettura, istituendo magari un gruppo di super nodi che svolgano parte delle funzioni dei full clients, al fine di ridurre il carico complessivo e favorire la scalabilità del sistema.

4 Aspetti analitici

In aggiunta alla costruzione del grafo delle transazioni menzionato in 3.2.2, gli aspetti maggiormente valutati nell'analisi dell'architettura Bitcoin, nonché punti critici del sistema, consistono probabilmente nel tempo medio necessario alla conferma di una transazione ed il guadagno medio, per transazione, di un miner che impiega risorse computazionali per aggregarle in un blocco valido.
Questi parametri sono indicativi della bontà del sistema e costituiscono le ragioni principali del suo utilizzo: il tempo necessario alla conferma non deve essere troppo alto e il guadagno dei miners proporzionale allo sforzo compiuto.
I grafici che seguono mostrano il variare dei sopracitati parametri in un periodo che va da Aprile 2013 ad Aprile 2014 e sono costruiti a partire da dati in formato CSV estratti da http://blockchain.info/stats.
L'ultima stima effettuata (20 Aprile 2014) mostra un tempo medio di 8 minuti; la media dell'ultimo anno è pari a 10 minuti, con picchi minimi e massimi di circa 5 e 15 minuti, rispettivamente.
Il tempo medio necessario alla conferma di una transazione rappresenta un punto critico del

