La Simplified Payment Verification in Bitcoin

Giancarlo Giuffra Moncayo

1 febbraio 2018

Indice

1 SPV (Simplified Payment Verification)

Ogni nodo della rete di Bitcoin riceve transazioni, le valida e le propaga. Lo stesso meccanismo viene applicato ai blocchi, si ricevono, si validano, ovviamente con regole diverse, e si propagano. Questo processo di validazione e propagazione serve a ogni nodo per ricostruire l'intera catena di blocchi. Ogni nodo ha la storia completa delle transazioni salvata nel proprio dispositivo. Questi nodi quindi devono soddisfare dei requisiti minimi per poter immagazzinare tale quantità di dati[1]. Inoltre ogni nuovo nodo che si connette alla rete deve ricevere e validare tutta la catena, dal blocco genesi fino a quell'istante, prima di poter ricevere nuove transazioni. In effetti deve conoscere tutta la storia per poter essere in grado di validarle. Quindi inizialmente il nodo deve sincronizzarsi con la rete, processo che attualmente richiede un lasso di tempo considerevole. Il nodo dovrà poi mantenersi connesso alla rete oppure sincronizzarsi con una certa frequenza se vuole poter continuare a validare transazioni, e.g. transazioni che inviano dei Bitcoin a chiavi pubbliche che controlla. Il comportamento che abbiamo descritto è quello di un cosiddetto full node, cioè un nodo che immagazzina l'intera

[1] Si può trovare l'andamento temporale delle dimensioni della blockchain di Bitcoin qui.

catena e conosce quindi la storia di tutte le transazioni effettuate. Si può immaginare che non tutti siano propensi a dedicare le risorse necessarie per salvare l'intera catena e mantenersi sincronizzati con la rete Bitcoin. Lo stesso Satoshi descrisse una alternativa[2] che richiede meno risorse e meno tempo per la sincronizzazione iniziale, ma che comunque permette al nodo di verificare che certe transazioni appartengano alla catena, e.g. quelle che inviano dei Bitcoin alle sue chiavi pubbliche. Questa alternativa si chiama SPV dall'acronimo inglese Simplified Payment Verification, e i nodi che utilizzano questo meccanismo per interagire con la rete vengono chiamati light nodes oppure thin clients.

I nodi che utilizzano il meccanismo SPV ricevono dai nodi ai quali sono connessi, detti peers, due tipi di informazioni:

1. informazioni riguardanti i blocchi, specificamente ricevono solo l'header del blocco,

2. informazioni che permettono di verificare l'appartenenza di una transazione a uno dei blocchi i cui header sono stati ricevuti.

L'header del blocco non contiene ovviamente tutte le transazioni ma solo poche informazioni[3] riguardo il blocco. Tra i dati lì contenuti ci sono tre che ci interessano in particolare:

1. l'hash del blocco che lo precede,

2. il target,

3. la radice del Merkle Tree costruito a partire dalle transazioni del blocco.

L'hash del blocco precedente insieme al valore del target servono al light node per poter ricostruire la catena di header di blocchi attiva, cioè quella che ha la difficoltà complessiva più alta. Invece la radice del Merkle Tree viene usata, insieme ad altre informazioni, per verificare se una transazione appartiene o meno al blocco.

Prima di procedere penso che sia utile capire come viene costruito un Merkle Tree a partire dalle transazioni. Iniziamo ricordando che un Merkle Tree è un albero binario pieno dove ciascun nodo contiene un hash. Nel nostro caso le foglie di questo albero sono gli id delle transazioni che fanno

[2]Si veda la sezione 8 del paper originale

[3]L'header di un blocco occupa solo 80B di spazio. Si veda la seguente pagina della bitcoin wiki per la sua struttura.

parte del blocco. Si ricorda che l'id di una transazione viene calcolato come l'hash[4] dell'intera transazione. Il resto dell'albero viene costruito seguendo una semplice procedura. Si prende una coppia di transazioni e il nodo genitore viene costruito come l'hash della concatenazione degli hash delle figlie. Questa procedura si applica a ciascun livello dell'albero per costruire quello precedente fino ad ottenere un unico hash. Quest'ultimo hash viene chiamato radice del Merkle Tree o anche Merkle Root. Siccome questa procedura richiede un numero pari di nodi a ciascun livello, se il numero di nodi è dispari, si duplica l'ultimo nodo. In Figura 1 si mostra un Merkle Tree costruito a partire da un gruppo di 3 transazioni.

