A method to compute the expected value of the profit gainable competing in the Bitcoin Network with commercial ASIC devices (5^{th} draft)

Luca Polverini
email luca.polverini@gmail.com

9 February 2020

Sommario

In questo capitolo si descrive un metodo per il calcolo del profitto (perdita) ricavabile dall'attività di mining di cryptovaluta.

Indice

1 Introduction

This note shows a simple method to compute the expected value of the coin-base subsides gainable competing as a miner in the Bitcoin network. In particular, we inspect the profitability of running ASIC devices summing to an amount of "hash rate" h.

Many web sites, to compute the profitability, use methods that can be misleading because they don't take into account the global hash rate growth of the Bitcoin network.

2 Trattazione della difficoltà

La funzione di hash SHASUM256 è una applicazione che mappa dati interi non negativi in un intero appartenente all'intervallo chiuso $\left[0, 2^{256} - 1\right]$:

$$h : \mathbb{Z}^+ \cup 0 \to \{z \in \mathbb{Z}^+ \cup 0 \mid z \leq 2^{256} - 1\} \tag{1}$$

In Bitcoin, per produrre un blocco valido, è necessario formare una testata di blocco tale per cui l'hash della stessa dia luogo a un codice h_{block} che rispetti la condizione:

$$h_{block} < T \tag{2}$$

dove T è il valore target (*bnTarget* in pow.cpp). In Bitcoin la difficoltà corrente è indicata nel campo nBits della testata del blocco. Il formato del campo nBits è Compact, maggiori informazioni sono in appendice.

La difficoltà di questa ricerca è dovuta al fatto che gli hash generati nel corso della *Proof of Work* sono distribuiti uniformemente nell'intervallo $\left[0, 2^{256} - 1\right]$ ma il valore target T è molto inferiore a $2^{256} - 1$.

Per scelta progettuale T_{max}, il massimo valore che T può assumere, vale $2^{224} - 1$, e dunque:

$$T \leq T_{max} = 2^{224} - 1 \ll 2^{256} - 1 \tag{3}$$

La difficoltà D è definita nel file blockchain.cpp e è data dalla seguente espressione:

$$D = \frac{T_{max}}{T} \tag{4}$$

La ricerca di un hash valido è una **prova di Bernoulli** che ha probabilità di successo:

$$p(h_{block}^{(i)} < T) = \frac{T}{2^{256}} = \frac{T_{max}}{D \cdot 2^{256}} = \frac{1}{2^{32} \cdot D} \tag{5}$$

dove i rappresenta l'i-esimo tentativo.

Dalla (5), si evince che la massima probabilità possibile è data da $D = 1$ e vale $p = 2^{-32}$.

La distribuzione di probabilità del numero di tentativi k necessari per per ottenere un hash valido è data dalla distribuzione geometrica:

$$P(X = k) = (1 - p)^{k-1} \cdot p^1 \tag{6}$$

per $k \in \mathbf{Z}^+$

La distribuzione geometrica ha valore atteso e varianza date dalle seguenti espressioni

$$\mathbb{E}(X) = p^{-1} = \frac{2^{256}}{T} \tag{7}$$

Dal quale si ricava che il numero medio di tentativi necessari è molto ampio ed ha varianza $\approx \frac{1}{p^2}$:

$$\sigma^2(X) = \frac{1 - p}{p^2} \tag{8}$$

In data odierna, i valori sono i seguenti:

$$\left\{ \begin{array}{l} D = 1.55 \cdot 10^{13} \\ p = 1.51 \cdot 10^{-23} \\ \mathbb{E}(X) = 6.64 \cdot 10^{22} \\ \sigma^2(X) = 4.41 \cdot 10^{45} \end{array} \right\} \tag{9}$$

Un processo di arrivi di Poisson è un processo stocastico utile per modellare gli arrivi in un sistema. Può essere visto come la versione continua di un processo di Bernoulli. Per un processo di Bernoulli gli arrivi possono verificarsi soltanto in corrispondenza in multipli interi di un incremento dato.

