

UNIVERSITÀ DEGLI STUDI DI UDINE

---

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Triennale in Informatica

Tesi di Laurea

BITCOIN  
MONETA ELETTRONICA  
PEER-TO-PEER

./img/uniud\_lite-eps converted to .pdf

Relatore:  
Prof. IVAN SCAGNETTO

Laureando:  
MATTEO PAOLUZZI

---

ANNO ACCADEMICO 2012-2013



Ai miei genitori  
per non avermi tagliato i viveri,  
a Serena  
per il continuo e incessante supporto



---

## **Sommario**

Verrà descritta la struttura e il funzionamento della rete Bitcoin, un sistema monetario decentralizzato virtuale. Per prima cosa si procederà ad un raffronto tra le altre tipologie di reti P2P e la rete Bitcoin, evidenziandone le differenze e il perché tale rete sfugga ai normali criteri di catalogazione, pur rientrandone sotto alcuni punti di vista ben specifici. Verrà poi analizzata la rete nello specifico, illustrandone scopi, funzionamento, utilizzi e criticità, queste ultime soprattutto a confronto con le altre tipologie di rete nei casi attinenti. Infine verranno trattati in modo informale alcuni temi di carattere socio-economico collegati all'utilizzo di Bitcoin, analizzando brevemente alcune vicende di cronaca che negli ultimi anni hanno avuto tra i protagonisti tale rete.



---

## **Abstract**

English abstract





# Indice

<b>1</b>	<b>Panoramica generale sulle reti P2P</b>	<b>1</b>
1.1	Distribuzione dei file in una rete P2P . . . . .	2
1.1.1	Scalabilità ( [13]) . . . . .	3
1.1.2	Ricerca di informazioni . . . . .	5
<b>2</b>	<b>Bitcoin: moneta elettronica decentralizzata</b>	<b>9</b>
2.1	Moneta Elettronica . . . . .	9
2.2	Portafogli e indirizzi . . . . .	10
2.3	Transazioni . . . . .	11
2.4	Timestamp e Proof-of-Work . . . . .	13
2.5	Network . . . . .	15
2.6	Risorse necessarie . . . . .	16
2.7	Gestione dei valori . . . . .	19
2.8	Analisi della rete . . . . .	20
2.8.1	Topologia . . . . .	21
2.8.2	Propagazione delle informazioni . . . . .	21
2.8.3	Informazioni scomparse . . . . .	25
2.9	Fork della blockchain . . . . .	26
2.9.1	Creazione del modello . . . . .	26
2.9.2	Misurazioni . . . . .	28
<b>3</b>	<b>Privacy</b>	<b>33</b>
3.1	Analisi quantitativa della privacy . . . . .	33
3.1.1	Il modello antagonistico . . . . .	34
3.1.2	Quantificazione della Privacy . . . . .	34
3.2	Valutazione della privacy in Bitcoin . . . . .	39
3.2.1	Euristiche per sfruttare il client . . . . .	39
3.2.2	Analisi basata sul comportamento . . . . .	39
<b>4</b>	<b>Sicurezza</b>	<b>41</b>
4.1	Reti P2P in genere . . . . .	41
4.1.1	Attacchi di Basso Livello . . . . .	41
4.1.2	Attacchi di Medio Livello . . . . .	43

---

4.1.3	Attacchi al livello P2P . . . . .	45
4.1.4	Contromisure generali . . . . .	47
4.2	La rete Bitcoin . . . . .	47
4.3	I Portafogli . . . . .	47
4.4	Stock Exchange . . . . .	47
<b>Bibliografia</b>		<b>49</b>

# Capitolo 1

## Panoramica generale sulle reti P2P

Negli ultimi anni si è assistito ad una progressiva alterazione delle leggi che regolano il mondo dell'hardware informatico in campo consumer: l'aumento della potenza di calcolo del singolo elaboratore non è più economicamente conveniente. Per fortuna la soluzione è stata immediata e consiste nello sfruttamento di più elaboratori collegati in rete che condividono (genericamente parlando) risorse. Sebbene per compiti specifici risulta spesso conveniente creare una rete ad-hoc, per l'utilizzo di tutti i giorni da parte dell'utente comune, la rete per eccellenza risulta essere senza ombra di dubbio la rete Internet. A causa di questa sua centralità, è stata scelta come base per lo sviluppo di applicazioni dedicate alla condivisione di risorse di vario tipo da parte di utenti con interessi in comune, creando di fatto una sottorete virtuale all'interno della già virtuale rete internet.

Questa condivisione di risorse da parte di utenti per un interesse comune definisce il nucleo di quelle che vengono chiamate **reti Peer-to-Peer**, da qui in avanti abbreviate come *reti P2P*. Data la grande diffusione di queste reti e i loro svariati obiettivi, è comprensibile che ci siano molti disaccordi sulla definizione esatta di *rete P2P*.

Una classificazione molto adatta agli scopi di questo documento è quella presente in [20] che distingue tre diversi livelli di rete:

1. **Infrastrutture P2P**, il cui scopo è porre le basi per i livelli successivi fornendo funzioni di comunicazione, integrazione e “traduzione” tra le varie componenti della rete. In particolare forniscono servizi che permettono la localizzazione e la comunicazione tra gli utenti (da ora in avanti **peers**) e l'identificazione, l'utilizzo e lo scambio delle risorse, oltre che l'implementazione delle politiche di sicurezza quali autenticazione e autorizzazione.
2. **Applicazioni P2P**, che utilizzano i servizi offerti dal livello di infrastruttura per offrire all'utente (qui inteso come essere umano interagente con la macchina, la quale è il peer vero e proprio) le funzionalità della rete. Sono in pratica le interfacce tra l'infrastruttura e la persona.

- 
3. **Fenomeni sociali** derivanti dall'utilizzo delle applicazioni P2P. Spesso infatti la forte coesione di intenti tra gli utenti di una rete P2P porta alla nascita di comunità virtuali, centri di aggregazione e correnti di pensiero che esulano dalla sfera prettamente informatica.

Come si vedrà più avanti nel corso della trattazione, la rete Bitcoin è caratterizzata da tutti e tre i livelli in modo molto più peculiare di altre reti P2P maggiormente diffuse e famose.

## 1.1 Distribuzione dei file in una rete P2P

La risorsa indubbiamente più abbondante e facilmente condivisibile è lo spazio di archiviazione, per questo le reti P2P in cui vengono condivisi file sono tra le più diffuse e variegate. La loro diffusione è talmente ampia che spesso l'intero concetto di rete P2P viene ridotto a quello di rete P2P per file-sharing, motivo per cui la classificazione delle reti P2P intese in senso generico spesso si fonda sul metodo di condivisione dei file, anche quando questo non è lo scopo della rete.

Indugiando in questa generalizzazione, studieremo un caso tipico: lo scaricamento di un file attraverso il modello classico client-server e alcune reti P2P ad ampia diffusione.

Prima di cominciare è necessario chiarire un concetto che spesso è causa di confusione: il modello P2P non è una alternativa al modello client-server, bensì una sua reimplementazione meno gravosa per il server.

Infatti, se nel modello client-server “puro” i ruoli sono definiti ed immutabili dall'inizio della comunicazione fino al suo completamento, quando si comunica in P2P i ruoli sono relativi al collegamento esistente tra i peer.

Prendiamo il caso in cui il peer **A** voglia ottenere dal peer **B** il file *file.txt*. All'inizio della comunicazione il peer **A** sarà il client e il peer **B** sarà invece il server. Immaginiamo che, mentre questo trasferimento è in atto, un terzo peer **C** voglia avere il file *file.txt*.

Se ci troviamo al di fuori di una rete P2P (ad esempio nel normale download di un file da internet tramite browser), i ruoli di **A** e **B** non subiscono variazioni e il peer **C** assume il ruolo di client del server **B**, il quale dovrà ottimizzare le sue risorse di banda e cpu per servire sia **A** che **C** contemporaneamente. NOTA:(da questo discorso esulano tematiche quali il multitasking della cpu: il punto di vista è quello di un ipotetico utente che osserva la macchina in tempo reale).

All'interno di una rete P2P invece, nel momento in cui comincia a scaricare *file.txt*, **C** è consapevole che **B** ne possiede una copia completa e **A** una parziale e, contemporaneamente, **A** verrà informato che **C** sta scaricando lo stesso file. Quello che succede è che **B** farà da server per **A** e **C** (i quali saranno i suoi client), mentre **A** e **C** si scambieranno le parti di *file.txt* che mancano l'uno all'altro (ovvero saranno sia client che server contemporaneamente). Il risultato è una sostanziale ottimizzazione delle risorse a disposizione nella rete e una maggiore velocità di “diffusione” del file.

---

Il linguaggio sopra adottato è forzatamente generico: ciò deriva dall'ampio numero di reti P2P per il filesharing esistenti, ognuna delle quali implementa a modo suo le casistiche di aggiunta/rimozione (*churn*) di un peer (chiamato solitamente **nodo** nell'ambito del file sharing) dalla rete, la ricerca dei file e il trasferimento dei contenuti, tutti comunque rispettando il procedimento generico sopra descritto.

### 1.1.1 Scalabilità ( [13] )

Analizziamo la questione da un punto di vista più formale. Avremo bisogno dei seguenti dati.

$u_s$  frequenza di upload verso il server

$u_i$  frequenza di upload dell' $i$ -esimo peer

$d_i$  frequenza di download dell' $i$ -esimo peer

$F$  dimensione in bit del file da distribuire

$N$  numero di peer che vuole una copia del file

$D_{cs}$  tempo di distribuzione del file per l'architettura client-server

Per semplificare i conti senza invalidarne l'efficacia, assumiamo che la rete in esame sia priva di "disturbi" e sia dedicata esclusivamente allo scambio di file in esame senza altre comunicazioni passanti su di essa.

#### Caso Client-Server

Osservazioni:

- Il server deve trasmettere il file a  $N$  peer, quindi  $NF$  bit. Data la frequenza di upload  $u_s$ , il tempo per distribuire il file deve essere almeno  $NF/u_s$ .
- sia  $d_{min} = \min\{d_1, d_2, \dots, d_N\}$  la frequenza di download del peer con il valore più basso. Tale peer riceverà il file in almeno  $F/d_{min}$  secondi, che è quindi il tempo minimo di distribuzione.

Da cui

$$D_{cs} \geq \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{min}} \right\}$$

Questo è il limite inferiore al tempo di distribuzione minimo per l'architettura client-server. Trattiamo il caso ottimo e consideriamolo come il tempo di distribuzione effettivo, ovvero:

$$D_{cs} = \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{min}} \right\}$$

---

Da questa ultima espressione si vede come per  $N$  sufficientemente grande, il tempo di distribuzione è dato da  $NF/u_s$ , stabilendo quindi che esso aumenta linearmente all'aumentare del numero  $N$  dei peer.

### Caso P2P

La situazione cambia nel caso di architetture P2P, in cui ciascun peer assiste il server nella distribuzione del file. Dato che bisogna tenere conto di come ogni singolo peer distribuisce le sue porzioni di file, il calcolo risulta molto complesso. Possiamo però ottenere una semplice espressione del tempo minimo di distribuzione. A tale scopo bisogna fare alcune osservazioni:

- All'inizio dell'analisi solo il server possiede il file completo. È quindi necessario inviare ogni singolo bit del file all'interno della rete almeno una volta perché sia trasmesso alla comunità, quindi il minimo tempo di distribuzione è almeno  $F/u_s$ . La prima differenza è già visibile: dato che i peer ridistribuiscono i file, non è necessario che il server invii due volte lo stesso bit.
- Vediamo anche che il peer con la frequenza di download più bassa non può ottenere il file in meno di  $F/d_{min}$  secondi.
- Infine osserviamo come la capacità di upload della rete è quella del server più quella di ciascun peer. Denotando quest'ultimo con  $u_{tot}$  e sapendo che il numero totale di bit da trasferire è  $FN$ , possiamo stabilire che il tempo di distribuzione minimo è almeno  $FN/u_{tot}$ .

Unendo le affermazioni precedenti in una unica formulazione otteniamo il tempo minimo di distribuzione per una rete P2P:

$$D_{P2P} \geq \max \left\{ \frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\}$$

### Confronto

Per un confronto diretto tra le due architetture, poniamo  $F/u = 1$  ora,  $u_s = 10u$  e  $d_{min} \geq u_s$ , ovvero abbiamo posto che un peer può trasmettere l'intero file in un'ora, la frequenza di trasmissione del server è 10 volte quella del peer e (per semplificare) le frequenze di trasmissione dei peer sono grandi abbastanza da essere irrilevanti. Otteniamo la seguente figura.

Si nota subito che per l'architettura client-server il tempo di distribuzione risulta essere lineare come calcolato, ma ancora più notevole è il tempo di distribuzione per l'architettura P2P che non solo è sempre minore della controparte client-server, ma addirittura risulta quasi costante all'aumentare del numero dei peer. Questo dimostra come le reti P2P siano estremamente scalabili, e non c'è da stupirsi come il successo di una particolare rete dipenda essenzialmente dal numero di peer presenti. %FIXME: questa è detta molto male.

---

### 1.1.2 Ricerca di informazioni

Si è parlato all'inizio della sezione di come i peer siano consapevoli di cosa altri peer stanno scaricando. Questo sottintende che esiste un sistema all'interno della rete per stabilire quanti peer sono connessi e cosa stanno condividendo e/o scaricando. Ciò significa che tale meccanismo si deve attivare ogniqualvolta un peer si (dis)connette (d)alla rete e un file viene rimosso/aggiunto. L'operazione di aggiunta/rimozione dinamica di un peer/file prende il nome di **churn**. Vediamo quindi come è possibile ricercare informazioni all'interno di una rete P2P quindi, nello specifico del file-sharing, vediamo come sia possibile per un peer sapere quali file sono a disposizione presso quali altri peer.

#### Directory centralizzata

La prima, grande applicazione P2P commercializzata (*Napster*) faceva uso di un indice centralizzato localizzato su un potente server centrale. Quando un utente si collegava alla rete, l'applicazione Napster informava tale server dell'indirizzo IP della macchina e del nome dei file che l'utente ha scelto di condividere con la rete. Il server raccoglie quindi queste informazioni da tutti peer collegati e aggiorna il proprio indice collegato ciascun nome di file agli indirizzi IP dei peer che in quel momento lo stanno condividendo. Ovviamente alla disconnessione di un peer l'indice sarà aggiornato di conseguenza.

Si noti come questa sia di fatto un'architettura ibrida: client-server nella ricerca dei contenuti, P2P nel trasferimento degli stessi.

La semplicità di questo metodo nasconde però alcuni gravi difetti. Il server dell'indice si trova infatti ad essere l'anello debole della rete: un suo guasto ha come conseguenza il crollo dell'intera rete, impedendo di fatto le ricerche. Inoltre tale server deve potenzialmente gestire milioni di utenti contemporaneamente (in quanto si è visto che la potenza delle reti P2P sta nel numero di peer) e quindi risulta essere estremamente costoso da installare e mantenere, oltre a rappresentare un potenziale collo di bottiglia per l'intera rete (Napster era infatti nota per i gravi problemi di traffico da cui era afflitta).

Esiste inoltre un terzo problema non strettamente tecnico ma fondamentale per l'argomento di questo documento: **la privacy dell'utente**. Con un indice centrale che associa un file ad una serie di IP, è facile per chiunque vedere chi possiede cosa, soprattutto nel caso in cui i file condivisi rappresentino una violazione del diritto d'autore. Pur essendo il caricamento dei file una responsabilità dei singoli utenti e non di chi fornisce il servizio, molte legislazioni internazionali possiedono leggi per la tutela del copyright che possono costringere il proprietario del server ad interrompere il servizio, bloccando così l'intera rete, anche quei peer che condividevano materiale lecito. Questo controllo a volte molto invasivo (se non dannoso) da parte di terze parti non coinvolte nella rete è uno dei motivi che ha portato alla nascita di reti sempre più decentralizzate, come appunto la rete Bitcoin.

---

## Query flooding

Rimanendo nell'ambito della ricerca di informazioni per file-sharing, la soluzione diametralmente opposta alla precedente risiede nell'approccio completamente distribuito del query flooding, implementato originariamente dal protocollo **Gnutella**. Questo approccio prevede che l'indice sia distribuito all'interno della comunità, in particolare ogni peer indicizza solamente quello che intende condividere con gli altri peer.

Nel query flooding viene quindi a formarsi una rete astratta e virtuale che collega i vari peer, una rete di copertura (*overlay network*) in cui i collegamenti tra i nodi non sono di tipo fisico. %fixme: questo è vero anche per napster in effetti..

Il protocollo Gnutella prevede che ogni peer sia collegato al massimo a dieci altri nodi dell'overlay network, nodi che vengono denominati *vicinato*. Se un peer vuole cercare un file, invia messaggi di ricerca ai suoi vicini, i quali instradano tali messaggi ai loro vicini che ripetono l'operazione e così via. Questo è il processo del *query flooding* in cui la rete viene "sommersa" (*flood*) di richieste (*query*). Quando un peer riceve una query, controlla se nel suo indice è presente una qualche corrispondenza e, in caso affermativo, invia al peer che ha effettuato la richiesta un messaggio di successo (*query-hit*) contenente il nome del file e la dimensione. La query-hit viene inviata al nodo che ha effettuato la ricerca seguendo lo stesso percorso seguito dal pacchetto query. Dopo qualche tempo il peer di origine avrà un elenco di peer che condividono file corrispondenti alla sua richiesta.