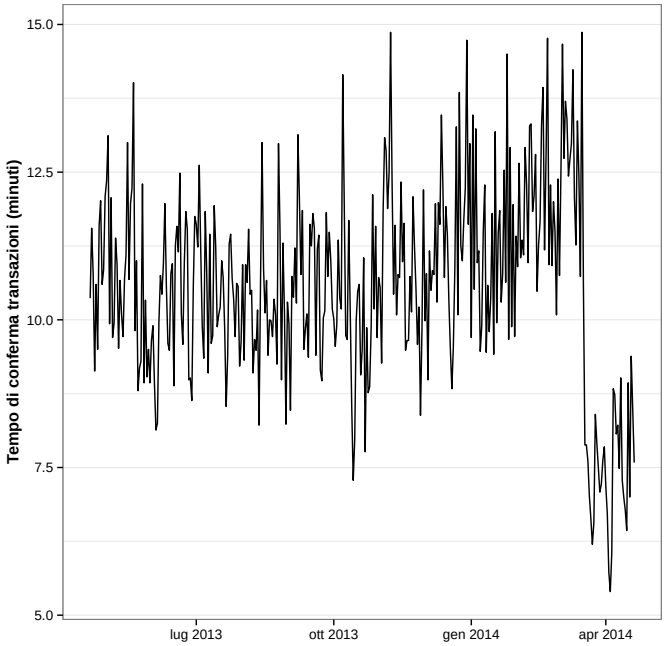

Figura 19. Tempo medio necessario a confermare una transazione

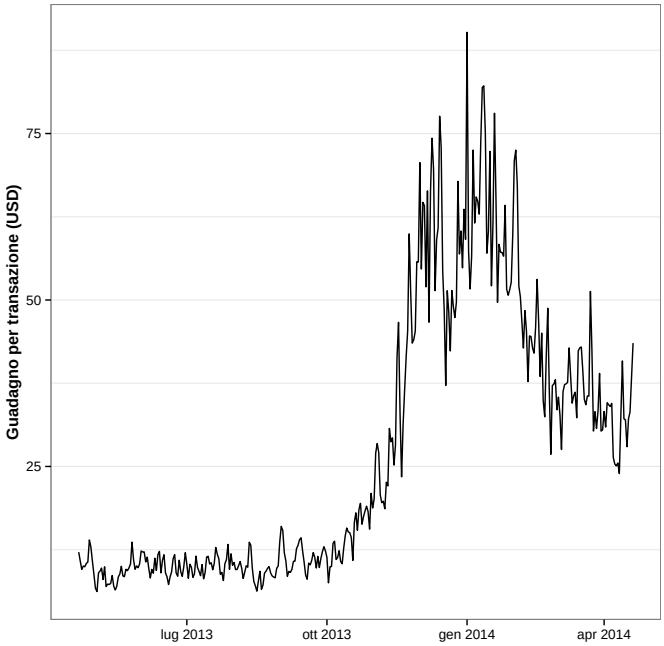

Figura 20. Guadagno medio di un miner, per transazione

sistema; in questo caso si parla di accettazione in un singolo blocco e non di una conferma statisticamente sicura, data da almeno 5 blocchi concatenati ad esso nella chain.
Questo parametro varia principalmente con il numero di nodi (e miners) attivi all'interno della rete Bitcoin e con la difficulty; quest'ultima, si ricorda, dipende dal numero di blocchi prodotti ogni 2 settimane e si adatta in maniera direttamente proporzionale a tale parametro: più blocchi sono prodotti, maggiore sarà la difficoltà nel produrre la serie successiva (questo spiega i bruschi sbalzi nella linea del grafico).
Il guadagno medio per transazione è pari a 44 USD per l'ultima stima effettuata (20 Aprile 2014); la media dell'ultimo anno è pari a 27 USD, con picchi minimi e massimi di circa 6 e 90 USD, rispettivamente.
Questo parametro non è solo influenzato dal numero di transazioni in circolo nella rete e dalle transaction fee al loro interno (utilizzate come incentivo per alcuni miners), ma principalmente dal valore del BTC in USD.
Probabilmente anche il fallimento di alcuni providers (Mt.Gox agli inizi del 2014) ha comportato una notevole fluttuazione nel suddetto valore.

5 ALTRE ARCHITETTURE

In questa sezione saranno presentate le architetture di alcuni sistemi di valuta elettronica alternativi a Bitcoin, sottolineandone le differenze principali e discutendone pregi e limitazioni.
Sebbene Bitcoin rimanga probabilmente la più utilizzata, esistono numerose criptovalute derivate.
Litecoin, per esempio, modifica il processo di mining utilizzando *scrypt* al posto di SHA-256: una funzione di derivazione di chiave basata su password che richiede una notevole quantità di memoria ma minore sforzo computazionale, rispetto alla controparte Bitcoin. **Vertcoin** modifica ulteriormente il processo aggiungendo una modifica adattiva nella memoria richiesta per computare scrypt, al fine di favorire CPU mining rispetto a quello tramite ASICs (3.3.5), dotati di memoria limitata.
Namecoin integra l'architettura Bitcoin con un DNS distribuito, consentendo di associare un

nome di dominio ad ogni blocco (in questo senso, la proof-of-work serve a validare anche il nome di dominio).
In aggiunta a queste leggere variazioni, esistono alternative a Bitcoin che propongono soluzioni molto differenti o specializzate in determinati aspetti; segue una descrizione di due delle più famose.

5.1 Ripple

Ideato da Ryan Fugger nel 2004 e attualmente mantenuto dal progetto Opencoin, Ripple [GR] rappresenta una delle alternative di maggiore successo a Bitcoin: un sistema monetario decentralizzato basato sul concetto di trust tra peers.
Ripple può essere utilizzato per scambiare qualunque tipo di moneta, fisica o virtuale (EUR, USD, BTC, . . .), includendo la propria valuta: i *ripples* (XRP).
Come Bitcoin, Ripple è una criptovaluta: le transazioni sono condotte mediante crittografia asimmetrica e firme digitali (ECDSA) e gli indirizzi corrispondono alle chiavi pubbliche degli utenti. La differenza principale risiede nel fatto che esse siano accettate dal beneficiario solo qualora il mittente sia nella sua lista di peers "fidati"; questa lista può essere estesa in qualunque momento e considera l'eventuale presenza di uno o più intermediari fidati fra le due parti.
Per facilitare le transazioni di utenti nuovi alla rete Ripple, è previsto l'uso di *gateway*: servizi commerciali che fungono da intermediari fidati nelle transazioni.
Sebbene il registro di tutte le transazioni effettuate sia pubblico e condiviso fra i nodi della rete, Ripple non utilizza uno schema proof-of-work per consentire ad un peer di attestare la validità di una o più transazioni, a differenza di Bitcoin.
Ogni utente possiede una *Unique Node List* (UNL): lista di entità distinte che, con alta probabilità, non coopererebbero nella falsificazione di una transazione. Ogni singola entità nella UNL potrebbe essere mossa da intenti fraudolenti, ma la loro presenza nella lista assicura che non lavorerebbero in gruppo per sovvertire la coerenza del registro.