In un processo di Poisson gli arrivi possono verificarsi arbitrariamente e la probabilità di un arrivo per ogni particolare istante è nulla. Ciò significa che non esiste un modo per descrivere la probabilità di arrivo in un punto ed è più facile riferirsi a sequenze di arrivi in un intervallo; tali sequenze sono quantità i.i.d.

In letteratura la ricerca *Proof of Work* è spesso ritenuta modellabile con sufficiente approssimazione attraverso un processo di Poisson di parametro λ.

Se la variabile casuale X rappresenta il numero di arrivi in un intervallo di tempo t, indipendente dall'origine dei tempi, sotto le le ipotesi:

- uniformità
- indipendenza e assenza di memoria
- ordinarietà

la densità si distribuisce come una variabile casuale di parametro λ.

$$\lambda = \frac{h}{2^{32} D} \tag{10}$$

dove h è la frequenza di hash ($\frac{H}{s}$) e λ è il valore atteso degli arrivi nell'unità di tempo:

$$\mathbb{E}_1(X) = \lambda \tag{11}$$

La distribuzione di probabilità (PDF) ha la seguente forma:

$$f(X;t) = \lambda \cdot e^{-\lambda \cdot t} \tag{12}$$

dove t è la durata della batteria di prove espresso in secondi (s).

La distribuzione di probabilità cumulativa (CDF) è data da:

$$P(X;t) = 1 - e^{-\lambda \cdot t} \tag{13}$$

2.1 Esempio

Se $h = 1\ GH/s$ e $D = 1690906$ la remunerazione per ogni arrivo e $t = 86400\ s$ (1 giorno), la probabilità di arrivo in un giorno è pari a:

$$\lambda = \frac{h \cdot t}{2^{32} \cdot D} = \frac{1 \cdot 10^9 \cdot 86400}{2^{32} \cdot 1690906} \approx 0.0119 \tag{14}$$

Se $S_B = 12.5\ BTC$ BTC, possiamo ricavare il sussidio utilizzando dalla seguente:

$$S = \lambda \cdot S_B \tag{15}$$

che nel caso in esame è pari a 0.59 BTC (i valori attuali sono molto distanti da questo semplice esempio!).

La varianza del processo è ancora λ, dunque 0.0119, e la deviazione standard $\sigma = 0.109$, circa 9.17 volte superiore.

Va sottolineato che il processo è random e senza memoria.

Imponendo la (13) a 0.8 otteniamo il seguente valore per t:

$$P(X;t) = 1 - e^{-\lambda \cdot t} = .80 \Rightarrow e^{-\lambda \cdot t} = 0.20 \Rightarrow \lambda \cdot t = -\ln(0.20) \Rightarrow t \approx 135 giorni! \tag{16}$$

3 Stima dell'hash globale del network

Il protocollo Bitcoin adatta il valore della difficoltà D ogni 2016 blocchi in maniera tale da inseguire il valore 1 blocco ogni 600 secondi.

Approssimando, è dunque possibile ricavare una stima della frequenza di hash della rete H.

$$\lambda_{net} \cdot t = \frac{H}{2^{32}D} \cdot t = 1 \Rightarrow H = \frac{2^{32} \cdot D}{t} \tag{17}$$

per $t = 600$ sec

4 Riduzione della varianza mediante la partecipazione in un Pool

La partecipazione ad un pool di mining di "potenza" $h_{pool} = k \cdot h$ induce una varianza σ^2_{pool} k volte inferiore.

Se ad esempio $K = 100$, pur mantenendo costante il valore atteso pari a 0.59 BTC, la P_{pool} diviene

$$t = -\frac{\ln(0.20)}{k \cdot \lambda} \approx 1.35 \ giorni! \tag{18}$$

La varianza e ridotta proporzionalmente a k.