Ma anche in questo approccio, la semplicità nasconde alcune problematiche. La più grave riguarda le ricerche, che vengono propagate all'intera rete di copertura generando una enorme quantità di traffico nella rete sottostante (nel caso di Gnutella, Internet) che connette i peer. Come soluzione a tale problema si è inserito nei messaggi di ricerca di Gnutella un campo *time-to-live*, il quale viene decrementato da ogni peer prima di reinoltrare la richiesta: quando il campo arriva a 0 la richiesta non viene più inoltrata, limitando quindi il raggio di azione della query. Tuttavia in questo modo si riduce anche il numero di peer contattati e la probabilità di ricevere una query-hit.

Esiste inoltre una difficoltà intrinseca alle reti P2P che Napster aveva avitato: il churm. Il protocollo Gnutella tenta di risolvere il problema in questo modo:

1. Per prima cosa, un peer, chiamiamolo X, che vuole connettersi alla rete deve conoscere almeno un altro peer che è già connesso alla rete (**problema del bootstrap**). I due approcci più comuni consistono nel mantenere in ogni peer una lista di client noti a cui tentare di connettersi e/o, in mancanza di connessione a tali nodi, nello scaricare una lista di nodi attualmente online da un sito *tracker*.
2. Dopo aver ottenuto i nodi di bootstrap, X deve tentare di connettersi ad ognuno di questi nodi, fino a riuscirci con uno che chiameremo Y.



- 
3. Dopo la connessione, X invia ad Y un messaggio di *ping* contenente un campo contatore di peer. Y inoltra tale messaggio a tutte i nodi nella sua rete di copertura, i quali continuano l'inoltro fino a quando il contatore non si azzerà.
  4. Ogni volta che un peer Z riceve il *ping*, risponde inviando un messaggio *pong* contenente il proprio IP attraverso la rete di copertura fino ad X.
  5. Grazie ai messaggi *pong*, X viene a conoscenza di molti altri peer online (quanti dipende dal contatore impostato nel messaggio *ping*) e può tentare di connettersi ad essi per ampliare il suo vicinato.

### Copertura gerarchica

Avendo descritto gli approcci diametralmente opposti, vediamo ora la proverbiale via di mezzo, il modello di copertura gerarchica. L'obiettivo è ottenere il meglio dalle due implementazioni sopra descritte, e fu per la prima volta realizzato da FastTrack, un protocollo implementato in Kazaa e Morpheus. Una variante di questo modello è tutt'oggi utilizzata dall'evoluzione di Gnutella, Gnutella2. Come per il query flooding, anche questo è un modello decentralizzato, ma a differenza di tale modello (e a discapito del principio Peer-to-Peer) non tutti i peer sono uguali: come il nome fa suggerire, esiste una gerarchia di peer che assegna a quelli più potenti (leggasi, quelli con più banda) alcune responsabilità in più designandoli come leader per altri peer. Ogni nuovo peer deve stabilire una connessione con un leader a cui notifica tutti i file che intende condividere, creando quindi una sorta di mini indice centralizzato rispettivamente ai peer connessi a quel leader (solitamente nell'ordine del centinaio). A differenza del modello centralizzato però, i leader non sono server dedicati, bensì peer veri e propri in collegamento tra di loro. Il risultato è che i leader sono collegati ad una rete di copertura del tutto simile a quella del modello query flooding, modello utilizzato infatti per l'inoltro delle ricerche. La ricerca di un file da parte di un peer segue quindi due fasi: una richiesta al proprio leader che provvede a consultare il suo indice e una eventuale propagazione della richiesta tramite query flooding da parte del leader agli altri leader. Questa struttura "a strati" permette quindi la connessione di molti peer senza generare una quantità eccessiva di traffico nella rete "ospite".

### DHT (Distributed Hash Table)

Crea un indice completamente distribuito che fa corrispondere gli identificatori dei file alla loro posizione. Consente agli utenti di determinare tutte le posizioni di un file senza generare un'eccessiva quantità di traffico di ricerca. Overnet (Kademlia) sfrutta DHT come anche BitTorrent.



## Capitolo 2

# Bitcoin: moneta elettronica decentralizzata

Tutte le reti P2P finora descritte sono in circolazione da molti anni, hanno una base di utenti che conta milioni di peer divisi tra utenti reali e server automatizzati, contano migliaia di forum di supporto e scambiano quotidianamente una immensa fetta del traffico totale della rete Internet (tanto che molti provider tendono a limitarne quanto più possibile l'utilizzo, soprattutto nelle fasce orarie di maggior traffico). Sono però tutte reti dedicate al file-sharing. Bitcoin no. O almeno, non proprio, come vedremo.

Bitcoin è una rete P2P (intesa per tutti e tre i livelli descritti in precedenza) che mira a creare un sistema di valuta digitale privo di controllo centrale, con pagamenti effettuati direttamente tra gli utenti senza l'intervento di terzi. È stata ideata e realizzata in origine da un anonimo noto con il nome di **Satoshi Nakamoto** [16] e si è in poco tempo evoluta in modo esponenziale fino a catturare di recente l'attenzione dei media internazionali, delle banche mondiali e, per alcuni suoi utilizzi illeciti, da FBI ed NSA. Ma vediamo di cosa si tratta.

### 2.1 Moneta Elettronica

**Bitcoin** è il nome dato alla rete, al client originale, al protocollo di comunicazione e alla moneta utilizzata per le transazioni all'interno della rete.

Le monete (d'ora in avanti **btc**) posso essere ottenute “gratuitamente” dopo aver impegnato la propria CPU o GPU in alcuni calcoli di crittografia (operazione chiamata **mining** e discussa più avanti), oppure acquistate da altri utenti della rete tramite una valuta reale <sup>1</sup>.

Entrambi questi metodi sono fondamentali nell'ecosistema Bitcoin:

- Il mining è una diretta conseguenza della partecipazione di un nodo alla rete (vedi più avanti NETWORK) ed è l'unico metodo con cui vengono create e

---

<sup>1</sup>Ad esempio tramite il sito Internet Mt. Gox [3].

---

messe in circolazioni nuove **btc**. Per ogni operazione di mining avvenuta con successo si riceve una quantità fissa di btc che viene dimezzata nel tempo, limitando il numero massimo di bitcoin in circolazione a circa 21 milioni. A Gennaio 2013 è stata generata circa la metà delle monete totali e si stima di arrivare a 3/4 nel 2017. La quota massima di moneta circolante e l'assenza di istituti centrali in grado di creare nuove monete rendono l'economia bitcoin invulnerabile all'inflazione che colpisce le economie reali. Il sistema è ispirato a quella che era l'economia del Dollaro prima della istituzionalizzazione della Federal Reserve come banca federale: il valore del Dollaro era legato al valore corrente dell'oro, il quale esiste in quantità limitata ed è ottenibile solo attraverso il lavoro dei minatori.

- La compravendita di bitcoin è invece simile alle compravendite di azioni effettuate nelle borse di tutto il mondo. Esistono infatti alcuni luoghi dedicati (ad esempio il sito internet *Mt. GOX*) che fungono da stock exchange offrendo agli utenti la possibilità di mettere in vendita o di acquistare bitcoin al prezzo che preferiscono. Sono delle vere e proprie borse che trattano unicamente bitcoin invece che molti titoli di aziende diverse, calcolano un valore di scambio medio basato sulle ultime transazioni portate a termine ma lasciano libero l'utente di scegliere a quanto vendere o comprare bitcoin, con prezzo medio che si adegua di conseguenza.

Come per le monete in valuta reale e le azioni borsistiche, anche le bitcoin vengono “tenute” in portafogli. Come vedremo, il termine *tenute* è usato impropriamente, ma questa è l'apparenza dal punto di vista dell'utente, per cui a tale apparenza al momento ci atterremo.

## 2.2 Portafogli e indirizzi

La prima volta che un nuovo utente avvia il suo client Bitcoin fresco di installazione, si vede assegnato un portafoglio contenente un indirizzo e una copia di chiavi di cifratura simmetrica. L'indirizzo è semplicemente una stringa di 33 caratteri alfanumerici che inizia con un 1 o con un 3, generata in modo casuale dalle chiavi create per l'utente. Gli indirizzi rappresentano il punto di uscita e/o il punto di ingresso per tutti i movimenti che coinvolgono bitcoin. Questo significa che nelle transazioni bitcoin compariranno unicamente questi indirizzi, rendendo di fatto **anonimi** tutti i movimenti di bitcoin (meno quelli che riguardano l'acquisto di bitcoin tramite moneta reale), in un modo del tutto equivalente a quello dei conti in Svizzera. Non esiste quindi nessuna correlazione diretta e ovvia tra un utente e il suo indirizzo. Essendo le chiavi associate al portafogli, e gli indirizzi generati dalle chiavi, viene di conseguenza pensato (giustamente) che un portafogli possa contenere più indirizzi: basta infatti usare le chiavi per generare un nuovo indirizzo e il gioco è fatto. Il numero di indirizzi esistenti è virtualmente infinito.

---

I portafogli sono solitamente legati al software che li crea, basta quindi cambiare software per poter creare un nuovo portafogli e una nuova coppia di chiavi (oppure installare un software in grado di gestire più portafogli). In alternativa è possibile “aprire un conto” presso numeri siti che offrono questa funzionalità, quale ad esempio blockchain.info. In questo caso però bisogna fare i conti con la sicurezza del sito in questione

## 2.3 Transazioni

Le transazioni rappresentano il nucleo fondamentale di bitcoin. Esse sono il metodo con cui, a livello astratto, le bitcoin vengono trasferite da un account all'altro e tramite la loro analisi ci si assicura che un indirizzo contenga esattamente quel numero di bitcoin, che una bitcoin non venga spesa più volte e che quella bitcoin appartiene a quello specifico indirizzo. Le transazioni si basano su meccanismi di crittografia a chiave pubblica, rendendo quindi obsoleto il coinvolgimento di terze parti nella transazione. Se infatti nelle normali compravendite online, volenti o nolenti si è costretti a fidarsi di terze parti che garantiscono per il buon esito dell'operazione (istituti di credito, compagnie di carte di credito, siti come Paypal, ecc), qui gli utenti hanno direttamente la prova crittografica senza aver quindi necessita di fidarsi di qualcuno.

Satoshi Nakamoto descrive la sua moneta elettronica come una serie di firme digitali. Il trasferimento di moneta da un utente all'altro avviene infatti applicando la firma digitale dell'acquirente ad un hash di una precedente transazione e della chiave pubblica del venditore, e aggiungendo ciò alla fine della moneta.

Riassumendo schematicamente:

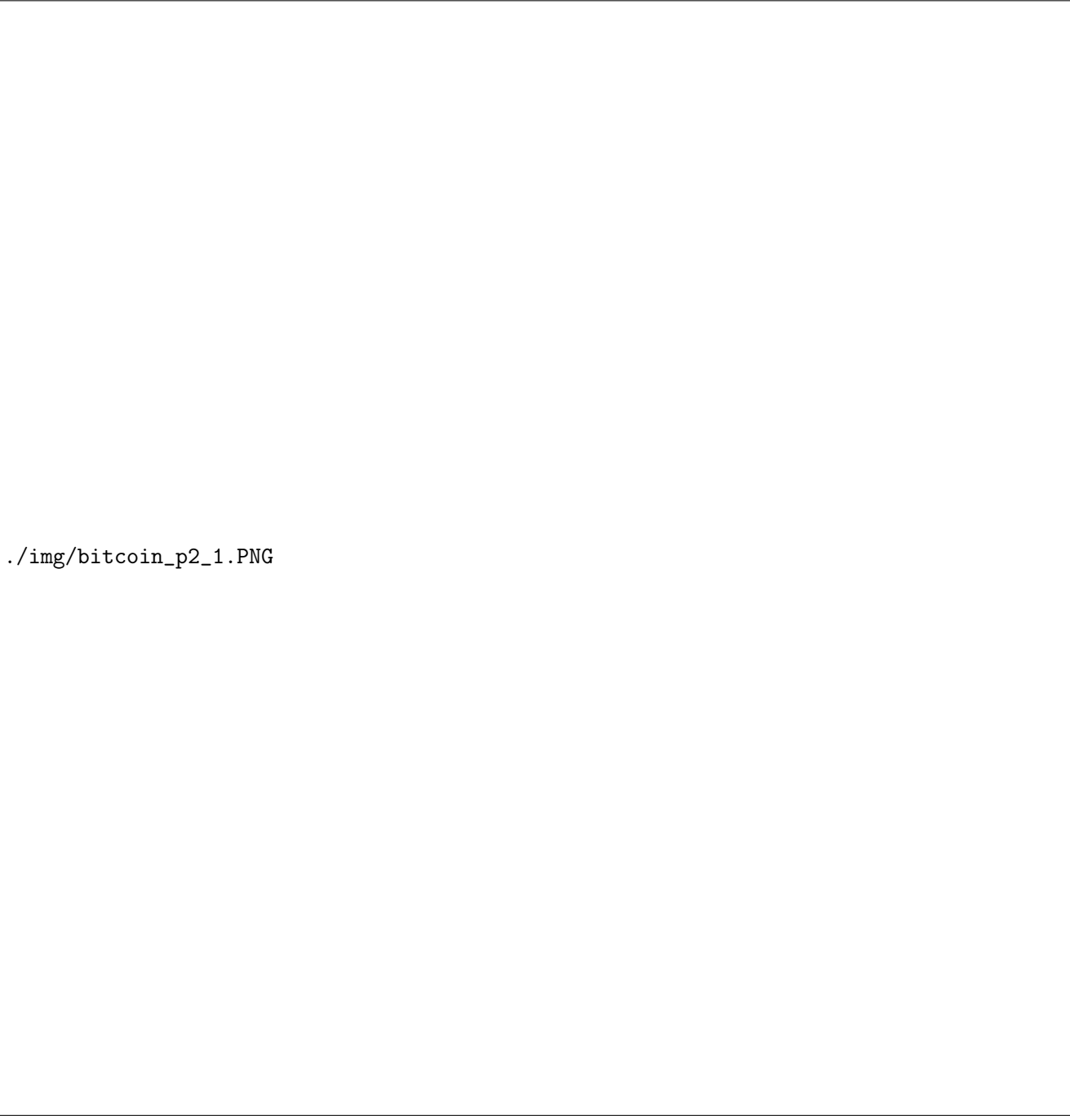
```
hash = hash(previous_transaction, vendor_public_key)
transaction = sign(hash, private_key)
```

La moneta diventa quindi non una unità atomica, ma il risultato di una serie di transazioni che coinvolgono firme digitali e verifiche che deve essere calcolato dinamicamente. La serie di tutte le transazioni mai effettuate viene raccolta in una sequenza denominata **blockchain**.

Questa implementazione però non garantisce che l'acquirente non abbia già effettuato una transazione con questa moneta, ovvero che stia spendendo una moneta già spesa in precedenza.

L'unico modo per garantire ciò senza utilizzare una terza parte di cui fidarsi, è tenere conto di **tutte** le transazioni. Questo vuol dire che tutte le transazioni devono essere annunciate ad un pubblico in grado di mettersi d'accordo sull'effettivo ordine temporale in cui sono state effettuate. Il venditore deve avere quindi la prova che, nel momento in cui riceve la transazione, la maggioranza dei nodi è d'accordo che quella è la prima transazione ricevuta.

---



./img/bitcoin\_p2\_1.PNG

Figura 2.1: Utilizzo delle chiavi e degli hash in una sequenza di transazioni.

---

## 2.4 Timestamp e Proof-of-Work

La soluzione consiste nell'utilizzo di un **timestamp server**. Un timestamp server funziona calcolando l'hash di un blocco di oggetti di cui si vuole realizzare il timestamp e rendendo tale hash pubblico. Il timestamp dimostra inequivocabilmente che gli oggetti esistevano al momento dell'hashing. Ogni timestamp include anche il precedente timestamp nell'hash, formando quindi una catena in cui ogni timestamp rinforza <sup>2</sup> quelli precedenti.

Ora il problema consiste nell'implementare questo server di timestamp in modo distribuito, come è appunto la rete Bitcoin. Per prima cosa bisogna trovare un sistema per cui effettuare il timestamp è un'operazione difficoltosa (computazionalmente parlando), ma verificare che il timestamp sia corretto deve essere immediato. Basandosi sul lavoro di Adam Back ([6]), Nakamoto ha deciso che la difficoltà dell'operazione deve essere trovare un valore che, una volta sottoposto ad hashing (ad esempio con SHA-256), il risultato sia un hash che comincia con uno specifico numero di bit pari a zero: la difficoltà del lavoro è esponenziale al numero di bit zero richiesti, ma è facilmente verificabile con un singolo hash. L'implementazione per bitcoin consiste quindi nella creazione di un blocco di dati di cui calcolare l'hash che contiene le transazioni interessate, l'hash precedente e un valore chiamato **nonce** da incrementare fino a quando l'hash non avrà le caratteristiche richieste. Modificare una transazione comporta modificare un blocco, e quindi ripetere tutto il lavoro di calcolo della nonce. Inoltre, se a questo blocco è già stato incatenato uno più blocchi successivi, anche tali blocchi andranno ricalcolati in sequenza, rendendo il lavoro estremamente gravoso.