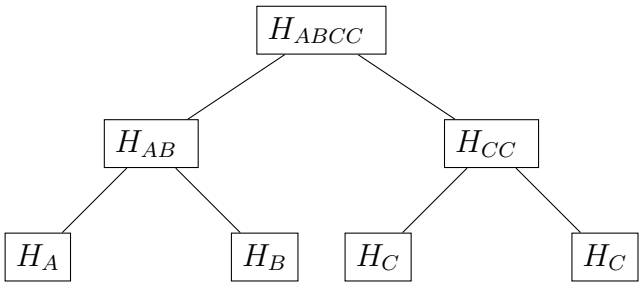

Figura 1: Esempio di Merkle Tree costruito a partire da 3 transazioni identificate con gli id H_A, H_B e H_C. La Merkle Root è H_{ABCC}.

Il secondo tipo di informazioni ricevute da un light node è una prova crittografica che dimostra che una transazione appartiene a un blocco. Questa prova crittografica si basa sulla struttura del Merkle Tree associato al blocco e viene chiamata Merkle Branch. Una Merkle Branch non è, a discapito del nome, un ramo dell'albero, ma bensì una lista di hash appartenenti al Merkle Tree che permette al light node di ricostruire il ramo al quale la transazione appartiene. Ricostruendo questo ramo si ottiene alla fine la radice dell'albero che può essere confrontata con la radice ricevuta in precedenza come parte dell'header e verificare così se la transazione appartiene al blocco. Vediamo un esempio di prova crittografica di questo tipo. Consideriamo l'albero in Figura 2. Con lo sfondo nero è stato evidenziato l'id della transazione della quale si vuole provare l'appartenenza al blocco, cioè H_K. Gli altri hash che fanno parte della prova crittografica sono stati evidenziati con uno sfondo blu. La validazione della prova inizia prendendo l'id della transazione che ci interessa, H_K, e il seguente hash della lista, H_L. Si procede alla concatenazione di questi valori, si applica la funzione di hash e si ottiene H_{KL}. Di seguito si prende questo valore calcolato e il successivo

[4]Per calcolare l'id di una transazione si usa l'algoritmo SHA-256 due volte.

valore della lista, H_{IJ}. Si applica la stessa procedura e si calcola H_{IJKL}. Questa iterazione continua fino ad aver utilizzato tutti gli hash della Merkle Branch ottenendo alla fine il valore $H_{ABCDEFGHIJKLMNOP}$. Se questo ultimo hash coincide con la Merkle Root inclusa nell'header del blocco la prova crittografica è valida e si è sicuri che H_K appartenga al blocco.

Si osserva che una Merkle Branch contiene molti meno hash rispetto al numero di transazioni che appartengono al blocco. In effetti, oltre all'hash della transazione di cui si vuole provare l'appartenenza, nella Merkle Branch viene incluso un solo hash per livello dell'albero. Vale quindi la seguente relazione:

$$L_{MB} = \log_2(N) + 1,$$

dove si è indicato con L_{MB} il numero di hash di una Merkle Branch che prova l'appartenenza di una transazione a un blocco che ne contiene N. Con questa osservazione, e considerando una dimensione media di 700B per transazione, è possibile calcolare in modo approssimativo lo spazio disco che occupa una Merkle Branch. Un blocco di 1MB può contenere $\frac{1\text{MB}}{700\text{B}} \sim 1429$ transazioni. Di conseguenza la Merkle Branch conterrà $\log_2(1429) + 1 \sim 12$ hash. Considerando che un hash SHA-256 occupa $\frac{256\text{bit}}{8\text{bit}} = 32\text{B}$, la Merkle Branch occupa circa $12 \times 32 = 384\text{B}$. Un light node deve quindi mantenere su disco circa $80\text{B} + 700\text{B} + 384\text{B} = 1164\text{B}$ nel caso di un blocco con una transazione che le interessa, cioè un risparmio di circa 3 ordini di grandezza rispetto al blocco di 1MB, e solo gli 80B di header per gli altri blocchi.