Il gestore del pool remunera la propria attività trattenendo una quota f del sussidio. Gli schemi adottati sono vari, ed includono la trattazione delle commissioni. In questa parte assumeremo l'uso della schema PPS e trascureremo, per il momento, le parte reltiva alle commissioni.

Alcuni degli schemi adottati da alcuni pool possono essere consultati.

In questo scenario dovremo tenere in considerazione che il ricavato dall'attività di mining, beneficiando della riduzione di varianza, subirà una riduzione:

$$\mathbb{E} = (1 - f) \cdot \frac{h \cdot t}{2^{32} \cdot D} \tag{19}$$

```
/* Example */
var k = 1000; // partecipanti pool
var P = .95;
var D = 15466098935554.65;
var day = 86400; // seconds
var h = 73 * Math.pow(10, 12);
var lambda = k * h / (Math.pow(2, 32) * D);
console.log("lambda pool", lambda)
var p = lambda * day;
console.log("p pool(1 day) = ", p)
var t = - Math.log(1- P) / lambda / (day);
console.log("P = ", P, "=> time = ", t, "days")
console.log("pool pay out day", p * 25 * 10000)
var BTCUSD = 9500
var b1 = p * 25 * BTCUSD / k
console.log("pay out day", b1)
var e = 24 * 2.9 * 0;
var c1 = e + 2000/365;
console.log("costs", c1)
console.log("profit per kWh", (b1 - c1)/24/2.9)
```

5 Hash rate growth

The global hash rate can be esteemed indirectly through the difficulty that is recorder in the block header:

$$\widehat{H_i} = \frac{2^{32}}{600} \cdot D_{i+1} \tag{20}$$

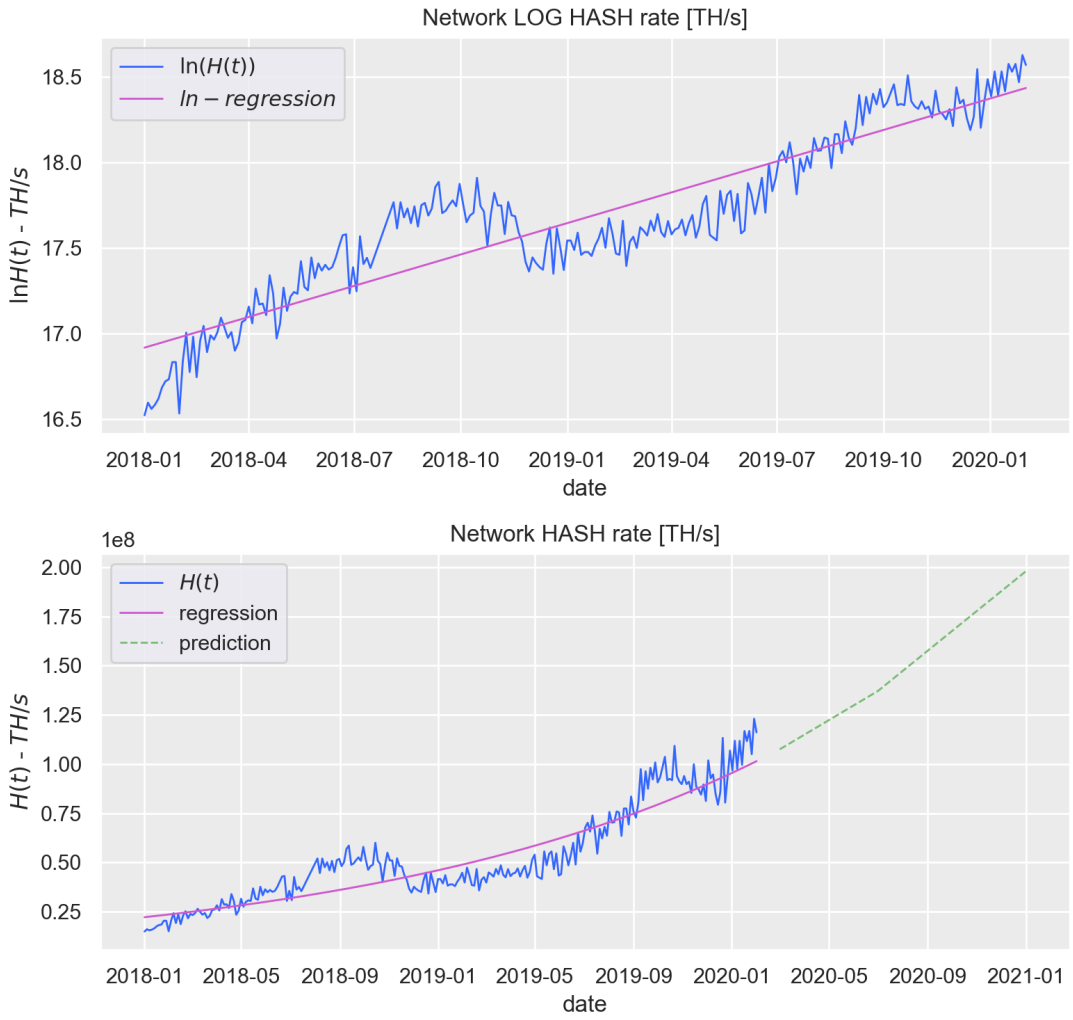

Figura 1: Global log-hash rate of the Bitcoin network from Jan, 2018 to Feb, 2020.

We have downloaded the estimated global hash rate data $\widehat{H}(t)$ from the web site https://www.blockchain.com/.

The Figure (1) shows the natural logarithm of the hash rate growth $\ln(H(t))$ from Jan 1, 2018 to Feb 1, 2020.

The time unit is seconds (s), and the format is the unix epoch, thus seconds elapsed from January 1, 1970.

The hash rate unit is tera hashes per seconds (TH/S).

Given the data, we have fit the data to a line using the log-linear regression:

$$\ln(\widehat{H}(t)) = m \cdot t + q \tag{21}$$

from which we have derived:

$$\widehat{H}(t) = e^{q} \cdot e^{m \cdot t} \tag{22}$$

Equation (22) serves as a short term predictor for future global hash rate values.

The regression exhibits an acceptable fitness to the data. In the future developments of this method, we will show errors and residuals.

We know that growth might rise or decline, but the behaviour is reasonably log-linear since 2014.

Fitting the data, the parameters of the equation (22) are the following values:

$$\left\{ \begin{array}{l} m = +2.29349988795e-08 \\ q = -17.816879116 \end{array} \right\} \tag{23}$$

6 Probability of winning a block

The Bitcoin protocol is coded to adjust the difficulty in such a way that the expected time to build a block is 10 minutes (600 seconds).

So we can imagine the rent-seeking competition among miners ([1]) to build a new block as composed of rounds of 10 minutes each.

Given the global network hash rate $H(t)$ and the hash rate owned by a single miner (or mining pool) $h(t)$, we can compute the probability $p_{win}(t)$ of gaining the subside S_{block} for a single round $r(t)$ that starts at t and ends at $t + 600$ seconds under the assumption that all the competitors use the "same technology" and the "same algorithms".

The same algorithms assumption seems to be accepted as guaranteed by the Bitcoin protocol and by the nature of the Proof of Work and its cryptographic hash functions (the underlying probability distribution is uniform) that entitles us to consider the random variables independent and identically distributed.

The same technology assumption is weak and serves to overcome some issues related to networking and other minors and majors issues.