Con la prova di lavoro si risolve anche il problema di cosa significa che la maggioranza deve accettare un timestamp. Con l'hash infatti si realizza una sorta di sistema one-CPU-one-vote, e la "decisione della maggioranza" è rappresentata dalla più lunga sequenza di timestamp, che è la sequenza per la quale è stata impiegata la maggior parte di lavoro computazionale. Ciò significa che se la maggior parte della forza-CPU è controllata da peer onesti (cioè che non hanno nessuna intenzione di modificare una transazione effettuata), un nodo disonesto che volesse modificare una transazione non solo dovrebbe rifare tutti i calcoli per il blocco della transazione e per tutti i blocchi successivi, ma avendo minor potenza di CPU a disposizione rispetto ai nodi onesti, verrebbe rapidamente soverchiato dal numero di calcoli da fare, in quando il numero di blocchi da ricalcolare sarebbe sempre superiori a quelli da lui già ricalcolati.


Si capisce subito che è nella rete bitcoin (e nelle reti P2P in generale) è importante che le risorse (potenza di calcolo in questo caso) siano equamente distribuite tra i peer, in modo da evitare che un solo nodo o un solo gruppo di nodi controlli l'intera rete.

Per far fronte alle differenti configurazioni hardware degli utenti, alla sempre crescente capacità di calcolo di CPU e GPU e anche ai potenzialmente mutevoli

---

<sup>2</sup>leggasi: rende più difficili da modificare.

---



./img/bitcoin\_p3\_1.PNG

Figura 2.2: Struttura minimale di una sequenza di blocchi.



---

interessi dei nodi, la difficoltà della prova di lavoro (ovvero il numero di bit zero) è determinata da una media calcolata sul numero medio di blocchi generati ogni ora. Se vengono generati troppi blocchi, vuol dire che la difficoltà è troppo bassa e viene subito aumentata.

## 2.5 Network

A questo punto abbiamo una prima approssimazione di come funziona la rete bitcoin:

1. Le nuove transazioni sono inviate a tutti i nodi.
2. Ogni nodo raccoglie le transazioni che riceve in un blocco.
3. Per ogni blocco, ogni nodo cerca di calcolare una proof-of-work.
4. Una volta trovata la prova, invia il blocco a tutti i nodi.
5. I nodi accettano il nuovo blocco se e solo se tutte le transazioni in esso sono valide (si verifica calcolando l'hash delle transazioni e confrontandole con l'ultimo blocco accettato) e non già spese in precedenza.
6. Il nodo esprime la sua accettazione del blocco appena arrivato mettendosi al lavoro per crearne uno nuovo, usando l'hash del nodo accettato.

I nodi considerano la catena più lunga quella corretta (e viene definita *blockchain*) e lavoreranno sempre in modo da prolungarla. Esiste la possibilità che uno stesso nodo riceva due versioni diverse dello stesso blocco in contemporanea. In questo caso, lavoreranno sul primo blocco ricevuto, ma manterranno una copia anche dell'altro nel caso in cui si rivelasse appartenente alla catena più lunga. La verifica viene fatta non appena viene trovata la nuova proof-of-work e una delle due catene si allunga: a questo punto si individua il blocco da mantenere in base all'hash contenuto nel blocco appena arrivato, gli altri blocchi vengono scartati e si continua il procedimento. Inserendo un po' di terminologia, chiamiamo il primo blocco mai realizzato (che è codificato all'interno di ogni client Bitcoin) *blocco genesi*, l'ultimo blocco della catena *blocco di testa*, la distanza tra un blocco  $b$  e il blocco genesi viene definita *altezza*.

Quando si inviano in broadcast le nuove transazioni, non è necessario che esse raggiungano tutti i nodi: fintanto che raggiungono quanti più nodi possibile, verranno velocemente inglobate in un blocco. I blocchi invece devono essere ricevuti da tutti i nodi, per questo se un nodo riceve un blocco e si accorge (tramite hash) che il blocco precedente gli manca, ne richiederà immediatamente una copia ad un altro nodo, e ripeterà la verifica fino ad ottenere la catena integrale.

Per convenzione, la prima transazione di un blocco è una transazione speciale che crea una nuova moneta e la assegna al creatore del blocco. In pratica il primo nodo che riesce a trovare la proof-work di un nuovo blocco riceve un premio in bitcoin. Tale premio serve ad incentivare i nodi a mantenere attiva la rete ed inoltre permette la messa in circolazione di nuove monete, senza che sia necessaria una zecca centrale. Questo profitto derivante dal calcolo della proof-of-work viene denominato **mining**

---

e ha spinto molti utenti ad acquistare hardware dedicato sempre più performante in modo da creare per primi il nuovo blocco e ottenere il relativo premio, inizialmente ammontante a 50btc ma ora dimezzato a 25btc).

Visto che il numero massimo di btc è stato determinato a priori ed è invariabile, per incentivare i nodi anche quando il mining risulterà inutile, sono state introdotte delle vere e proprie tasse di transazione, le **transaction fees**. Una transaction fee non ha un valore fisso, ma viene decisa da chi effettua la transazione e può anche essere nulla: viene registrata come una differenza tra il valore di input e il valore di output della transazione, con quest'ultimo valore inferiore del primo (come in una tassa, l'acquirente spende più dell'importo effettivo). Tale differenza verrà trasferita nella transazione dedicata all'incentivo al momento della creazione del nuovo blocco, sempre ammesso che l'utente che ha creato il blocco voglia ricevere queste btc.

Oltre ad invogliare un nodo a rimanere attivo, gli incentivi incoraggiano i nodi a rimanere onesti: infatti se un attaccante avido riuscisse ad accumulare abbastanza potenza di calcolo da surclassare quella di tutti gli altri nodi, potrebbe scegliere se truffare gli altri nodi ritirando i suoi pagamenti precedenti oppure usarla per accumulare nuove monete con gli incentivi. Un attaccante si vede quindi incoraggiato a mettere la sua potenza di calcolo a favore del sistema facendogli guadagnare più btc di tutti gli altri nodi messi insieme, invece di usare la stessa potenza per minare le basi dello stesso sistema in cui egli stessi investe i propri soldi.

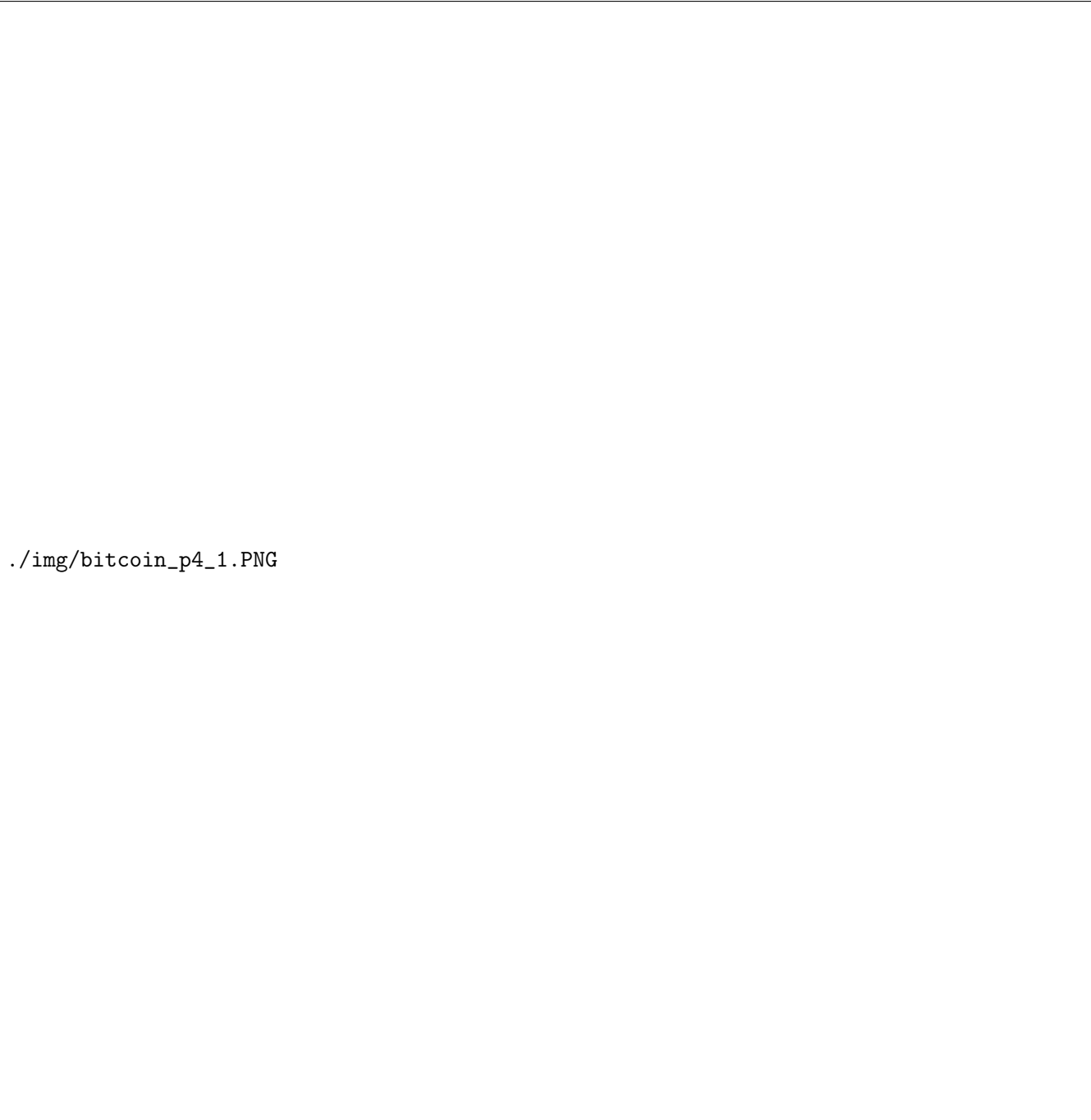
## 2.6 Risorse necessarie

Il sito Blockchain.info [2] offre vari servizi agli utenti bitcoin, tra i quali spiccano un portafogli online, un sistema di navigazione dell'intera catena dei blocchi e dettagliate statistiche sull'intera rete. Da tale sito si vede come in media, in 24 ore vengono effettuate 56700 transazioni archiviate in 220 blocchi, il che vuol dire un blocco ogni 6.55 minuti. Se ogni nodo dovesse mantenere ogni singola transazione, lo spazio di memoria occupato renderebbe la rete bitcoin non così appetibile per l'utente medio. Risulta necessario minimizzare la quantità di memoria necessaria a mantenere la blockchain senza compromettere la sicurezza.

Si è detto infatti che l'hash di un blocco viene calcolato a partire dall'hash del blocco precedente, da una nonce e dalle transazioni contenute nel blocco. Ma invece che memorizzare le intere transazioni, esse vengono inserite come foglie di un albero di Merkle [14], una struttura dati in cui ogni nodo non foglia è l'hash di tutti i suoi figli, in modo che nella radice di tale albero ci sia un solo hash che riassume in se tutte le transazioni. È pertanto possibile calcolare l'hash di un blocco basandosi unicamente sull'hash del blocco precedente, sulla nonce e sulla radice dell'albero di Merkle, ovvero sull'hash riassuntivo delle transazioni, rendendo quindi possibile eliminare alcune transazioni dalla blockchain per risparmiare un poco di spazio.

Con questa struttura, il **block header** viene ad occupare esattamente 80 Bytes (vedi sezione tecnica per i dettagli), il che significa circa 6.2 MB all'anno (Satoshi Nakamoto con una stima di un blocco ogni 10 minuti aveva previsto 4.2 MB annui).

---



./img/bitcoin\_p4\_1.PNG

Figura 2.3: Struttura di un blocco prima e dopo la compressione dell'albero delle transazioni.

---

Una quantità decisamente ridotta che rende la blockchain tollerabile su ogni computer degli ultimi 7 anni.

È possibile quindi verificare i pagamenti senza disporre di un nodo personale. L'utente deve possedere una copia degli header della catena più lunga (che può ottenere dai nodi della rete) e ottenere il ramo di Merkle che collega la transazione al blocco in cui è stata inserita. Non può verificare la transazione da solo calcolando gli hash (non ha le altre foglie dell'albero di Merkle), ma collegandola ad un punto specifico della catena può vedere che un nodo l'ha accettata, ed eventuali blocchi successivi confermano che anche l'intera rete l'ha accettata.



./img/bitcoin\_p5\_1.PNG

Figura 2.4: Struttura di un blocco prima e dopo la compressione dell'albero delle transazioni.

Così come è descritta, la verifica è affidabile fin tanto che la rete è controllata da nodi onesti, ma è vulnerabile se la rete è soverchiata da un attaccante. Mentre un nodo può effettuare le verifiche da se, il metodo semplificato è vulnerabile alle

---

transazioni ad-hoc create da un attaccante che controlla la rete. Una strategia di difesa consiste nell'accettare avvisi dai nodi della rete quando questi rilevano blocchi non validi, richiedendo all'utente di scaricare l'intero blocco. Per questo motivo chi utilizza bitcoin per business ritiene preferibile avere un nodo personale con intera blockchain, in modo da poter effettuare verifiche autonomamente e velocemente.

Da questo si può dedurre un fatto importante che distingue la rete bitcoin dalle altre reti P2P: si può partecipare alla rete anche senza il relativo software (in questo senso, si partecipa al livello comunitario della rete): non serve essere un nodo, basta essere un utente.

## 2.7 Gestione dei valori

Anche se è possibile trattare le monete una ad una, è improponibile fare una transazione per ogni singolo centesimo. Per la divisione degli importi, le transazioni contengono molteplici input e output. In una transazione normale si avranno o un singolo input proveniente da una grande transazione precedente oppure più input provenienti da più transazioni piccole precedenti, e al massimo due output: uno per il pagamento vero e proprio e uno per restituire il resto, se presente, al mittente.

Chiarimo meglio il concetto:

- Un input è un riferimento ad un output di una precedente transazione che contiene l'indirizzo di chi sta effettuando la transazione. Se ci sono più input, l'importo degli output da loro referenziati viene sommato ed il totale è il massimo valore utilizzabile dall'output.
- Un output contiene l'indirizzo del destinatario e il valore da spedire. Dato che, in una futura transazione, un output può essere referenziato da un solo input, potrebbe verificarsi il caso in cui l'input sia maggiore dell'output e della transaction fee desiderata. In questo caso è necessario creare due output, uno con il valore da spedire e l'indirizzo del destinatario, l'altro con la differenza da restituire al mittente, il **resto**. La differenza tra il totale degli input e il totale degli output è la transaction fee.

La situazione è visibile in 2.5.

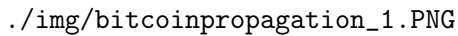
Nonostante la stretta dipendenza tra le varie transazioni, non è necessario estrarre l'intero background di ogni input, in quanto la transazione verrà accettata solo se il blocco che contiene le transazioni con gli output referenziati dagli input è stato accettato dalla rete.

Dato il valore elevato di una singola btc (che può variare da poche decine a migliaia di dollari) e le difficoltà inerenti nel trattare numeri in virgola mobile su un computer, l'unità di misura base della transazione non è il btc ma il Satoshi<sup>3</sup>, e  $1 \text{ BTC} = 100000000 \text{ Satoshi}$ .

---

<sup>3</sup>dal "nome" dell'inventore di Bitcoin

---



./img/bitcoinpropagation\_1.PNG

Figura 2.5: Schema di una transazione in cui vengono evidenziati input e output.

Questo sistema del calcolo del valore di una transazione è lo stesso utilizzato dal software che gestisce il portafoglio per calcolare il proprio valore: per ogni indirizzo all'interno del portafoglio, il software scansiona le transazioni presenti nella blockchain che contengono l'indirizzo in esame, somma i valori entranti nell'indirizzo, sottrae quelli uscenti e ricava il valore “contenuto” nell'indirizzo. Usando questo sistema, i possedimenti di un utente sono temporalmente limitati all'ultimo blocco accettato nella catena, e non è quindi possibile spendere moneta ricevuta da una transazione non ancora approvata <sup>4</sup>.

%%%%%%%%%% % La documentazione di Satoshi prevede anche privacy e calcoli statistici sugli attacchi. % Visto che prevedo una sezione apposta per anonimato e sicurezza, ne parlo la e non qua. % Attenzione che la parte di topologia successiva nomina alcuni risultati statistici di questa parte. . . una volta scritta saranno da fare i collegamenti

## 2.8 Analisi della rete

Sappiamo che i nodi della rete Bitcoin sono tutti omogenei e nessuno di essi ha un ruolo di coordinatore o comunque diverso da quello degli altri nodi, e ognuno

---

<sup>4</sup>tecnicamente, nessuna moneta è ricevuta fin tanto che la transazione non è approvata. L'utente non è nemmeno consapevole della transazione a lui destinata fino ad avvenuta approvazione.

---

di essi mantiene una copia di tutte le informazioni necessarie per far funzionare il sistema. Vediamo ora più formalmente come si struttura la rete Bitcoin e come le informazioni (ovvero, le transazioni e i blocchi) si propagano in essa.

### 2.8.1 Topologia

Non essendoci coordinazione tra i nodi, il grafo rappresentante la rete ha una struttura casuale. Durante il *churn* il nuovo nodo interroga alcuni server DNS gestiti da nodi volontari e si vede restituiti un insieme casuale di nodi con cui fare il bootstrap. Una volta connesso, impara dai suoi vicini gli indirizzi dei nodi raggiungibili e si mette in ascolto nel caso nuovi nodi vengano annunciati in broadcast. A differenza di una rete P2P “tradizionale”, non esiste alcun modo per un nodo di lasciare la rete: gli indirizzi restano memorizzati per diverse ore prima che gli altri nodi li rimuovano dalla loro lista di indirizzi noti.