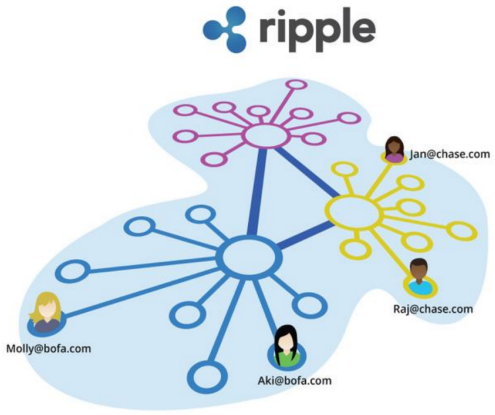

Figura 21. Ripple come sistema di pagamento federato

Una volta costruita questa lista, un nodo può utilizzarla per raggiungere il consenso collettivo: se riceve una nuova transazione dalla rete, gli basta verificare che la maggioranza di peers nella sua UNL l'abbia accettata.
Se una transazione è valida per un nodo (non è un tentativo di double-spending, per esempio) e lo è per la maggioranza della sua UNL, questa viene propagata ad altri: in questo modo il nodo esprime il suo voto positivo.
Poiché non esiste attività di mining, il sistema non prevede che i ripples siano prodotti dai peers: un totale di 100,000,000,000 XRP sono stati creati da Opencoin (*premined*) e, in piccola parte, distribuiti al pubblico, al fine di promuoverne l'uso.
Vale anche la pena menzionare che per creare un account è necessario depositarvi almeno 200 XRP (mediante conversione da altre valute, per esempio): questo limita notevolmente i meccanismi a supporto della privacy, quali lo schema 1 a 1 per coppie di chiavi e transazioni utilizzato da Bitcoin.
Sebbene Ripple sembri essere finalizzato a promuovere operazioni più veloci e meno dispendiose di computazione, eliminando del tutto il meccanismo di mining utilizzato da Bitcoin, la necessità di trust negli altri peers potrebbe rappresentare un suo punto debole, insieme alla controversa scelta di "centralizzare", allocando fin da subito l'intero capitale nelle mani degli sviluppatori, la produzione di ripples.

5.2 Ethereum

Presentata dal ventenne Vitalik Buterin alla fine del 2013 [But13], Ethereum consiste in una piattaforma software per gestione decentralizzata di transazioni tra utenti, inclusiva della criptovaluta *ether*.
Ethereum riprende numerosi concetti introdotti da Bitcoin: transazioni firmate digitalmente, proof-of-work, block chain, disseminazione delle informazioni fra i peers della rete, consenso collettivo per validare transazioni e diversi schemi transazionali tramite scripting.
In particolar modo è quest'ultimo concetto (contracts, vedi 3.5.3) che viene ampliato notevolmente: Ethereum utilizza un linguaggio di scripting Turing-completo incorporato al suo interno, permettendo la creazione di contracts complessi e applicazioni decentralizzate che specificano regole arbitrarie per il possesso degli ethers.
Il codice nei contracts è scritto in questo linguaggio bytecode stack-based, denominato *Ethereum virtual machine code* (EVMC), il quale consente costrutti preclusi a quello utilizzato da Bitcoin, quali i loops.
Le transazioni non sono costituite da input ed output, bensì dalla firma che identifica il mittente, un importo in ethers, un programma scritto in EVMC e due parametri chiamati *STARTGAS* e *GASPRICE*; il primo rappresenta il limite di steps computazionali riservati all'esecuzione del programma (onde evitare cicli infiniti) e il secondo è la *fee* da pagare al miner per ogni step. Se il limite viene superato tutti i cambiamenti derivati dalla transazione sono annullati.
A ricezione e accettazione di un blocco, un nodo esegue i programmi EVMC contenuti all'interno di ogni transazione nell'apposita macchina virtuale incorporata nel client. Questi programmi alterano lo stato della block chain, a differenza di quanto accade in Bitcoin, modificando le transazioni nei precedenti (per esempio sottraendo una quantità all'importo in ethers o alterando altre variabili).
In sostanza la block chain passa da uno stato all'altro, eseguendo gli scripts contenuti all'interno delle transazioni in ogni nuovo blocco.