The probability is:

$$p_{win}(t) = \frac{h(t)}{H(t)} \tag{24}$$

In our scenario, $h(t)$ is constant in time and it is much lower than $H(t)$:

$$h(t) = h_o \ll H(t) \tag{25}$$

So we can write:

$$p_{win}(t) = \frac{h(t)}{H(t)} = \frac{h_0}{H(t)} \tag{26}$$

7 Expectation

We can define a random variable X such that:

$$X = \left\{ \begin{array}{l} X = 1, \textit{ if the miner wins the round} \\ X = 0, \textit{ if the miner loses the round} \end{array} \right\} \tag{27}$$

The expected value of X over n rounds is:

$$\mathbf{E}(X) = \sum_{r=1}^{n} 1 \cdot p_{win}(t(r)) + 0 \cdot (1 - p_{win}(t(r))) \tag{28}$$

Where:

$$t(r) = t_{(r=1)} + 600 \cdot r \tag{29}$$

Rearranging the preceding equations we can write:

$$\mathbf{E}(X) = \sum_{r=1}^{n} \frac{h0}{\widehat{H}(t(r))} \tag{30}$$

Given equation (30) the following code follows:

```
# ------------------------------------------------
# Compute the expected value of X
# ------------------------------------------------

E = 0
r = 0

while(r <= 365 * 24 * 6): # e.g. 1 year
    r += 1
    E += h0 / H(t(r))

return E
```

Or, if we want to compute the expected number of rounds needed to win a block:

```
# ------------------------------------------------
# Compute the number of rounds needed to win a round
# ------------------------------------------------

E = 0
r = 0

while(E < 1): # e.g. 1 win
    r += 1
```

```
    E += h0 / H(t(r))

return r
```

8 Results

Given a single high-end commercial ASIC device, the hash rate is $h_0 = 73\ TH/s$ and under our assumptions the expected number of wins in a year, starting from Feb. 2, 2020, is:

$$\mathbf{E}(X) = \sum_{r=0}^{365*24*6} p_{win}(t(r)) \approx 0.026 \tag{31}$$

Obviously, given $N = 100$ devices of the same model, the result is:

$$\mathbf{E}(X) = \sum_{r=0}^{365*24*6} p_{win}(t(r)) \approx 2.58 \tag{32}$$

9 Economic prospects drafts

In this paragraph, we show two simplified economic prospects.

Given the exhibited exponential growth of the global hash-rate, and given the fact that ASIC devices are not repurposable, we have decided to take into account all the device cost without amortization.

Given the BTC price fluctuation (volatility), we have also avoided taking account of interests rates.

```
Rounds            : 52560 ( 365 ) days
Device hash rate  : 73 TH/s
Expeted wins      : 0.0269128246059
Subside  and fees: 12.55 BTC
Expected BTCs     : 0.337755948804
BTC/USD           : 9450 USD
Expected Value    : 3191.7937162 USD
Energy consum.    : 25404.0 kWh (28.95 W/TH/s)
Energy price.     : 0.1 USD/kWh
Energy cost       : 2540.4 USD
Device cost       : 2000 USD
                    ----------
Total costs       : 4540.4
Expected Profit   : -1348.6062838
          per kWh : -0.0530863755237
```

Figura 2: Model results for $S = 12.5 BTC$

```
Rounds           : 52560 ( 365 ) days
-------------------------------------------------------
Device hash rate : 73 TH/s
Expeted wins     : 0.0269128102299
Subside  and fees: 6.3 BTC
Expected BTCs    : 0.169550704448
BTC/USD          : 9450 USD
Expected Value   : 1602.25415704 USD
Energy consum.   : 25404.0 kWh (28.95 W/TH/s)
Energy price.    : 0.1 USD/kWh
Energy cost      : 2540.4 USD
Device cost      : 2000 USD
                   ----------
Total costs      : 4540.4
Expected Profit  : -2938.14584296
         per kWh : -0.115656819515
```

Figura 3: Model results for $S = 6.25BTC$

9.1 Simulazione del Processo di Poisson

In questo paragrafo simuleremo il processo di arrivo di un hash valido in Bitcoin. Faremo ricorso ad un metodo stocastico.

$$t = \frac{\ln(1-U)}{r} \tag{33}$$

dove U è un numero random corrispondente ad una distribuzione uniforme.