Ogni nodo tenta di mantenere un certo numero  $p$  di connessioni con gli altri nodi, collegandosi ad un indirizzo scelto a caso tra quelli che conosce nel caso il numero di connessioni sia inferiore a  $p$  ma senza bloccare connessioni in ingresso nel caso tale numero sia superiore. Il valore  $p$  rappresenta quindi un valore minimo di connessioni spesso e volentieri superato per nodi abilitati ad accettare connessioni in ingresso. Per il client *bitcoind*, il primo implementato da Nakamoto e tutt’ora il più diffuso, il default è  $p = 8$ , ma il numero medio di connessioni contemporanee è 32 nel caso in cui non ci siano firewall o NAT <sup>5</sup> ad intercettare connessioni esterne.

Dato il tipo di connessione tra i nodi, le partizioni non sono individuabili nel momento in cui si creano e, nel caso in cui dovessero esistere, esse continuerebbero ad operare indipendentemente. Una tale situazione comporta nel tempo una divergenza tra le situazioni tracciate dalle due partizioni, situazioni potenzialmente incompatibili. Pertanto è fondamentale individuare le partizioni nel momento in cui si creano, e uno dei modi per farlo è tracciare il potere computazionale complessivamente presente nella rete: una rapida diminuzione della frequenza di creazione di blocchi può indicare la presenza di una partizione.

### 2.8.2 Propagazione delle informazioni

Per ciò che riguarda il mantenimento della blockchain, solamente i messaggi di transazione  $tx$  e i blocchi sono rilevanti. Tali messaggi sono molto più comuni di tutti gli altri scambiati nella rete e potrebbero raggiungere dimensioni rilevanti. Per evitare di sprecare banda e ridurre l’overhead, è necessario fare in modo di inviare ognuno di questi messaggi una sola volta per ogni nodo, evitando quindi l’inoltro ai nodi che già hanno ricevuto quel messaggio.

L’implementazione utilizzata da Bitcoin sfrutta una tecnica passiva: invece che inoltrare tutti i messaggi, ogni nodo dopo aver verificato un messaggio o una transa-

---

<sup>5</sup>Network Address Translator: maschera gli indirizzi di una LAN come un unico indirizzo IP sulla rete Internet, e spesso impedisce a host presenti in Internet di connettersi ad host specifici in una LAN.

---

zione invia a suoi vicini un messaggio *inv* per segnalare la disponibilità di nuovi tx o blocchi. Il messaggio *inv* contiene gli hash dei blocchi e delle transazioni disponibili per l'invio, che ogni vicino può richiedere nel caso gli mancasse tramite un messaggio *getdata*. I messaggi *inv* e *getdata* (61 B) sono ovviamente significativamente più piccoli dei messaggi *tx* e *block* (fino a 500 kB per i blocchi).

Ogni volta che si verifica lo scambio di *inv* e *getdata*, la velocità di propagazione del messaggio subisce un rallentamento, dovuta sia allo scambio di messaggi sia alla necessità di verificare la transazione/blocco con calcoli crittografici: bisogna verificare ogni nodo prima di notificarne la disponibilità, e tale verifica include la verifica di ogni transazione nel blocco, le quali a loro volta richiedono accesso random ai dati del disco rigido.

Vediamo di stimare formalmente come si propaga un dato nella rete.

Definiamo  $t_{i,j}$  come il tempo trascorso tra il primo annuncio e il momento in cui il nodo  $j$  riceve l'oggetto  $i$ . Il nodo  $o$  sarà l'origine dell'oggetto  $i$  (ovvero il nodo che ha trovato il blocco o il nodo che ha creato la transazione), quindi  $t_{i,o} = 0$ .

Il tempo  $t_{i,j}$  ha un comportamento **doppio esponenziale**. Il periodo di propagazione segue due fasi: una prima crescita esponenziale in cui la maggior parte dei nodi che ricevono un *inv* richiederanno il relativo dato che non possiedono, e una seconda fase di compressione esponenziale in cui la maggior parte dei nodi che ricevono un *inv* hanno già il dato.

Per effettuare le loro misurazioni, i ricercatori Christian Decker e Roger Wattenhofer dello Swiss Federal Institute of Technology di Zurigo hanno implementato il protocollo bitcoin su un nodo creato in modo tale da non inviare messaggi *inv*, *tx* o *block* e collegato a direttamente a quanti più nodi tradizionali possibile. Tale implementazione ha il solo scopo di tracciare come i blocchi si propagano nella rete restando in ascolto dei messaggi *inv* che ne annunciano la disponibilità [10]. La ricezione di un *inv* implica che il nodo che ha inviato il messaggio ha ricevuto e verificato un blocco.

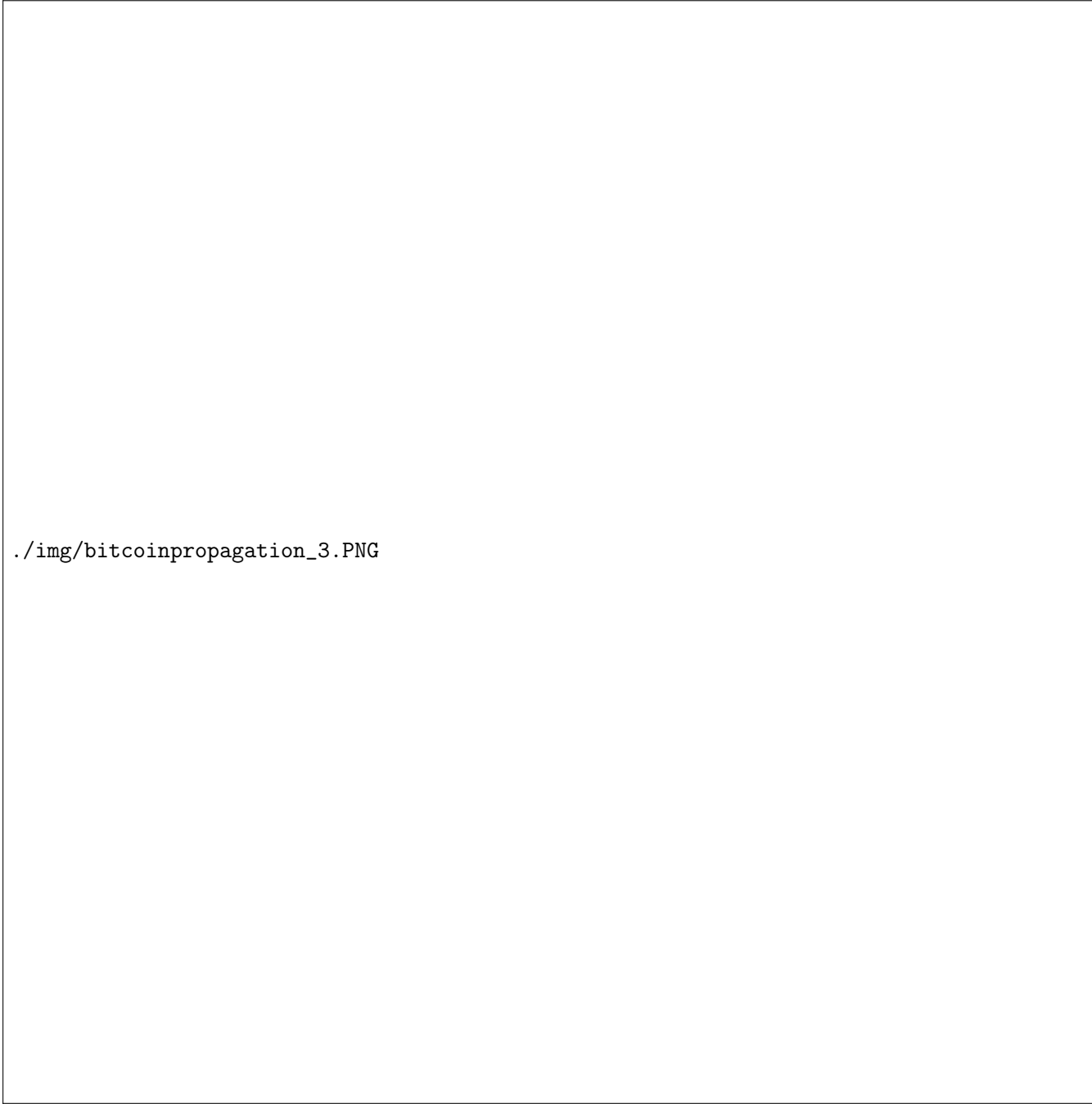
La rilevazione è partita su nodi di altezza 180000 ed è durata per 10000 blocchi. Le informazioni rilevate includono l'hash del blocco, l'IP del noto annunciante e una timestamp della ricezione dell'*inv*. La stima per  $t_{i,j}$  è ottenuta sottraendo il timestamp del primo annuncio di un blocco da tutti gli annunci successivi per quel blocco. I risultati sono riassunti nel grafico 2.6.

Le misurazioni effettuate ci permettono di stabilire che il tempo mediano in cui un nodo riceve un blocco è di 6.5 secondi, mentre la media è di 12.6 secondi. La lunghezza della seconda fase, quella di contrazione esponenziale, evidenzia come dopo soli 40 secondi il 95% dei nodi abbia ricevuto il blocco.

Abbiamo però detto come la dimensione di un blocco sia variabile, fino a 500kB. I dati precedenti sono aggregati per tutti i blocchi e non tengono conto delle differenti dimensioni degli stessi. Perciò definiamo ora il *costo di ritardo* come il ritardo che ogni kB causa alla diffusione di una transazione o di un blocco. Tale costo risulta essere una combinazione sia del tempo di trasmissione che del tempo di verifica. I risultati sono contenuti nel grafico 2.7.



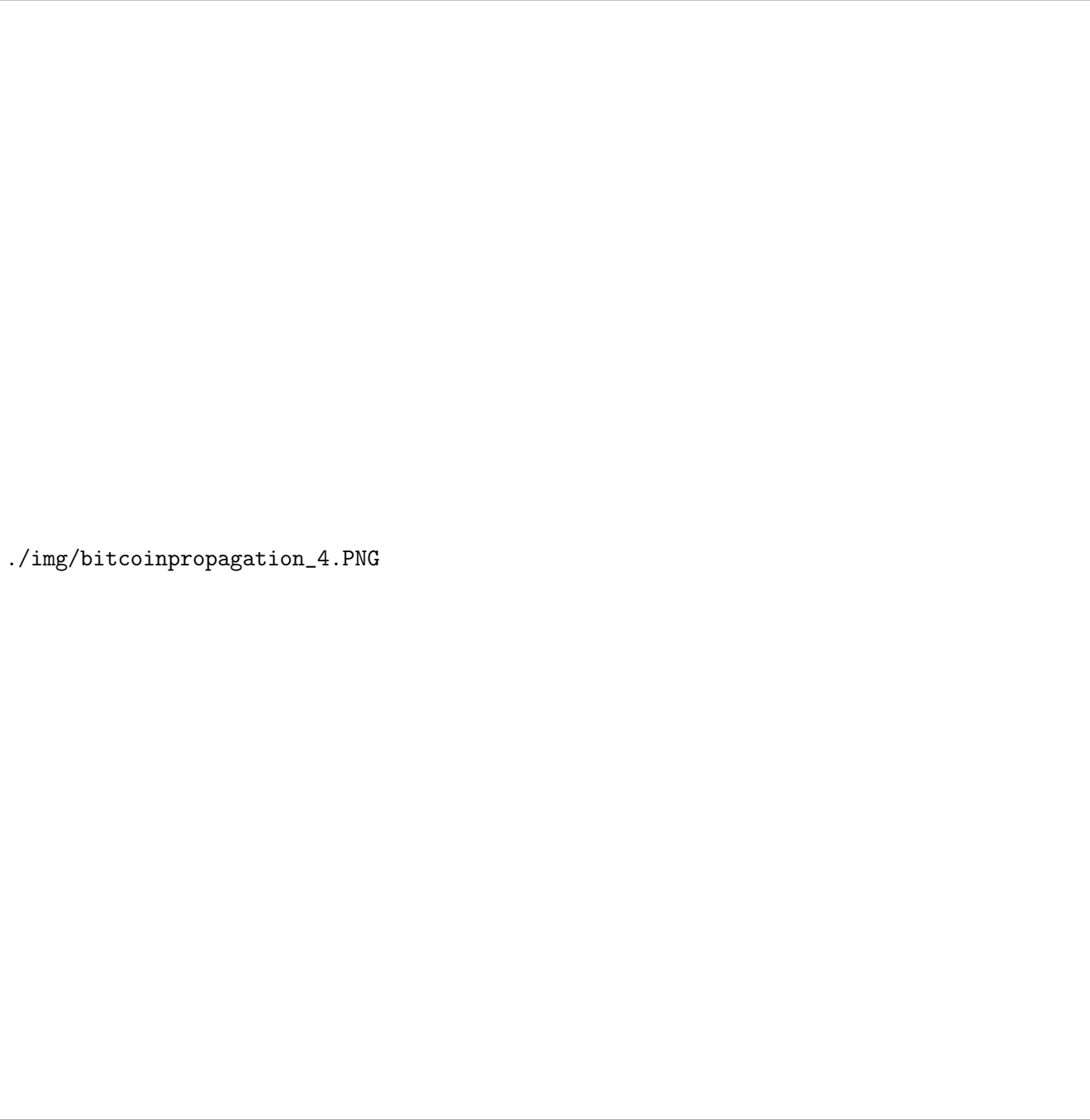
---



`./img/bitcoinpropagation_3.PNG`

Figura 2.6: Istogramma normalizzato del tempo che intercorre dal primo annuncio del blocco con una curva di interpolazione esponenziale.

---



`./img/bitcoinpropagation_4.PNG`

Figura 2.7: Costo del ritardo per il 50°, 75° e 90° percentile. Il grafico è focalizzato sui valori più bassi delle ascisse per poter mostrare il comportamento costante presente dopo i 20kB.

---

Per pacchetti di dimensione superiore ai 20kB si vede come il costo sia pressoché costante, mentre per dimensioni minori si assiste a notevole ritardo. La causa di ciò sta nel **ritardo da un roundtrip**, ovvero il fatto che anche i piccoli messaggi vengono annunciati con *inv* e richiesti con *getdata*. Il roundtrip è dominante per le transazioni in quanto il 96% di tutte le transazioni sono inferiori ad 1kB. Per i blocchi, la cui dimensione è per la maggior parte superiore ai 20kB, ogni kB di dimensione costa circa 80ms di ritardo fino al momento in cui la maggioranza dei nodi non ha il blocco. Nel caso di piccoli blocchi sarebbe pertanto ottimale evitare l'annuncio e inoltrare direttamente il blocco ai nodi vicini.

### 2.8.3 Informazioni scomparse

Trattiamo ora il caso in cui un blocco inoltrato in rete porti ad un fork della blockchain che viene rilevato solo da una piccola parte dei nodi.

Definiamo il grafo che descrive la rete come  $G = (V, E)$ , con  $V$  i nodi (vertici) ed  $E$  le connessioni tra i nodi (archi/edges). Definiamo inoltre la partizione  $P_h \subset V$  come l'insieme dei nodi il cui blocco di testa si trova ad altezza  $h$ . Trovare un nuovo nodo  $b_{h+1}$  crea una nuova partizione  $P_{h+1,b}$  contenente i nodi che considerano questo nuovo blocco come blocco di testa, in altre parole questo è il primo blocco di altezza  $h+1$  che abbiano ricevuto. Se non vengono trovati altri blocchi, allora i nodi di  $P_h$  adiacenti a  $P_{h+1,b}$  si uniscono a  $P_{h+1,b}$  lasciando (e quindi eliminando) la partizione  $P_h$ . Dal'altro canto, se viene trovato un blocco  $b'_{h+1}$  da un nodo in  $P_h$  viene creata una nuova partizione  $P_{h+1,b'}$ . Anche in questo caso i nodi di  $P_h$  abbandoneranno la loro partizione per unirsi ad una delle due nuove di altezza superiore. Solamente i nodi di  $P_h$  che sono a contatto sia con  $P_{h+1,b}$  che con  $P_{h+1,b'}$  saranno consapevoli dell'esistenza di entrambe le partizioni e quindi del fork della blockchain, e considereranno invalido il blocco di altezza  $h+1$  proveniente dall'altra partizione e di conseguenza non lo annunceranno ai loro vicini fermando quindi l'espansione della partizione. Tale meccanismo si applica anche alle transazioni: se due tx tentano di spendere lo stesso output, solo la prima transazione ricevuta da un nodo verrà considerata valida, la seconda verrà considerata invalida e non annunciata ai vicini. Questo comportamento ha il vantaggio di impedire ad un nodo malevolo di inondare la rete con centinaia di transazioni contraddittorie senza costo addizionale, in termini di transaction fees, per il nodo malevolo. Il rovescio della medaglia è che questo sistema di nascondere le informazioni ritenute sbagliate da un nodo permettere l'implementazione di **attacchi double spend** che risultano invisibili ai commercianti. Nel caso delle transazioni, dato che esse non devono per forza propagarsi a tutti i nodi, il meccanismo descritto è ragionevole e protegge la rete da transazioni spam. Nel caso dei blocchi invece, fermarne la propagazione è controproducente: i fork della blockchain, che con tale sistema sono invisibili per la maggior parte dei nodi, sono una cartina tornasole che indica come nella rete ci siano alcune inconsistenze non risolte. Dato che i blocchi valido ma potenzialmente in conflitto non possono essere creati con un frequenza arbitraria come le transazioni, permettere il loro inoltro anche nel caso di conflitto non crea possibilità per un potenziale attacco.