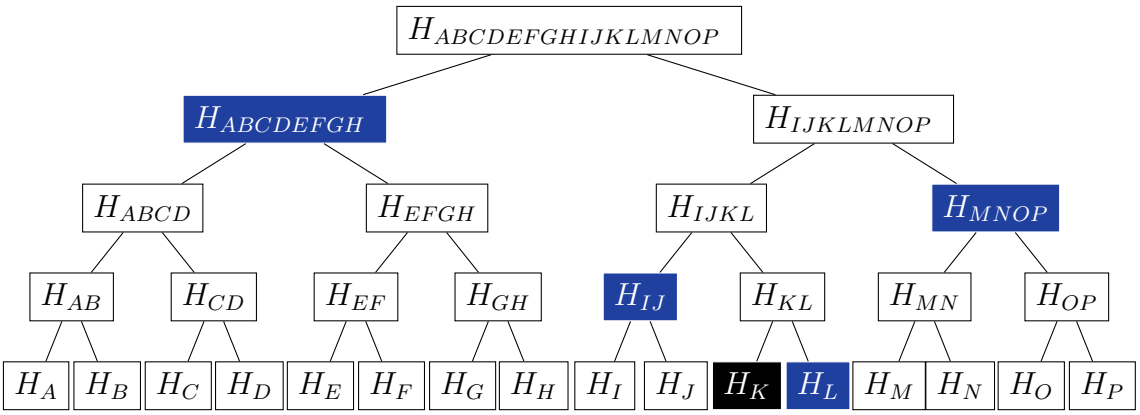

Figura 2: Esempio di Merkle Branch. Sia H_K, la transazione della quale si vuole provare l'appartenenza al blocco, sia gli hash evidenziati con lo sfondo blu fanno parte della Merkle Branch.

1.1 Bloom Filters

Finora abbiamo dato per scontato che un light node riceva solo le Merkle Branch che riguardano transazioni a cui è interessato. Bitcoin però utilizza una comunicazione di tipo *gossip* per trasmettere sia blocchi che transazioni. Questo vuol dire che ogni nodo trasmette indistintamente tutte le informazioni che riceve ai nodi a cui è collegato. Per un light node questo implica ricevere una grande quantità di dati che poi saranno semplicemente scartati, visto che salva su disco solo gli header dei blocchi e le Merkle Branch di suo interesse. Questo spreco di banda è il problema che risolve la BIP 37[5] introducendo i Bloom Filters[6] nelle connessioni tra nodi.

La BIP 37 introduce un meccanismo che permette ai light node di comunicare ai loro peers un Bloom Filter. Per configurare questo filtro vengono implementati 3 nuovi messaggi di protocollo: `filterload`, `filteradd` e `filterclear`. Il filtro viene poi utilizzato dai peers per determinare se una transazione appartiene all'insieme di transazioni che interessano al light node. In questo modo il light node riceve, oltre ai header dei blocchi, solo le Merkle Branch che sono di suo interesse.

1.2 Considerazioni di Sicurezza per i Light Node

Abbiamo visto che un light node è in grado di verificare soltanto la validità della Proof of Work inclusa negli header che riceve. Non può verificare la validità delle transazioni perchè non conosce le altre transazioni della catena. E riceve dai loro peers delle Merkle Branch che gli permettono di verificare la loro appartenenza a un blocco. Un light node quindi non contribuisce alla validazione di nessuna transazione nella rete di Bitcoin. Da un altro lato un light node è più suscettibile, rispetto a un full node, ad attacchi che si basano nell'isolare il nodo dal resto della rete Bitcoin. Non è necessario in effetti trasmettere delle transazioni valide al light node. Una volta che l'attaccante è sicuro di aver isolato il nodo deve trasmettere solo header di blocchi e Merkle Branch per convincerlo di una storia di transazioni diversa da quella che il resto della rete vede. L'attaccante dovrà comunque creare delle Proof of Work e delle Merkle Branch valide perchè vengono validate dal nodo. L'attaccante quindi può convincere al light node che una transazione non valida sia stata inclusa nella catena di blocchi ma

[5] Si veda la pagina github della BIP 37 per approfondimenti.

[6] Un Bloom Filter è una struttura dati probabilistica che permette stabilire in modo efficiente se un elemento è membro di un insieme.

deve comunque spendere risorse computazionali per creare le Proof of Work, tante prove quante il nodo attaccato richieda per considerare la transazione eseguita. Si può dire che per l'attaccante il vantaggio economico dipenda della quantità di Bitcoin trasferiti nella transazione rispetto alla quantità di Bitcoin che potrebbe aver minato.

Un'altro tipo di attacco, forse meglio chiamarlo disservizio, al quale un light node è soggetto è quello che consiste nel non fornire le Merkle Branch che gli interessano. I suoi peers in effetti possono semplicemente non trasmettergiele. Ovviamente questo disservizio funziona se l'attaccante ha isolato il nodo dal resto della rete, altrimenti un nodo onesto potrebbe fornire la Merkle Branch. Comunque il light node può avere conferma che la sua transazione sia stata inclusa nella catena da altre fonti, e.g. un block explorer.