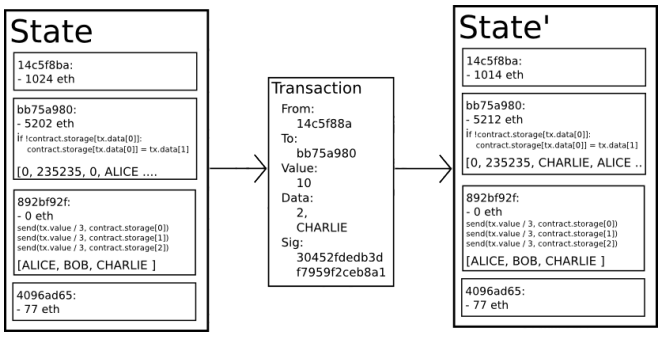

Figura 22. Cambiamenti di stato della block chain in Ethereum

Questo consente ad un client di memorizzare solo lo stato attuale (ethers posseduti da ogni indirizzo che abbia beneficiato di transazioni) e gli headers dei blocchi ricevuti, riducendo notevolmente il consumo di memoria.
L'obiettivo ultimo di Ethereum sembra essere quello di estendere il ridotto potere espressivo di Bitcoin nel definire complesse modalità di pagamento, nonché approntare alcune ottimizzazioni tecniche. Questo applicativo è ancora giovane, ma può rappresentare un valido concorrente a Bitcoin in futuro.

6 Conclusioni

L'obiettivo di questo documento è stato quello di fornire una rappresentazione di Bitcoin da un punto di vista architetturale.
Come prima cosa è stato descritto il contesto di utilizzo di Bitcoin, con i principali attori (utenti comuni, miners, providers di servizi) e proprietà rilevanti.
É stata in seguito descritta la struttura di un applicativo partecipante alla rete Bitcoin, comprensivo dei componenti necessari alla comunicazione con altri peers, gestione dei bitcoins posseduti dall'utente, visualizzazione della block chain e mining.
Sono state brevemente introdotte alcune variazioni del client originale, comprensive di determinati sottoinsiemi dei suoi componenti e rappresentanti una deviazione dal modello peer-to-peer decentralizzato originale: headers-only, signing-only, thin e mining clients.
É quindi seguita una descrizione delle operazioni principali compiute dal client: bootstrap-

ping, discovery di altri peers, gestione delle transazioni e mining.
Per concludere l'analisi dell'architettura ne è stato discusso il razionale ed è stato fatto cenno ad alcuni aspetti analitici.
Infine, sono state presentati alcuni sistemi di valute elettroniche alternative e maggiormente diffuse: Ripple ed Ethereum.
É possibile quindi affermare che l'evoluzione di Bitcoin, al fine di continuare a permetterne l'utilizzo a singoli utenti e senza l'appoggio di servizi esterni per la gestione dei wallets, presenterà numerose sfide: prima fra tutte la necessità di trovare un compromesso più adeguato fra efficienza operativa e mantenimento della decentralizzazione che lo ha reso così popolare.

10 maggio 2014.

BIBLIOGRAFIA

[AKR+13] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. *Evaluating user privacy in bitcoin*. In Financial Cryptography and Data Security, pages 34–51. Springer, 2013.

[Ara14] Giulia Arangüena. *Bitcoin. L'altra faccia della moneta*, volume 8. goWare, 2014.

[But13] Vitalik Buterin. *[English] White Paper: A Next-Generation Smart Contract and Decentralized Application Platform*. ethereum / wiki on GitHub (Self-published), 2013.

[DPSHJ] Joan Antoni Donet Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartı. *The Bitcoin P2P network*.

[DW13] Christian Decker and Roger Wattenhofer. *Information propagation in the bitcoin network*. In Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on, pages 1–10. IEEE, 2013.

[GKCC] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. *Is Bitcoin a Decentralized Currency?*

[GR] Patrick Griffin and Philip Rapoport. *Ripple: A Primer*.

[Nak08] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Consulted, 1:2012, 2008.

[OKH13] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. *Structure and anonymity of the bitcoin transaction graph*. Future Internet, 5(2):237–250, 2013.

[RH13] Fergal Reid and Martin Harrigan. *An analysis of anonymity in the bitcoin system*. In Security and Privacy in Social Networks, pages 197–223. Springer, 2013.

[Sku12] Rostislav Skudnov. *Bitcoin clients*. B.s. thesis, Turku University of Applied Sciences, 2012.

SITOGRAFIA

[12] Bitcoin wiki. https://en.bitcoin.it/wiki, 2014. [Online; accessed 14-April-2014].

[13] Ken Shirriff. Bitcoins the hard way: using the raw bitcoin protocol. http://www.righto.com/2014/02/bitcoins-hard-way-using-raw-bitcoin.html, 2014. [Online; accessed 22-April-2014].

[14] Charlie Stross. Why i want bitcoin to die in a fire. http://www.antipope.org/charlie/blog-static/2013/12/why-i-want-bitcoin-to-die-in-a.html, 2013. [Online; accessed 12-April-2014].

[15] Wikipedia. Bitcoin — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Bitcoin&oldid=606742319, 2014. [Online; accessed 10-April-2014].