---

## 2.9 Fork della blockchain

Durante il normale utilizzo della rete potrebbe capitare di essere testimoni di un fork se si ricevono due blocchi conflittuali, ma osservare tutti fork che avvengono è molto difficile a causa della non propagazione dei blocchi in conflitto discussa in precedenza. Se a questo aggiungiamo il fatto che una partizione potrebbe avere dimensione unitaria (un nodo genera un nuovo blocco in conflitto con il blocco di testa di tutti i suoi vicini) è evidente come per poter rilevare tutti i fork bisognerebbe connettersi ad ogni nodo della rete. Ma abbiamo detto che alcuni nodi non sono raggiungibili dall'esterno, per cui possiamo solo stimare il numero di fork che avvengono.

Utilizzando la configurazione descritta nella sezione precedente, sono stati raccolti tutti i blocchi di altezza compresa tra 180000 e 190000. Essendo un grande campione che coinvolge tutti i nodi raggiungibili, è abbastanza probabile che tutti i blocchi generati siano stati propagati fino al nodo spia implementato permettendo di individuare la maggior parte dei fork avvenuti nell'intervallo di rilevazione. Nei 10000 blocchi osservati sono stati identificati 169 fork, il che si traduce in rateo di forking  $r = 1.69\%$ . L'istogramma 2.8 mostra i risultati in modo più dettagliato.

### 2.9.1 Creazione del modello

Il protocollo bitcoin adatta la difficoltà della proof-of-work ogni 10 minuti in modo da mantenerla sufficientemente elevata da essere significativa. Definendo  $X_b$  come la variabile casuale che rappresenta i secondi trascorsi tra il ritrovamento di un nodo e il ritrovamento del nodo precedente, allora la *probabilità che un blocco venga trovato* nella rete in un dato secondo è

$$P_b = \Pr[X_b < t + 1 | X_b \geq t] \approx 1/600$$

Un fork avviene se, durante la propagazione del blocco  $b$ , viene trovato un blocco  $b'$  in conflitto nella parte di rete non ancora a conoscenza di  $b$ . Definiamo  $t_j$  come il tempo in secondi in cui  $j$  apprende dell'esistenza di  $b$  da quando esso è stato creato. La funzione  $I_j(t)$  identifica se il nodo  $j$  sa dell'esistenza di  $b$  nell'istante  $t$ , e la funzione  $I(t)$  conta il numero di nodi che hanno ricevuto e verificato  $b$  all'istante  $t$ .

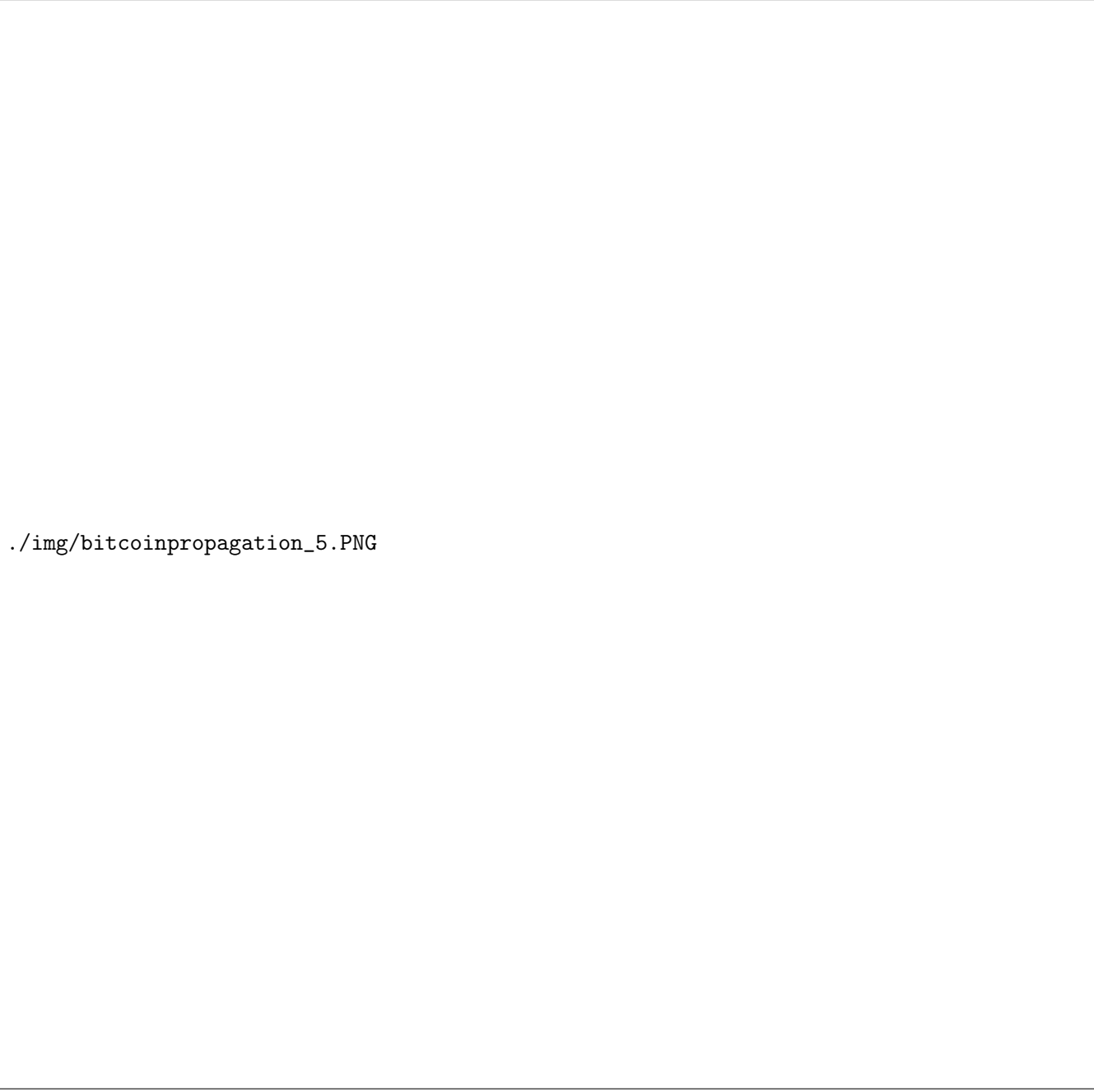
$$I_j(t) = \begin{cases} 0 & \text{se } t_j > t \\ 1 & \text{se } t_j \leq t \end{cases}$$

$$I(t) = \sum_{j \in V} I_j(t) \quad \text{con } V \text{ insieme dei vettori}$$

Da cui ottengo il rateo di nodi informati:

$$f(t) = \mathbb{E}[I(t)] \cdot n^{-1}$$

---



`./img/bitcoinpropagation_5.PNG`

Figura 2.8: Fork osservati tra i blocchi 180000 e 190000 durante il collegamento alla rete.

---

Notare come la  $f(t)$  sia equivalente alla funzione di distribuzione cumulativa (**CDF**) della frequenza alla quale i peer vengono informati. Possiamo quindi utilizzare la funzione di densità di probabilità (**PDF**) della frequenza con cui i peer vengono informati rappresentata in 2.6 come una approssimazione durante le rilevazioni. Solo i nodi non informati possono produrre blocchi in conflitto, per cui combinando la probabilità di trovare un blocco con il rateo di nodi non informati otteniamo la probabilità di un fork. Definiamo  $F$  come la variabile casuale discreta che conta il numero di blocchi in conflitto trovati mentre un altro blocco viene propagato. Allora la probabilità di un fork risulta:

$$\Pr[F \geq 1] = 1 - (1 - P_b)^{\int_0^\infty (1-f(t)) dt}$$

In questo ultimo passaggio si è assunta la semplificazione per la quale la probabilità di un nodo di trovare un blocco è distribuita uniformemente in modo casuale tra tutti i nodi.

Per cui, sapendo la probabilità dell'intera rete di trovare un blocco  $P_b$  e la distribuzione di come i nodi apprendono dell'esistenza del blocco  $I_j$ , si può derivare la probabilità di un fork. Questi due valori dipendono dal potere computazionale della rete nonché dalla sua topologia e dimensione.

### 2.9.2 Misurazioni

Per confermare il corretto funzionamento del modello proposto è necessario confrontare la probabilità ottenuta con i dati rilevati.

Bisogna innanzitutto notare come i nodi non sincronizzino i loro orologi interni, bensì si regolano sui loro vicini, esiste una differenza non trascurabile tra i timestamp. Ad esempio il blocco ad altezza 209873 ha un timestamp pari a 22:10:13 mentre il blocco ad altezza 209874 ha un timestamp di 22:08:44. Dato che il secondo include l'hash del primo, i blocchi sono stati trovati nell'ordine corretto. Da questo si deduce che il conflitto nei timestamp è derivato dalla mancata sincronizzazione degli orologi dei nodi.

In questa analisi si potrebbe tenere conto dell'orario in cui il nodo spia rileva l'annuncio del blocco e l'orario in cui il blocco è stato trovato. Anche se questo metodo non subisce la differenza di orario tra i nodi, potrebbe esistere un lieve ritardo tra il calcolo del blocco e la rilevazione del relativo annuncio, e per la misurazione abbiamo a disposizione solo il timestamp del blocco. Dato che il calcolo della proof-of-work è un processo di Poisson, la differenza temporale segue una distribuzione esponenziale. La combinazione della differenza temporale degli orologi e il tempo intercorso tra i ritrovamenti dei blocchi causa uno spostamento a destra del massimo. Possiamo correggere la situazione spostando a sinistra fin quando il massimo non risulta in  $t = 0$ . L'orario di annuncio rilevato durante la misura non è influenzato dalla differenza temporale e produce l'istogramma corretto.

$$g(t) = \lambda e^{-\lambda \cdot x}$$

---

Estraendo i timestamp dai blocchi tra altezza 180000 e 190000 si ottiene la distribuzione illustrata nel grafico 2.9. Interpolando la distribuzione ottenuta con la distribuzione esponenziale si ottiene  $\lambda = 0.001578$  da cui risulta un tempo atteso tra due blocchi pari a  $1/\lambda = 633.68$  secondi. Interpolando la densità di probabilità del tempo tra i primi annunci e le misurazioni si ottiene  $\lambda = 0.001576$  che si traduce in un tempo atteso tra due blocchi  $1/\lambda = 634.17$  secondi. Le due approssimazioni sono coerenti ma sono entrambe leggermente sopra il valore obiettivo di 600 secondi. La differenza è probabilmente dovuta ad un decremento del potere computazione della rete.

Per quando riguarda la propagazione dei nodi nella rete, a causa della normalizzazione il diagramma 2.6 rappresenta anche la funzione di densità di probabilità (**PDF**) delle variabili casuali  $t_{b,j}$  per tutti i blocchi  $b$  dell'intervallo di misurazione. Per cui la frequenza dei nodi informati  $f(t)$  è l'area che sottostà all'istogramma 2.6 fino al tempo  $t$ .

Combinando la probabilità di trovare un blocco e la funzione della frequenza dei nodi informati si ottiene la seguente probabilità per un fork:

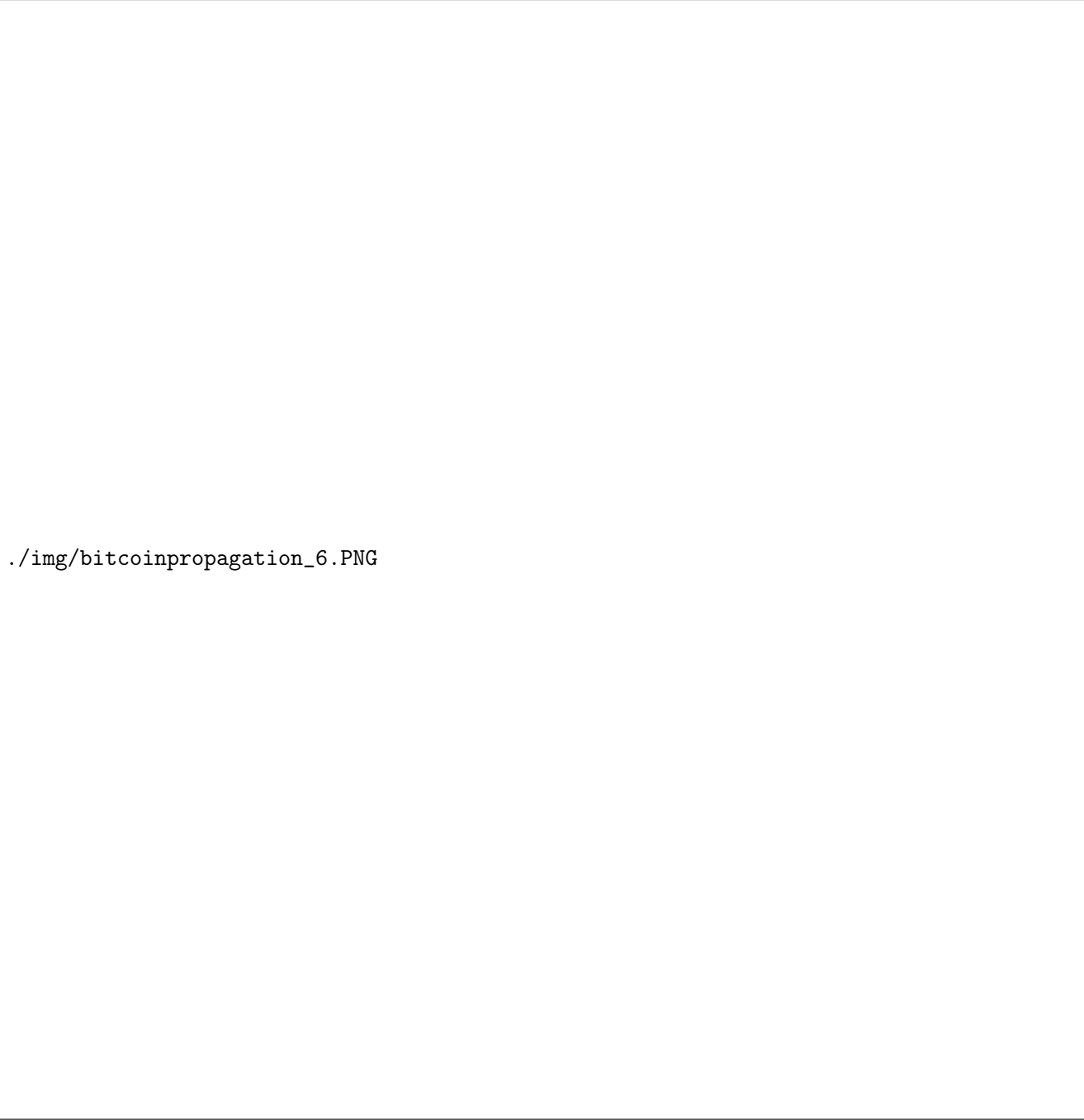
$$\begin{aligned}\Pr[F \geq 1] &= 1 - (1 - P_b)^{\int_0^\infty (1-f(t)) dt} \\ &= 1 - \left(1 - \frac{1}{633.68}\right)^{11.37} \\ &\approx 1.78\end{aligned}\tag{2.1}$$

Comparando questo risultato con quello osservato di 1.69% si nota di aver sovrastimato il valore osservato solo del 5%. Il risultato leggermente superiore può essere spiegato dall'assunto fatto che la potenza di calcolo sia uniformemente distribuita tra tutti i nodi nella rete. Ciononostante, il buon risultato ottenuto dimostra come il modello sia una efficace rappresentazione della realtà.

Dato che il numero di transazioni e le dimensioni della rete molto probabilmente cresceranno mano a mano che aumenta l'adozione di Bitcoin, la frequenza dei fork è destinata a salire. Una rete più grande, con una topologia casuale e il numero di connessioni limitate per singolo nodo, aumenta il suo diametro e la distanza media tra i nodi e l'origine di un blocco. L'aumento del numero di transazioni provoca una crescita nella dimensione dei blocchi il che a sua volta aumenta il tempo necessario per la verifica e la trasmissione ad ogni passo della propagazione.

Una interpretazione alternativa del risultato proposto in è che ogni volta che un blocco viene trovato, l'equivalente di 11.37 secondi di potere computazionale dell'intera rete viene sprecato. Infatti il lavoro impiegato per trovare il primo blocco di una blockchain alternativa (che potrebbe essere scartata) non contribuisce alla sicurezza della rete e costituisce un eventuale punto a favore di un attaccante che cerca di implementare una sua blockchain alternativa. Come detto in precedenza da Nakamoto, un attaccante capace di controllare più del 50% del potere computazione delle rete è in grado di trovare proof-of-work più velocemente di tutto il resto della rete. L'attaccante sarebbe perciò in grado di rimpiazzare l'intera storia delle transazione a partire da un qualsiasi blocco. Pur sicuramente sufficiente, questa

---



`./img/bitcoinpropagation_6.PNG`

Figura 2.9: Tempo di distribuzione spostato a sinistra per i blocchi trovati tra le altezze 180000 e 190000.



---

condizione non è minima. In realtà l'efficienza della rete come intero, incluso il ritardo di propagazione, non è ottimale. La potenza computazionale effettiva nella rete così come si presenta a Settembre 2013 è pari a  $1 - 11.37/633.68 = 98.20\%$ . Per cui ad un attaccante basta controllare il 49.1% della forza di calcolo della rete per poter portare un attacco e cambiare la blockchain. Al momento questo è un risultato difficile da ottenere, ma la situazione potrebbe cambiare in peggio a causa dell'aumento costante del ritardo di propagazione.