10 Conclusion

Many input variables condition the profitability of the mining activity.

For example, the BTC price and fees, the global hash-rate growth, the energy and devices costs, and so on. Further, we should take into account network costs and others.

So, we should refine this method, but it serves as a starting point to debate and improve current public available practices.

This note highlight that within the assumptions and the given costs, solo-mining (or participating in a small pool) is hardly-profitable and poses some questions that we would further investigate:

- The cost of electric energy has a high impact on profitability. In which condition we might reduce energy costs?
- To take account of other factors that have an impact on the marginal utility (e.g. privacy)
- Is the ASIC market overrated? Considering that the devices are not repurposable and the obsolescence rate is very high, the current commercial price seems unjustified.
- The system seems to be under the pressure of centralizing forces (both geographically and economically). What will change in the next future? The BTC price? The hash-rate growth? What else? How?

With this note, we would give publicity to the fact that the end-user market needs a better model to compute profitability.The proposed method is a point where to start reasoning.

A Blocco

Le informazioni che riguardano la struttura di un blocco sono contenute nei file bitcoin-core primitives/block.h e primitives/block.cpp. Come si legge nel file, i nodi verificano e collezionano le transazioni dentro un blocco (con le logiche che vedremo) e ricavano un hash complessivo delle stesse attraverso una struttura dati che prende il nome di albero di Merkle. Dunque risolvono l'algoritmo di consenso *Proof of Work* che consiste nella ricerca iterativa di un valore nonce 'nNonce' da inserire nella testata del blocco (insieme alle altre informazioni) e che dia luogo a un hash di blocco inferiore ad un numero dato che prende il nome di *target*. Il target è collegato alla [difficoltà](./difficolta.md) 'nBits'. Risolto questo algoritmo trasferiscono il blocco agli altri nodi e tutti, se valido, lo inseriscono nella blockchain. La prima transazione del blocco è una transazione speciale che prende il nome di *coinbase*. Si tratta di una transazione che conia moneta a ricompensa del lavoro svolto il nodo che ha trasmesso per primo il nuovo blocco valido. Questa procedura è competitiva: tutti i nodi minatori costruiscoo il blocco destinando la transazione coinbase a un destinatario di loro scelta (in genere il minatore stesso). ¿ Dopo [SegWit](segregated-witness.md) l'albero di Merkle contiene anche script e firme tramite l'innesto di un sotto-albero per tramite della transazione coinbase.

B Rappresentazione difficoltà in Bitcoin

In Bitcoin, il campo nBits **codifica** la difficoltà D della Proof of Work del blocco. Non va confuso con la difficoltà D alla quale è legato dalla relazione (4) che riformulata diviene:

$$T = \frac{T_{max}}{D} \tag{34}$$

Il valore **target** è rappresentato in formato Compat, simile ai numeri in virgola mobile IEEE754, e occupa 32 bit della testata.

Esponente (b)	Segno (s)	Mantissa (a)
8 bit [31-24]	1 bit [23]	23 bits [22-00]

Per convertire questo valore in formato intero si utilizza la seguente formula:

$$T = (-1)^s \cdot a \cdot 256^b \tag{35}$$

B.1 Conversione da nBits a Target

La procedura per computare T è la seguente:

- ricavare la mantissa tramite i primi 23 bits: nBits & 0x007fffff
- ricavare il segno: nBits & 0x00800000 != 0
- ricavare l'esponente shiftando a destra 24 bit: nBits $\gg$ 24
- se $b \leq 3$: translare a destra di $8 \cdot (3 - b)$ bit la mantissa
- se $b > 3$: translare a sinistra di $8 \cdot (b - 3)$ bit la mantissa
- applicare il segno