## Capitolo 3

# Privacy

Uno degli obiettivi del design di Bitcoin è l'anonimato dell'utente. Il paragone più appropriato è quello con i conti in banca della Svizzera. Ogni utente bitcoin può infatti possedere più di un portafogli (ovvero di una coppia di chiavi pubblica e privata) e per ogni portafogli più di un conto/indirizzo (che altro non è che una stringa unica generata grazie alla coppia di chiavi) e tutte le transazioni si basano unicamente sull'indirizzo. Non esiste quindi nessun modo per risalire al proprietario di un dato indirizzo bitcoin basandosi unicamente sulla struttura della rete.

Esistono ovviamente diverse tecniche sia per ridurre il livello di privacy, ad esempio tramite il furto delle chiavi o un'analisi statistica degli input e degli output delle transazioni, sia per aumentarlo, magari creando un diverso portafoglio per ogni conto che si desidera creare.

### 3.1 Analisi quantitativa della privacy

Ma come si fa a misurare il livello di privacy offerto da Bitcoin? In questa sezione si discuteranno alcune metriche arbitrarie create appositamente che trattano i vari aspetti dell'anonimato nella rete Bitcoin.

Essendo una analisi quantitativa, è necessario fornire una definizione formale per i termini fin qui usati in modo più o meno descrittivo. Sotto tale ottica verranno quindi proposte alcune definizioni formali, la prima delle quali è il concetto portante della rete:

Transazione:

$$\tau(a_S \rightarrow a_R) = \{\text{source}, B, a_R, \text{SIG}_{\text{sk}_{a_S}}(\text{source}, B, a_R)\}$$

definisce una transazione che intercorre tra i due indirizzi  $a_S$  e  $a_R$ , in cui  $\text{SIG}_{\text{sk}_{a_S}}$  è la firma digitale creata utilizzando la chiave privata  $\text{sk}_{a_S}$  corrispondente alla chiave pubblica associata all'indirizzo  $a_S$ .  $B$  è la quantità di BTC trasferite e source è un riferimento alla più recente transazione da cui  $a_S$  ha ottenuto le  $B$  BTC.

Al fine dell'analisi, assumiamo la situazione tipica della rete Bitcoin, in cui ogni singolo utente dispone di diversi indirizzi a lui associati. Mentre questa può sembrare

---

una forzatura, in realtà non lo è affatto. Come abbiamo detto in precedenza, ogni transazione contiene un indirizzo di ritorno per il resto. Tale indirizzo viene definito *indirizzo ombra* ed è creato automaticamente senza l'intervento dell'utente. Per cui ogni utente possiede almeno due indirizzi: quello creato al momento della creazione del portafogli e l'indirizzo ombra.

### 3.1.1 Il modello antagonistico

Gli algoritmi proposti consistono nell'osservazione del log pubblico di Bitcoin (*pubLog*) durante un periodo  $\Delta t$ . Durante questo periodo,  $n_U$  utenti (dall'insieme  $U = \{u_1, \dots, u_{n_U}\}$ ) contribuiscono al *pubLog* con l'insieme di indirizzi  $A = \{a_1, \dots, a_{n_A}\}$ . Si ha poi l'insieme delle transazioni  $T = \{\tau_1(S_1 \rightarrow R_1), \dots, \tau_{n_T}(S_{n_T} \rightarrow R_{n_T})\}$  in cui  $\tau_i(S_i \rightarrow R_i)$  descrive una singola transazione con ID univoco pari a  $i$ , e con  $S_i$  e  $R_i$  l'insieme degli indirizzi di mittente e destinatario rispettivamente. Viene introdotto anche un avversario  $\mathcal{A}$  interessato ad ottenere informazioni su tutte le transazioni e gli indirizzi appartenenti ad un insieme di utenti Bitcoin. Si assume quindi che  $\mathcal{A}$  sia un nodo legittimo della rete, abbia accesso a *pubLog*, ad alcuni indirizzi di commercianti resi pubblici e altre informazioni statistiche pubblicamente disponibili. Inoltre si assume anche che gli utenti siano incoraggiati a cambiare frequentemente i loro indirizzi, spostando le loro monete da un indirizzo all'altro. Questa è una delle abitudini consigliate da Nakamoto.

### 3.1.2 Quantificazione della Privacy

Esistono (almeno) due nozioni distinte di privacy per la rete Bitcoin.

*Activity unlinkability*: Un avversario  $\mathcal{A}$  non dovrebbe essere in grado di associare due indirizzi distinti (*address unlinkability*) o transazioni (*transaction unlinkability*) ad un utente scelto dell'avversario stesso.

*Profile indistinguishability*: L'avversario non deve essere in grado di ricostruire i profili (insieme di indirizzi e transazioni) di tutti gli utenti del *pubLog*. Questa nozione di privacy è più forte della precedente, in quanto riguarda l'intera rete e non solo un utente. Inoltre i due profili (per transazioni e per indirizzi) non sono equivalenti quando si tratta di modellare il profilo di un utente, in quanto è possibile indovinare correttamente gli indirizzi di utenti coinvolti in poche transazioni ma sbagliare nel caso di pochi indirizzi coinvolti in molte transazioni.

Vengono definiti due algoritmi *AddUnl* e *ProfInd* che implementano una sorta di sfida in cui l'attaccante tenta di violare le due nozioni di privacy sopra descritte. Il risultato sarà una quantificazione delle nozioni di privacy nei termini del vantaggio che  $\mathcal{A}$  possiede per vincere queste sfide nei confronti di un avversario  $\mathcal{A}^{\mathcal{R}}$  che tenta di vincere le stesse sfide rispondendo in modo casuale. Ipotizziamo che  $\mathcal{A}$  abbia accesso completo a *pubLog* e che sia  $\mathcal{A}$  che  $\mathcal{A}^{\mathcal{R}}$  abbiano accesso ad una base di conoscenza comune  $\mathcal{K}_{\mathcal{A}}$  che contengono informazioni verificate su un sottoinsieme di indirizzi e i relativi proprietari.

---

## Address Unlinkability

Il seguente algoritmo descrive il meccanismo con cui avviene la sfida per la address unlinkability. In questo algoritmo si utilizza  $\mathcal{A}$  come un generico avversario, in quanto per i calcoli di quantificazione l'algoritmo verrà eseguito sia da  $\mathcal{A}$  che da  $\mathcal{A}^{\mathcal{R}}$ . Si comincia con  $\mathcal{A}$  che seleziona in modo causale un indirizzo presente in pubLog e lo invia ad uno sfidante  $\mathcal{C}$  (che si assume sia a conoscenza delle corrette correlazioni utenti-indirizzi). Se l'indirizzo scelto da  $\mathcal{A}$  è l'unico che appartiene all'utente,  $\mathcal{A}$  vince la sfida. Altrimenti  $\mathcal{C}$  sceglie un bit casuale  $b$ . Se  $b = 1$  allora  $\mathcal{C}$  sceglie un nuovo indirizzo tra quelli disponibili in pubLog che appartenga allo stesso utente del primo indirizzo, altrimenti il nuovo indirizzo verrà scelto tra quelli in pubLog che non appartengono al proprietario del nuovo indirizzo. L'indirizzo scelto insieme al precedente indirizzo vengono inviati ad  $\mathcal{A}$ , il quale stima (con un algoritmo di sua scelta, ininfluenza per la sfida) se i due indirizzi che ha ricevuto appartengono o meno allo stesso utente, e memorizza il suo risultato nel bit  $b'$ . Se  $\mathcal{A}$  ha indovinato, ovvero se  $b = b'$ , allora  $\mathcal{A}$  vince la sfida. L'algoritmo è visibile in 1.

---

### Algorithm 1 AddUnl

---

```
 $a_0 \leftarrow \mathcal{A}.\text{randomAddr}(\text{pubLog})$ 
 $\mathcal{A}.\text{send}(a_0, \mathcal{C})$ 
if  $\mathcal{C}.\text{isUniqueAddr}(a_0)$  then
   $\mathcal{A}.\text{win}()$ 
else
   $b \leftarrow \mathcal{C}.\text{randomBit}()$ 
  if  $b = 1$  then
     $a_1 \leftarrow \mathcal{C}.\text{sameUsrAddr}(a_0, \text{pubLog})$ 
  else
     $a_1 \leftarrow \mathcal{C}.\text{otherUsrAddr}(a_0, \text{pubLog})$ 
  end if
   $\mathcal{C}.\text{send}(\langle a_0, a_1 \rangle, \mathcal{A})$ 
   $b' \leftarrow \mathcal{A}.\text{estimateSameUser}(a_0, a_1)$ 
  if  $b = b'$  then
     $\mathcal{A}.\text{win}()$ 
  end if
end if
```

---

Stabiliamo che Bitcoin soddisfa il principio di *Address Unlinkability* se, per ogni avversario  $\mathcal{A}$  (che ha un ben preciso algoritmo per rispondere alla domanda di  $\mathcal{C}$ ) e per ogni coppia di indirizzi scelti dall'algoritmo,  $\mathcal{A}$  ha solo un piccolo vantaggio rispetto ad un avversario come  $\mathcal{A}^{\mathcal{R}}$  che risponde a caso alla domanda posta da  $\mathcal{C}$ . Formalmente diciamo che Bitcoin soddisfa la Address Unlinkability se:

$$\Pr[b' \leftarrow \mathcal{A}(\text{pubLog}, \mathcal{K}_{\mathcal{A}}) : b = b'] - \Pr[b' \leftarrow \mathcal{A}^{\mathcal{R}}(\mathcal{K}_{\mathcal{A}}) : b = b'] \leq \varepsilon$$

con  $\varepsilon$  trascurabile.

---

**Quantificazione di Address Unlinkability** Sfruttando i risultati ottenuti dalla sfida, è possibile stimare il *grado* con cui gli indirizzi Bitcoin possono essere collegati ad uno stesso utente. Verranno definite alcune strutture matematiche necessarie per i calcoli.

$$E_{\text{link}}[i, j] = \{p_{i,j}\}_{i,j \in [1, n_A]}$$

Rappresenta una matrice  $n_A \times n_A$  in cui ogni argomento esprime la probabilità  $p_{i,j}$  stimata da  $\mathcal{A}$  che l'indirizzo  $a_i$  appartenga allo stesso utente dell'indirizzo  $a_j$  (in simboli,  $a_i \equiv_U a_j$ ). Si definisce poi una matrice  $n_A \times n_A$  che mantenga le informazioni sulle reali connessioni tra gli indirizzi:

$$\text{GT}_{\text{link}}[i, j] = \begin{cases} 1 & \text{se } a_i \equiv_U a_j \\ 0 & \text{altrimenti} \end{cases}$$

Date queste due strutture, si definisce l'errore commesso da  $\mathcal{A}$  nella sua stima come la *distanza di Manhattan* che intercorre tra  $E_{\text{link}}[i, *]$  e le vere associazioni di  $a_i$  con tutti gli indirizzi in pubLog:

$$\begin{aligned} \text{Er}_{\mathcal{A}} &= \|E_{\text{link}}[i, *] - \text{GT}_{\text{link}}[i, *]\|_1 \\ &= \sum_x |E_{\text{link}}[i, x] - \text{GT}_{\text{link}}[i, x]| \quad \text{con } x \in [1, n_A] \end{aligned}$$

Si può dunque determinare il successo di  $\mathcal{A}$  per AddUnl come il massimo del suo errore:

$$\begin{aligned} \text{Succ}_{\mathcal{A}} &= \max_{\forall a_i \notin \mathcal{K}_{\mathcal{A}}} (\|E_{\text{link}}[i, *] - \text{GT}_{\text{link}}[i, *]\|_1) \\ &= \max_{\forall a_i \notin \mathcal{K}_{\mathcal{A}}} \left( \sum_x |E_{\text{link}}[i, x] - \text{GT}_{\text{link}}[i, x]| \right) \quad \text{con } x \in [1, n_A] \end{aligned}$$

Stesse condizioni devono essere fatte per l'avversario con criterio di decisione casuale,  $\mathcal{A}^{\mathcal{R}}$ . Mantenendo uguale  $\text{GT}_{\text{link}}[i, j]$  è necessario definire la matrice  $E^{\mathcal{R}}_{\text{link}}$  come segue:

$$E^{\mathcal{R}}_{\text{link}}[i, j] = \begin{cases} \pi_{i,j} & \text{se } \langle a_i, a_j \rangle \in \mathcal{K}_{\mathcal{A}} \\ \rho + \rho(1 - \rho)\frac{1}{2} & \text{altrimenti} \end{cases}$$

Dove  $\pi_{i,j}$  rappresenta la probabilità che gli indirizzi  $a_i$  e  $a_j$  appartengano allo stesso utente secondo  $\mathcal{K}_{\mathcal{A}}$ , mentre  $\rho$  è la frazione di indirizzi in  $\{\text{pubLog} \setminus \mathcal{K}_{\mathcal{A}}\}$  che non può essere associata ad altri indirizzi (il che capita quando un utente ha solo un indirizzo)<sup>1</sup>. Per le coppie di indirizzi non incluse in  $\mathcal{K}_{\mathcal{A}}$  tale probabilità è  $\rho + \rho(1 - \rho)\frac{1}{2}$ .

---

<sup>1</sup>l'esistenza degli indirizzi ombra non ha al momento rilevanza, ma più avanti  $\rho$  diventerà trascurabile a causa della loro presenza.

---

Le altre formule vengono mantenute uguali sostituendo  $E_{\text{link}}^{\mathcal{R}}[i, j]$  a  $E_{\text{link}}[i, j]$ . Il grado di address unlinkability risulta quindi essere il grado di successo aggiuntivo che  $\mathcal{A}$  può ottenere da pubLog in comparazione con  $\mathcal{A}^{\mathcal{R}}$ , chiamato  $Link_{\mathcal{A}}^{\text{abs}}$

$$\begin{aligned} Link_{\mathcal{A}}^{\text{abs}} &= \text{Succ}_{\mathcal{A}} - \text{Succ}_{\mathcal{A}^{\mathcal{R}}} \\ &= \frac{\text{Succ}_{\mathcal{A}} - \text{Succ}_{\mathcal{A}^{\mathcal{R}}}}{\text{Succ}_{\mathcal{A}^{\mathcal{R}}}} \quad \text{in versione normalizzata} \end{aligned}$$

### User Profile Indistinguishability

L'impossibilità di ricostruire i profili di tutti gli utenti è una proprietà molto più forte rispetto all'impossibilità di associare due indirizzi diversi ad uno stesso utente.

L'algoritmo di sfida *ProfInd* viene costruito nel modo seguente. Lo sfidante  $\mathcal{C}$  invia ad  $\mathcal{A}$  il numero  $n_U$  di utenti in  $\{\text{pubLog} \setminus \mathcal{K}_{\mathcal{A}}\}$ , che risponde con  $n_U$  insiemi di indirizzi (o transazioni) e con la matrice  $E_{\text{prof}} = \{g_i\}_{i=1}^{n_U}$  rappresentante la stima fatta da  $\mathcal{A}$  sui profili degli utenti nel sistema. Come per l'algoritmo precedente, definiamo  $\text{GT}_{\text{prof}} = \{\text{gt}_i\}_{i=1}^{n_U}$  che rappresenta le vere associazioni tra indirizzi (o transazioni) e utenti, per cui:

$$\text{GT}_{\text{prof}} = \begin{cases} \{a_{u_i}\}_{i=1}^{n_U} & \text{per profili basati su indirizzi} \\ \{\tau_{u_i}\}_{i=1}^{n_U} & \text{per profili basati su transazioni} \end{cases}$$

dove  $a_{u_i}$  e  $\tau_{u_i}$  rappresentano insiemi di indirizzi/transazioni per l'utente  $u_i$ . Ovviamente,  $\mathcal{A}$  vince la sfida se indovina correttamente il profilo, ovvero  $E_{\text{prof}} \equiv \text{GT}_{\text{prof}}$ .

---

#### Algorithm 2 ProfInd

---

```

 $\mathcal{C}.\text{send}(n_U, \mathcal{A})$ 
 $E_{\text{prof}} \leftarrow \mathcal{A}.\text{estimateProfile}()$ 
if  $E_{\text{prof}} = \text{GT}_{\text{prof}}$  then
     $\mathcal{A}.\text{win}()$ 
end if

```

---

Diciamo che un sistema soddisfa la proprietà di *Profile Indistinguishability* se non esiste nessun avversario  $\mathcal{A}$  in grado di vincere *ProfInd* con una probabilità migliore dell'avversario che risponde casualmente  $\mathcal{A}^{\mathcal{R}}$ , ovvero:

$$\begin{aligned} \forall \mathcal{A} : \Pr[E_{\text{prof}} \leftarrow \mathcal{A}(\text{pubLog}, n_U) : E_{\text{prof}} \equiv \text{GT}_{\text{prof}}] - \\ \Pr[E_{\text{prof}}^{\mathcal{R}} \leftarrow \mathcal{A}^{\mathcal{R}}(n_U) : E_{\text{prof}}^{\mathcal{R}} \equiv \text{GT}_{\text{prof}}] \leq \varepsilon \end{aligned}$$

**Quantificazione di Profile Indistinguishability** Anche in questa stima come per la precedente, la quantificazione dipende dal confronto tra la stima di  $\mathcal{A}$  e i dati reali. Definiamo quindi la similitudine tra  $E_{\text{prof}}$  e  $\text{GT}_{\text{prof}}$  come la funzione  $\text{Sim}(E_{\text{prof}}, \text{GT}_{\text{prof}})$ , i cui valori appartengono all'insieme  $[0, 1]$ . Come per la address

unlinkability, misurano la profile indistinguishability contro  $\mathcal{A}$  stimando il grado con cui  $\mathcal{A}$  è in grado di stilare un profilo utente corretto, ovvero misurando il vantaggio che  $\mathcal{A}$  ha rispetto a  $\mathcal{A}^{\mathcal{R}}$  nell'avvicinarsi a  $\text{GT}_{\text{prof}}$ :

$$\text{Prof}_{\mathcal{A}} = \text{Sim}(E_{\text{prof}}, \text{GT}_{\text{prof}}) - \text{Sim}(E_{\text{prof}}^{\mathcal{R}}, \text{GT}_{\text{prof}})$$

Per quantificare  $\text{Sim}(E_{\text{prof}}, \text{GT}_{\text{prof}})$  e  $\text{Prof}_{\mathcal{A}}$  è necessario sfruttare alcune metriche per calcolare le distanze basate sull'entropia, la Normalized Mutual Information (NMI) e la Adjusted Mutual Information (AMI). La NMI valuta la similarità di due raggruppamenti degli stessi oggetti e assume valori tanto più alti (il massimo è 1) tanto più i due raggruppamenti sono identici. La AMI, dati due raggruppamenti  $G_1$  e  $G_2$ , si avvicina allo 0 quando  $G_1$  risulta simile ad un raggruppamento casuale, mentre si avvicina a 1 quando  $G_1 = G_2$ . Quindi AMI valuta direttamente il vantaggio che ha  $\mathcal{A}$  nel vincere la sfida di *ProfInd*.

Nel caso di profili basati sugli indirizzi, NMI e AMI sono calcolate come segue:

$$\text{NMI} = \frac{\mathcal{I}(E_{\text{prof}}, \text{GT}_{\text{prof}})}{\max(H(E_{\text{prof}}), H(\text{GT}_{\text{prof}}))}$$

$$\text{AMI} = \frac{\mathcal{I}(E_{\text{prof}}, \text{GT}_{\text{prof}} - \mathcal{E})}{\max(H(E_{\text{prof}}), H(\text{GT}_{\text{prof}})) - \mathcal{E}}$$

dove:

$$\mathcal{I}(E_{\text{prof}}, \text{GT}_{\text{prof}}) = \sum_{i=1}^{n_U} \left( \sum_{j=1}^{n_U} \left( \frac{n_{(i,j)}}{n_A} \log \left( \frac{n_{(i,j)} \cdot n_A}{n_{(i,*)} n_{(*,j)}} \right) \right) \right)$$

$$H(E_{\text{prof}}) = - \sum_{i=1}^{n_U} \left( \frac{n_{(i,*)}}{n_A} \log \left( \frac{n_{(i,*)}}{n_A} \right) \right)$$

$$H(\text{GT}_{\text{prof}}) = - \sum_{j=1}^{n_U} \left( \frac{n_{(*,j)}}{n_A} \log \left( \frac{n_{(*,j)}}{n_A} \right) \right)$$

$$\mathcal{E} = \sum_{i=1}^{n_U} \left( \sum_{j=1}^{n_U} \left( \sum_{n \in \mathcal{M}} \left( \frac{n}{n_A} \log \left( \frac{n_A n}{n_{(i,*)} n_{(*,j)}} \right) \frac{n_{(i,*)}! n_{(*,j)}! (n_A - n_{(i,*)})! (n_A - n_{(*,j)})!}{n_A! (n_{(i,*)} - n)! (n_{(*,j)} - n)! (n_A - n_{(i,*)} - n_{(*,j)} - n)!} \right) \right) \right)$$

$$\mathcal{M} = [\max(n_{(i,*)} + n_{(*,j)} - n_A, 0), \min(n_{(i,*)}, n_{(*,j)})]$$

In queste formule,  $n_A$  è il numero di indirizzi Bitcoin,  $n_{(i,j)}$  è il numero di indirizzi di  $u_i$ , che vengono assegnati al gruppo  $g_j$ ,  $n_{(i,*)}$  e  $n_{(*,j)}$  sono il numero di indirizzi di  $u_i$  e  $g_i$  rispettivamente.  $\mathcal{E}$  rappresenta l'informazione mutuale attesa tra il raggruppamento  $\text{GT}_{\text{prof}}$  e il raggruppamento casuale di indirizzi  $E_{\text{prof}}^{\mathcal{R}}$ . Si possono ottenere risultati simili per calcolare NMI e AMI nel caso di profili basati sulle transazioni.

È importante far presente che, sebbene NMI e AMI siano efficaci per rappresentare il successo di  $\mathcal{A}$  nella creazione di un profilo per tutti gli utenti della rete, non sono in grado di misurare il successo dell'avversario nel creare un profilo per un utente specifico.



---

## 3.2 Valutazione della privacy in Bitcoin

Vediamo ora come il nostro avversario, dato pubLog, può raccogliere informazioni sugli utenti Bitcoin sfruttando alcune proprietà delle attuali implementazioni del client Bitcoin. Grazie ad un simulatore vedremo poi quanto il modello precedentemente descritto risulti valido data la conoscenza acquisita.

### 3.2.1 Euristiche per sfruttare il client

**Transazioni multi-input** Quando per una transazione non è possibile sfruttare l'output di una singola transazione precedente, è necessario usare come input più di una transazione. I client scelgono un insieme di BTC dal portafoglio dell'utente in modo che il loro valore totale sia quello richiesto dalla transazione ed effettuano così una transazione multi-input. Dato che le BTC sono nel portafoglio dell'utente, se tali BTC sono prese da indirizzi diversi, allora tali indirizzi appartengono tutti allo stesso utente.

**Indirizzi ombra** Come detto in precedenza, gli indirizzi ombra vengono creati dal client per raccogliere l'eventuale resto delle transazioni. Immaginando quindi una transazione con un input e due output, si può ragionevolmente stabilire quale dei due indirizzi sia quello ombra appartenente allo stesso utente che ha inviato la transazione. L'indirizzo ombra sarà infatti creato sul momento, dunque sarà l'indirizzo che non è presente nel pubLog in data precedente alla transazione. Questo è valido se si assume che gli utenti della rete siano attivi (abbiano fatto almeno una transazione) e che il client non consenta di creare una transazione con più indirizzi in output.

Per valutare la bontà delle euristiche, creiamo un parser che analizzi alcuni blocchi e raggruppi gli indirizzi in cluster di indirizzi generici (GA) basati sulle euristiche.

Lanciando il parser sui primi 140000 blocchi della blockchain (ovvero i blocchi creati fino ad Agosto 2011), il parser restituisce 1632648 indirizzi unici. Utilizzando la prima euristica, tali indirizzi possono essere classificati in 1069699 distinti GA, mentre con la seconda euristica i numeri di distinti GA scende a 693051. Dato che a Settembre 2011 esistevano circa 60000 utenti Bitcoin, i risultati ottenuti indicano che è stato raggruppato circa il 58% degli indirizzi Bitcoin con una media di 11.55 indirizzi per cluster. Ciò dimostra che raccogliendo informazioni con queste euristiche, il vantaggio di  $\mathcal{A}$  per AddUnl è considerevole.

### 3.2.2 Analisi basata sul comportamento

Ora come ora, esistono molteplici client per la rete Bitcoin, e anche quello ufficiale va spesso incontro a modifiche e miglioramenti. Quello che difficilmente cambia è il comportamento degli utenti. Esistono alcune tecniche di raggruppamento basate sul comportamento che  $\mathcal{A}$  può sfruttare, come ad esempio gli algoritmi K-Means Clustering (KMC) e Hierarchical Agglomerative Clustering (HAC). Definiamo  $U$  come l'insieme di tutti gli utenti Bitcoin, e  $(GA_1, \dots, GA_{n_{GA}})$  i raggruppamenti ottenuti da

---

$\mathcal{A}$  applicando le due euristiche precedentemente descritte a pubLog. Con questi dati, l'obiettivo di  $\mathcal{A}$  è ottenere un gruppo di clusters di indirizzi  $E_{\text{prof}} = \{g_1, \dots, g_{n_U}\}$  che approssimi il più possibile  $U$ . Dato che ogni GA è posseduta da esattamente un utente, la stima dell'assegnamento di ogni GA può essere modellata con una variabile  $z_i$  tale che  $z_i = k \iff \text{GA}_i$  appartiene a  $g_k$ . L'algoritmo HAC assume che inizialmente ogni GA rappresenti un utente separato ( $\{z_i = i\}_{i=1}^{n_{\text{GA}}}$ ) e calcola valori di similitudine per ogni coppia di cluster. I cluster con elevata similarità vengono raggruppati insieme e il processo si ripete sui nuovi raggruppamenti, fermandosi quando il numero di raggruppamenti è esattamente uguale al numero di utenti  $n_U$ . A questo punto si avvia KMC, inizializzato con l'output di HAC, il quale assume che ogni utente sia rappresentato dal punto centrale di ogni raggruppamento. L'algoritmo tenta di minimizzare la distanza tra i GA e i cluster a cui essi sono stati assegnati, ricalcolando ad ogni round sia il centro dei cluster sia la distanza GA-cluster.

Nella loro implementazione, gli autori di [4] hanno rappresentato ogni transazione all'interno delle GA con tre punti:

- L'orario in cui la transazione ha avuto luogo.
- Gli indici dei diversi GA che appaiono all'interno della transazione come mittenti o destinatari.
- La quantità di BTC spese nella transazione.

$\tau_x$  identifica l'insieme delle transazioni di  $\text{GA}_x$ , e il grado di similitudine tra due cluster  $\text{GA}_i$  e  $\text{GA}_j$  è rappresentato dal coseno di similitudine <sup>2</sup> tra le liste  $\tau_i$  e  $\tau_j$ , ovvero:

$$\text{Sim}^{\text{hac}}(\text{GA}_i, \text{GA}_j) = \frac{\sum_{\forall \tau \in \tau_i \cap \tau_j} (f_{(\tau,i)} \cdot f_{(\tau,j)})}{\|\tau_i\|_2 \cdot \|\tau_j\|_2}$$

- an analysis of anonymity in bitcoin sistem

---

<sup>2</sup>Tecnica euristica che misura la differenza tra due vettori, spesso usata per il confronto dei testi

# Capitolo 4

## Sicurezza

### 4.1 Reti P2P in genere

Una rete di computer, di qualsiasi tipo essa sia, richiede necessariamente un certo livello di sicurezza. Questo è ancora più vero per reti pubbliche in cui chiunque può inserirsi, come le reti P2P che si appoggiano a Internet. Tali reti sono un bersaglio particolarmente ghiotto per gli attaccanti proprio a causa della loro maggiore forza: il grande numero di utenti, che equivale ad un grande numero di possibili bersagli. Ogni rete P2P deve quindi confrontarsi con la certezza che alcuni dei suoi nodi siano malevoli. Data la grande varietà di reti P2P, è necessario specificare cosa intendiamo per “reti P2P in genere”. Il modello [11] a cui si farà riferimento consiste nelle seguenti componenti di base:

- Uno spazio degli ID composto da  $b$  bits.
- Un sistema di mappatura degli ID.
- Un sistema di routing, che utilizza una chiave per inoltrare un messaggio alla sua destinazione. Questo include una serie di regole per il churn.

Basandoci sul modello di rete OSI, classifichiamo gli attacchi come di basso, medio e alto livello, a seconda della vicinanza al livello di comunicazione fisico: più il livello dell’attacco è alto, più si basa sul software invece che sull’hardware.

Visto che le reti P2P sono costruite sulla base di altre reti, i livelli ISO/OSI a cui siamo interessati sono il livello applicativo (per il P2P vero e proprio) e il livello di comunicazione (per attacchi basati su TCP e IP).

#### 4.1.1 Attacchi di Basso Livello

##### Denial of Service (DoS)

Uno degli attacchi più basilari che interessa praticamente ogni tipo di rete. Consiste nel bloccare i servizi offerti da uno specifico bersaglio. Si tratta di un attacco estremamente comune esistente in infinite varianti, ma nel caso di P2P la sua versione

---

più semplice consiste nel flood. Consiste nell'estremizzare quanto descritto nel query flood: inondare la rete di pacchetti, legittimi o creati ad-hoc, in modo da ostacolare la normale comunicazione tra i nodi. Semplice, ma estremamente efficace.

Una variante ancora più efficace, e resa tale proprio dalla natura distribuita delle reti P2P, consiste nel **Distribute Denial of Service** (DDoS). Come dice il nome, la differenza sta nel fatto che l'attaccante non è un singolo nodo bensì un insieme di nodi. Il vantaggio in questa tecnica, oltre che nell'immenso numero di pacchetti che inondano la rete, sta nell'anonimato che circonda l'attaccante. Spesso infatti i nodi che a tutti gli effetti inviano i pacchetti sono inconsapevoli di essere parte dell'attacco e manipolati remotamente da un attaccante. Questo aggiunge un ulteriore strato tra l'attaccante e i nodi legittimi che tentano di impedire l'attacco.

Gli attacchi DoS e DDoS diventano tanto più probabili e quanto è vasta la rete, non solo per il maggior numero di nodi compromettibili per un DDoS, ma anche per le politiche aziendali attuate da molte società e provider. Infatti, più la rete è diffusa, più è probabile che firewall aziendali ne limitino l'accesso ai propri utenti e che ISP nazionali ne limitino l'utilizzo. Questo costringe gli utenti a piazzarsi al di fuori di tali reti protette per poter usufruire della rete P2P, e quindi ad esporsi maggiormente agli attacchi.

**Contromisure al DDoS** Il primo problema consiste nell'identificare i DDoS. I sintomi di un DDoS sono del tutto identici a quelli di un normale elevato traffico di rete, in particolare quando i pacchetti usati sono legittimi e non forgiati ad-hoc per l'attacco. Inoltre in un DDoS non tutti i nodi sono stati creati appositamente per attaccare, spesso sono nodi legittimi che vengono utilizzati dall'attaccante per far rimbalzare i suoi pacchetti di attacco. Questi due fattori combinati rendono di fatto *impossibile* bloccare tutti gli attacchi DoS.

Esiste però una tecnica ampiamente diffusa per rendere poco pratico il DoS, o almeno per rallentarlo in maniera drastica. Si tratta del **pricing**, una tecnica che limita la velocità alla quale i nodi possono fare richieste alla rete. Se un nodo deve fare richieste ad un altro nodo (ad esempio, una query per un file), il nodo risponde con richieste di calcoli di hash, esattamente gli stessi calcoli necessari per creare un blocco in Bitcoin. Solo dopo aver risolto il calcolo ed inviata la risposta, la richiesta viene presa in considerazione, tutte le altre comunicazioni inviate nel frattempo vengono scartate, scoraggiando quindi qualsiasi richiesta invadente.

### **Attacco Man-in-the-Middle (MitM)**

Questo attacco consiste nell'inserimento dell'attaccante tra due nodi della rete, in modo che tutte le comunicazioni tra i due nodi passino attraverso l'attaccante. L'attacco è irrilevabile fin tanto che l'attaccante rimane passivo. Una volta ottenute tutte le informazioni che desidera, l'attaccante può diventare attivo modificando i messaggi che vengono scambiati oppure forgiandone di propri spacciandosi per uno o entrambi dei nodi. Inoltre, dato che l'attaccante può influenzare la visione che i

---

due nodi hanno del resto della rete, può creare false identità per simulare messaggi legittimi.

Se questo attacco viene effettuato al livello di rete, l'attaccante è in grado di vedere tutto ciò che passa tra i due nodi e, essendo questo livello inferiore a quello P2P, l'attaccante non ha nessun problema nel creare qualsiasi tipo di pacchetto P2P egli desideri.

Come il DoS ma in misura estremamente maggiore, le reti P2P sono un bersaglio goloso per questo genere di attacchi. Questo perché è difficile inserirsi tra due nodi in una rete normale, ma in una rete P2P è estremamente banale: tali reti infatti non hanno nessun controllo su come sono localizzati i nodi e sono quindi *estremamente* vulnerabili al MitM. Dato che l'attaccante può piazzarsi dove vuole all'interno della rete, gli attacchi risultano essere molto specifici e deterministici, fino anche ad impedire ad un determinato nodo di accedere ad un altro nodo.

### Contrastare il Man-in-the-Middle

Il metodo principale per difendersi dal MitM è rendere tale attacco il più infruttuoso possibile. In una rete senza nodi privilegiati (ad esempio un server centrale, un supernodo o un'autorità centrale di autenticazione), il MitM si limita a compromettere la sicurezza tra due soli nodi nella rete, senza minacciare per nulla il resto dei nodi <sup>1</sup>.

Molte reti (non bitcoin) sfruttano nodi privilegiati per affrontare altre minacce o come parte integrante della loro struttura, e anche nelle reti maggiormente distribuite è possibile effettuare un attacco MitM su larga scala (*Eclipse*).

Il metodo più diffuso per evitare la fuoriuscita di informazioni nella comunicazione tra due nodi è la cifratura a chiave pubblica. Con tale sistema si garantisce l'origine del messaggio, il fatto che non è stato alterato in alcun modo e, volendo, fornisce anche un metodo per evitare che una terza parte non autorizzata possa leggerne il contenuto. L'implementazione di questo semplice meccanismo rende praticamente inutile qualsiasi tentativo di MitM tra due nodi.