B.1.1 Esempio

Ad esempio il valore di nBits del blocco di genesi vale nBits = 0x1d00ffff da cui:

$$
\begin{aligned}
&a = \text{0x1d00ffff} \ \& \ \text{0x007fffff} = \text{0xffff} \\
&s = (\text{0x1d00ffff} \ \&\text{0x00800000!} = 0) = 0 \\
&b = \text{0x1d00ffff} \gg 24 = \text{0x1d} > 3 \\
&a = a << 8 \cdot (b - 3) = \text{0xffff} << \text{0xd0} = \text{0xffff0000000000000000000000000000000000000000000000000000} \\
&a = (-1)^0 \cdot a = \text{0xffff0000000000000000000000000000000000000000000000000000}
\end{aligned}
\tag{36}
$$

Il valore del target massimo, in esadecimale, è il seguente:

$T_{max} = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF$ (56 volte F = $\frac{224}{\ln(16)}$),

che, a causa della rappresentazione compatta, diviene:

$T_{max} = 0xFFFF0000000000000000000000000000000000000000000000000000$ (4 volte F e 52 volte 0).

B.2 Conversione da Target a nBits

Il valore nBits è settato dal metodo unsigned int *GetNextWorkRequired()* che, sotto le opportune condizioni, invoca *GetCompact()*.

Come si vede dal codice sotto riportato si tratta della procedura inversa.

```
// arith_uint256
template <unsigned int BITS>
unsigned int base_uint<BITS>::bits() const
{
    for (int pos = WIDTH - 1; pos >= 0; pos--) {
        if (pn[pos]) {
            for (int nbits = 31; nbits > 0; nbits--) {
                if (pn[pos] & 1U << nbits)
                    return 32 * pos + nbits + 1;
            }
            return 32 * pos + 1;
        }
    }
    return 0;
}

uint32_t arith_uint256::GetCompact(bool fNegative) const
{
    int nSize = (bits() + 7) / 8;
    uint32_t nCompact = 0;
    if (nSize <= 3) {
        nCompact = GetLow64() << 8 * (3 - nSize); // qui viene impostata la
    mantissa
    } else {
        arith_uint256 bn = *this >> 8 * (nSize - 3);
        nCompact = bn.GetLow64(); // qui viene impostata la mantissa
    }
    // The 0x00800000 bit denotes the sign.
    // Thus, if it is already set, divide the mantissa by 256 and increase the
    exponent.
    if (nCompact & 0x00800000) {
        nCompact >>= 8;
        nSize++;
```

```
32      }
33      assert((nCompact & ~0x007fffff) == 0);
34      assert(nSize < 256);
35      nCompact |= nSize << 24; // qui viene impostato l'esponente
36      nCompact |= (fNegative && (nCompact & 0x007fffff) ? 0x00800000 : 0); // qui
        viene impostato il segno
37      return nCompact;
38  }
```

C Proof of Work

Bitcoin-core esegue la PoW in mining.cpp all'interno della funzione generateBlocks.

```
1  // permette di inizializzare il nuovo blocco prima della PoW con
2  void IncrementExtraNonce(CBlock* pblock, const CBlockIndex* pindexPrev, unsigned
       int& nExtraNonce)
3  {
4    //...
5  }
```

Poi con un ciclo while viene modificato solo il nounce

```
1  while (nMaxTries > 0 && pblock->nNonce < std::numeric_limits<uint32_t>::max() && !
       CheckProofOfWork(pblock->GetHash(), pblock->nBits, Params().GetConsensus()) &&
        !ShutdownRequested()) {
2      ++pblock->nNonce; // qui
3      --nMaxTries;
4  }
```

GetNextWorkRequired è chiamata da UpdateTime chiamato da CreateNewBlock In maniera analogo dobbiamo modificare la PoW per evitare di inizializzare i dati ad ogni step. In analogia sposterò generateBlock in mining. Al momento MineTx in Network

Riferimenti bibliografici

[1] Eric Budish. The economic limits of bitcoin and the blockchain. 2018.