#### 4.1.2 Attacchi di Medio Livello

##### Worms

Un worm è un programma auto-replicante simile a un virus che, a differenza di questo, è indipendente da altri programmi presenti sul sistema. La minaccia rappresentata da un worm è decisamente significativa per una rete P2P, in quanto si diffondono a tutti i nodi della rete tramite vulnerabilità presenti ad un livello più basso rispetto a quello della rete stessa. Pur non essendo legati direttamente al P2P, i worm vengono diffusi in modo capillare da esso principalmente a causa di come il P2P viene implementato. In molte architetture infatti i nodi per comunicare tra loro

---

<sup>1</sup>almeno fino a quando la perdita di tre nodi (due vittime e un attaccante) risulta tollerabile per la rete, il che è vero per tutte quelle reti ampiamente diffuse.

---

devono avere installato lo stesso software. Ciò significa che quando questo software ha una certa vulnerabilità (ad esempio, un buffer overflow), tutti i nodi della rete sono vulnerabili. Quindi mentre un worm “normale” deve effettuare una scansione dell’intera rete per trovare degli host vulnerabili, un worm P2P deve solo guardare le tabelle di routing e infettare tutti i nodi vicini, diffondendosi esponenzialmente: in confronto ai worm normali, i worm P2P infettano l’intera rete in modo praticamente istantaneo.

Oltre alla grande velocità di diffusione, il fatto che molte reti P2P siano concepite per il file-sharing e abbiano quindi una grande abbondanza di banda, consente al worm P2P di avere dimensioni maggiori e di essere quindi capace di azioni ed attacchi molto più complicati rispetto ad un worm normale, a volte grande solo come un pacchetto TCP/IP. Molti nodi sono inoltre spesso computer personali di utenti normali che utilizzano quotidianamente Internet. Worm sufficientemente complessi sono in grado di monitorare l’attività dell’utente anche al di fuori della rete P2P e di accedere a dati quali numero di carta di credito, password di account, ecc., rendendo le reti P2P un bersaglio di estremo valore.

Infine, i worm possono usare la rete come uno strumento. Si è parlato prima parlando del DDoS di come un nodo qualsiasi possa diventare un ignaro vettore di attacco. I worm sono il modo in cui questo viene reso possibile: il worm contiene tutto il codice necessario ad effettuare l’attacco, oltre al codice per replicare se stesso negli altri nodi.

**Contrastare i Worm** Il modo principale è mantenere le applicazioni sicure: senza una vulnerabilità comune un worm non può diffondersi in modo efficace. La sicurezza di un software dipende dal modo in cui esso è stato programmato, ad esempio per ridurre il rischio di buffer overflow è possibile usare linguaggi fortemente tipati.

Per ridurre invece l’efficacia di un worm si può decentralizzare il più possibile la rete evitando di implementare nodi privilegiati e/o utilizzare sistemi operativi **hardened**. OpenBSD dalla versione 3.8 per esempio utilizza indirizzi pseudocausali in fase di allocazione della memoria, rendendo quindi difficile sfruttare vulnerabilità presenti nei vari applicativi.

Ma il modo più pratico per difendersi dai worm è mantenere aperta la rete. Ciò significa basarsi su standard diffusi ed aperti per i propri applicativi. Rilasciare i protocolli al pubblico e distribuire il codice dei propri software incoraggia altri sviluppatori ad una analisi critica, il che porta alla più tempestiva scoperta di vulnerabilità ed alla rapida creazione di patch, bug-fix e fork più robusti del software in questione. Inoltre, con opportune licenze, ogni sviluppatore può creare il proprio client per interfacciarsi con la rete costringendo un attaccante a creare un worm specifico per ogni versione di ogni client, e a mantenere tale worm aggiornato mano a mano che nuove vulnerabilità vengono scoperte e risolte o nuovi client implementati. Con una grande varietà di client a disposizione, non tutti i nodi saranno vulnerabili allo stesso identico difetto presente in un altro client.

---

### 4.1.3 Attacchi al livello P2P

#### Comportamento scorretto

Non si tratta di danneggiare una rete intera o un singolo nodo, bensì di trarre il massimo profitto offrendo minima collaborazione. Questo comportamento viene definito genericamente **attacco razionale**. La terminologia si basa sull'assunzione che dietro ogni nodo ci sia un utente razionale che per istinto tenta di trarre il massimo beneficio con il minimo sforzo. Ad esempio nell'ambito della rete BitTorrent un utente scarica un file eliminandolo dalla condivisione non appena completo, oppure riduce ai minimi termini la banda in upload massimizzando quella in download: tale comportamento viene definito significativamente **leeching**, letteralmente *sanguisuga*.

Ci sono molte ragioni per cui un nodo debba comportarsi in questo modo:

- Per preservare la banda in upload, spesso molto limitata dagli ISP.
- Per motivi legali, in special modo dove la condivisione di contenuti protetti dal copyright può risultare in un'azione legale nei confronti del nodo che condivide. Data la natura aperta delle reti, spesso è molto facile risalire all'origine di un contenuto.
- Per istinto: molte persone, se viene lasciata loro possibilità di scelta, tendono a non cooperare per il solo beneficio di aiutare la comunità a cui appartengono, non importa quanto minimo sia il costo che comporta loro.

Come descritto, due sono i metodi in cui ci può implementare questo “attacco”: riducendo le risorse a disposizione della rete oppure riducendo il contenuto condiviso.

**Contrastare i comportamenti scorretti** Ogni rete deve implementare il suo diverso meccanismo per favorire la collaborazione tra i peer: in Bitcoin ci sono le transaction fee, Napster utilizzava un meccanismo di reputazione che favoriva gli utenti che condividevano di più, Samsara (una rete di backup distribuito) consente ad un utente di utilizzare uno spazio su un altro nodo solo pari a quello che l'utente mette a disposizione per gli altri nodi.

Un esempio encomiabile è quello di BitTorrent: il protocollo è disinteressato al numero di file condivisi da un utente o dal contenuto di per se, ma si interessa solamente di quante risorse vengono condivise. Secondo il protocollo BitTorrent i file da condividere vengono suddivisi in parti (**chunks**) di lunghezza variabile che vengono barattati tra i nodi: più un nodo è disposto a dare, più si vedrà restituire. In pratica, più è alta la velocità di upload, più gli altri nodi assegneranno banda a quel nodo, aumentandone la velocità di download.

#### Attacco Sibilla

Si ha questo attacco quando una singola entità malevola rappresenta un grande numero (spesso estremamente elevato) di utenti in una rete P2P con l'obiettivo di

---

assumere il controllo di un segmento della rete. L'attacco si implementa con l'attaccante che tenta di creare un grande numero dei nodi a lui vicini. L'attacco diventa più efficace se l'attaccante è in grado di decidere dove posizionarsi nella rete, in quanto potrebbe aver bisogno di meno nodi per poter causare gravi danni alla rete. Con molti nodi a propria disposizione è possibile controllare tutti i messaggi che passano per il segmento formato dai nodi in questione. Questo attacco è inoltre un **attacco gateway**, il che sta ad indicare una categoria di attacchi solitamente usati come passo preliminare per attacchi su vasta scala di altro tipo, come per esempio l'attacco Eclipse descritto più avanti.

**Contromisure all'attacco Sibilla** La natura aperta delle reti P2P gioca a favore di questo tipo di attacco: senza un'autorità centrale è impossibile fermare completamente un attacco Sibilla. Il meglio che si può fare è renderlo impraticabile.

Per rallentare un attacco si può usare lo stesso metodo usato con gli attacchi DoS: il pricing. Il grande numero di calcoli richiesti per unire molti nodi alla rete può richiedere all'attaccante più tempo di quanto sia disposto ad impiegarne. Se inoltre la rete implementa una sorta di tempo massimo durante il quale un nodo mantiene un certo identificativo, tutti gli attacchi hanno un tempo massimo entro il quale devono essere portati a termine, dopo di che i nodi creati dovranno ricollegarsi alla rete e quindi si dovranno ripetere tutte le pratiche del pricing.

## Attacco Eclipse

L'obiettivo di questo attacco è di separare la rete in due o più partizioni. Quando l'attacco ha successo, tutte le comunicazioni tra le due partizioni devono passare attraverso un singolo nodo malevolo. In pratica questo risulta in un attacco Man-in-the-Middle su vasta scala eseguito a livello applicativo e non a livello di rete, con tutte le potenzialità del MitM normale. Per eseguire l'attacco bisogna piazzare i propri nodi in punti di routing strategici che esistono tra le due partizioni che si vogliono creare. Un attacco Eclipse di successo può distruggere qualsiasi rete P2P, soprattutto quelle che non si curano molto di mantenere tabelle di routing efficienti, perché i nodi finti possono essere posizionati in modo da riempire i vuoti nelle tabelle di routing di ogni altro nodo.

**Contromisure ad Eclipse** Data la similarità di Eclipse con il MitM, le contromisure sono anche simili: cifratura a chiave pubblica. Tuttavia, sebbene MitM non sia una minaccia per la rete, le dimensioni di Eclipse lo rendono estremamente pericoloso anche in caso di cifratura a chiave pubblica. Se i messaggi illeciti indirizzati ai nodi legittimi vengono bloccati, le due partizioni create da Eclipse rimangono di fatto isolate. Con abbastanza nodi piazzati in punti strategici della rete, è possibile creare quante divisioni si desidera, riducendo quindi le dimensioni della rete.

Come per l'attacco Sibilla, è importante impedire ad un attaccante di decidere dove piazzare i suoi nodi. Questo significa che sarà richiesto un elevato numero di



---

nodi per avere una speranza di ottenere sufficiente controllo per poter creare una partizione. Per cui è importante notare che con un attacco Sibilla sufficiente grande è *sempre* possibile eseguire un attacco Eclipse.

#### 4.1.4 Contromisure generali

Si è quindi visto come, oltre alle caratteristiche di tolleranza ai guasti e grande scalabilità che ne rappresentano la base del design, le reti P2P devono anche essere progettate per difendersi dagli attacchi. Solo in questo modo una rete potrà consentire ad ogni nodo di collegarsi e realizzare in pieno il concetto di collaborazione che sta alla base.

In particolare i progettisti di reti P2P dovrebbero implementare le seguenti caratteristiche:

- L'impossibilità per un nodo di decidere in che punto della rete piazzarsi.
- Un limite per la velocità di churn per i nuovi nodi.
- Limitare la velocità di scambio di messaggi tra i nodi, ad esempio con un pruning.
- Utilizzare la crittografia a chiave pubblica per garantire solo messaggi legittimi tra i nodi.
- Usare ed implementare solo standard aperti, per diversificare il software a disposizione degli utenti ed irrobustire quelli esistenti.

Se queste caratteristiche vengono implementate, allora tutti gli attacchi fin qui descritti perdono gran parte della loro efficacia, ripagando con la sicurezza il costo che richiede la loro realizzazione.

## 4.2 La rete Bitcoin

- spiegazione di satoshi
- estensione di propagation
- soluzioni di propagation
- ? how to make bitcoin a better currency

## 4.3 I Portafogli

- contromisure al furto di portafogli
- paperwallets

## 4.4 Stock Exchange

- beware the middleman



# Bibliografia

- [1] *Bitcoin Wiki*. <https://en.bitcoin.it/wiki/>.
- [2] *BlockChain.info*. <https://www.blockchain.info>.
- [3] *Mt. GOX*. <https://mtgox.org>.
- [4] Androulaki, Elli, Ghassam Karame, Marc Roeschlin, Tobias Scherer e Srdjan Capkun: *Evaluating User Privacy in Bitcoin*. Cryptology ePrint Archive, Report 2012/596, 2012. <http://eprint.iacr.org/2012/596.pdf>.
- [5] Babaioff, Moshe, Shahar Dobzinski, Sigal Oren e Aviv Zohar: *On bitcoin and red balloons*. Nel *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, pagine 56–73, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1415-2. <http://doi.acm.org/10.1145/2229012.2229022>, Disponibile in download gratuito all'indirizzo <http://arxiv.org/pdf/1111.2626.pdf>.
- [6] Back, Adam: *Hashcash - a denial of service counter-measure*, 2002. <http://www.hashcash.org/papers/hashcash.pdf>.
- [7] Bamert, Tobias, Christian Decker, Lennart Elsen, Roger Wattenhofer e Samuel Welten: *Have a Snack, Pay with Bitcoins*. Nel *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Trento, Italy, September 2013. [http://www.tik.ee.ethz.ch/file/848064fa2e80f88a57aef43d7d5956c6/P2P2013\\_093.pdf](http://www.tik.ee.ethz.ch/file/848064fa2e80f88a57aef43d7d5956c6/P2P2013_093.pdf).
- [8] Barber, Simon, Xavier Boyen, Elaine Shi e Ersin Uzun: *Bitter to Better - How to Make Bitcoin a Better Currency*. Nel Keromytis, Angelos D. (curatore): *Financial Cryptography and Data Security*, volume 7397 della serie *Lecture Notes in Computer Science*, pagine 399–414. Springer Berlin Heidelberg, 2012, ISBN 978-3-642-32945-6. [http://dx.doi.org/10.1007/978-3-642-32946-3\\_29](http://dx.doi.org/10.1007/978-3-642-32946-3_29), Disponibile in download gratuito all'indirizzo <http://www.cs.stanford.edu/~xb/fc12/>.
- [9] Clark, Jeremy e Aleksander Essex: *CommitCoin: Carbon Dating Commitments with Bitcoin*. Nel Keromytis, Angelos D. (curatore): *Finan-*

- 
- cial Cryptography and Data Security*, volume 7397 della serie *Lecture Notes in Computer Science*, pagine 390–398. Springer Berlin Heidelberg, 2012, ISBN 978-3-642-32945-6. [http://dx.doi.org/10.1007/978-3-642-32946-3\\_28](http://dx.doi.org/10.1007/978-3-642-32946-3_28), Disponibile in download gratuito all'indirizzo <http://eprint.iacr.org/2011/677.pdf>.
- [10] Decker, Christian e Roger Wattenhofer: *Information Propagation in the Bitcoin Network*. Nel *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Trento, Italy, September 2013. [http://www.tik.ee.ethz.ch/file/49318d3f56c1d525aabf7fda78b23fc0/P2P2013\\_041.pdf](http://www.tik.ee.ethz.ch/file/49318d3f56c1d525aabf7fda78b23fc0/P2P2013_041.pdf).
- [11] Engle, Marling e Javed I. Khan: *Vulnerabilities of P2P Systems and a Critical Look at their Solution*. Rapporto Tecnico, Kent State University, Networking and Media Communications Research Laboratories, Department of Computer Science, 233 MSB, Kent, OH 44242, November 2006. <http://medianet.kent.edu/technicalreports.html>.
- [12] Kshemkalyani, Ajay D. e Mukesh Singhal: *Peer-to-peer computing and overlay graphs*, capitolo 18. Cambridge University Press, The Edinburgh Building, Cambridge CB2 8RU, UK, first edizione, 2008.
- [13] Kurose, James F. e Keith W. Ross: *Applicazioni peer-to-peer*, capitolo 2, pagine 131–144. Pearson Education, Addison-Wesley, quarta edizione, 2006.
- [14] Merkle, Ralph C.: *Protocols for Public Key Cryptosystems*. Nel *IEEE Symposium on Security and privacy*, volume 1109, pagine 122–134, April 1980. <http://www.cs.washington.edu/research/projects/poirot3/Oakland/sp/PAPERS/00044729.PDF>.
- [15] Moore, Tyler e Nicola Christian: *Beware the Middleman: Empirical Analysis of Bitcoin-Exchange Risk*. Nel *Proceedings of Financial Cryptography 2013*, April 2013. <http://www.truststc.org/pubs/907.html>, Disponibile in download gratuito all'indirizzo <http://fc13.ifca.ai/proc/1-2.pdf>.
- [16] Nakamoto, Satoshi: *Bitcoin: A Peer-to-Peer Electronic Cash System*. [satoshin@gmx.com](mailto:satoshin@gmx.com). [www.bitcoin.org](http://www.bitcoin.org).
- [17] Reid, Fergal e Martin Harrigan: *An Analysis of Anonymity in the Bitcoin System*. Nel Altshuler, Yaniv, Yuval Elovici, Armin B. Cremers, Nadav Aharoni e Alex Pentland (curatori): *Security and Privacy in Social Networks*, pagine 197–223. Springer New York, 2013, ISBN 978-1-4614-4138-0. [http://dx.doi.org/10.1007/978-1-4614-4139-7\\_10](http://dx.doi.org/10.1007/978-1-4614-4139-7_10), Disponibile in download gratuito all'indirizzo <http://arxiv.org/pdf/1107.4524.pdf>.
- [18] Ron, Dorit e Adi Shamir: *Quantitative Analysis of the Full Bitcoin Transaction Graph*. Cryptology ePrint Archive, Report 2012/584, 2012. <http://eprint.iacr.org/2012/584.pdf>.

- 
- [19] Rosenfeld, Meni: *Analysis of Bitcoin Pooled Mining Reward Systems*. CoRR, abs/1112.4980, 2011. Disponibile in download gratuito all'indirizzo <http://arxiv.org/pdf/1112.4980v1.pdf>.
- [20] Schoeder, Detlef, Kai Fischbach e Christian Schmitt: *Core Concepts in Peer-to-Peer Networking*, capitolo 1. Idead Group Inc., 2005.