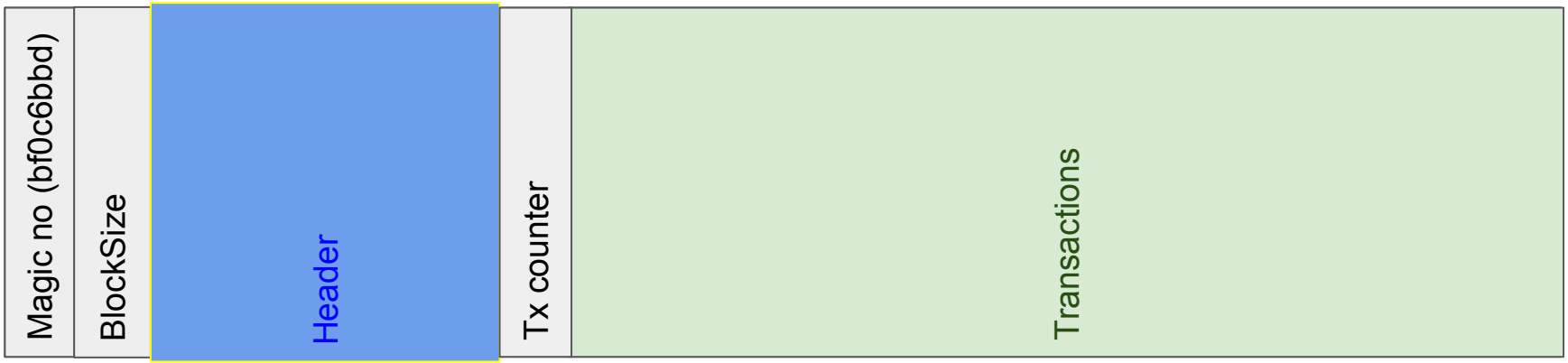
Blockchain

Mining

Block: Structure



Block: Header

Field	Purpose	Updated when...	Size (Bytes)
Version	Block version number	You upgrade the software and it specifies a new version	4
hashPrevBlock	SHA-256d hash of the previous block header	A new block comes in	32
hashMerkleRoot	SHA-256d hash based on all of the transactions in the block	A transaction is accepted	32
Time	Current timestamp as seconds since 1970-01-01T00:00 UTC	Every few seconds	4
Bits	Current target in compact format	The difficulty is adjusted	4
Nonce	32-bit number (starts at 0)	A hash is tried (increments)	4

Block: Hash vs PoW

1. Block Hash **is not** PoW but sometimes **is**
2. Block Hash = SHA256d (Block Header)
3. PoW = <PoW Algorithm> (Block Header)
    a. PoW(Dash) == X11
    b. PoW(LTC) == SCRYPT
    c. PoW(BTC) == SHA256d

Block: Header: variables

Field	Purpose	Updated when...	Size (Bytes)
Version	Block version number	You upgrade the software and it specifies a new version	4
hashPrevBlock	SHA-256d hash of the previous block header	A new block comes in	32
hashMerkleRoot	SHA-256d hash based on all of the transactions in the block	A transaction is accepted	32
Time	Current timestamp as seconds since 1970-01-01T00:00 UTC	Every few seconds	4
Bits	Current target in compact format	The difficulty is adjusted	4
Nonce	32-bit number (starts at 0)	A hash is tried (increments)	4

The **Nonce** starts at 0 and is incremented for each hash. Whenever it overflows, the **extraNonce** portion of the **generation transaction** is incremented, which changes the **Merkle root**.

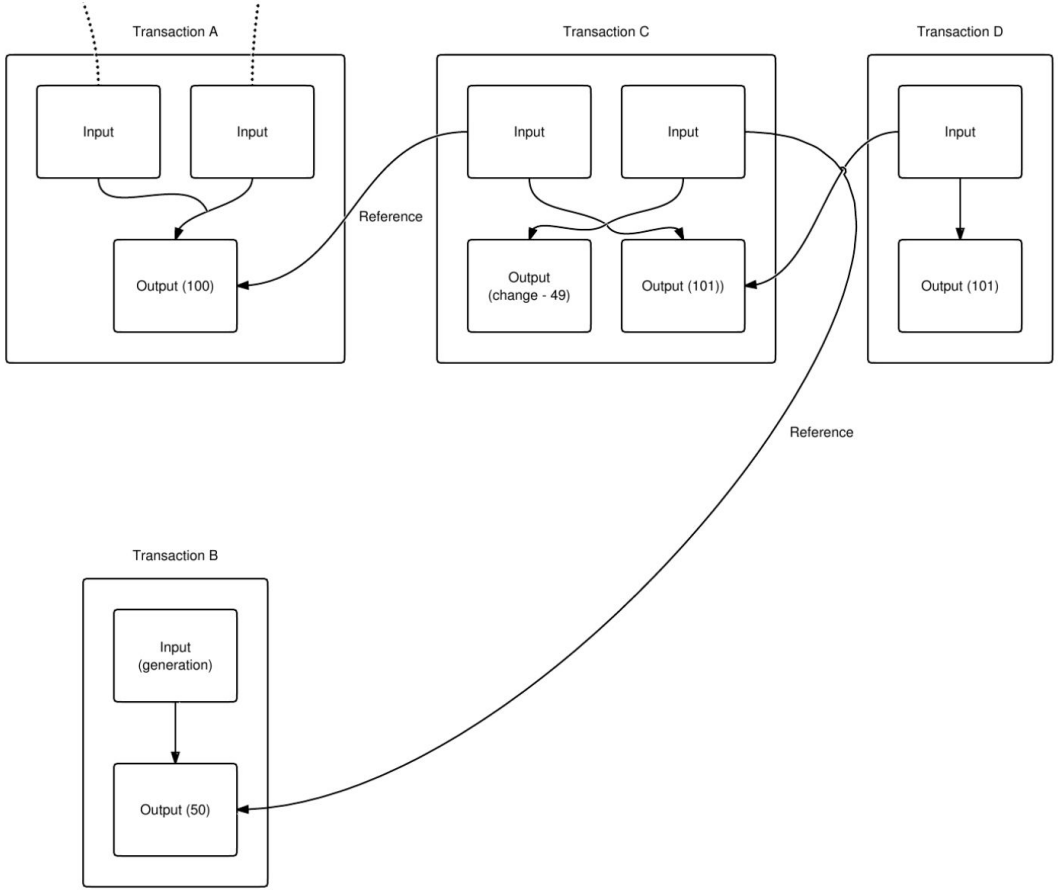
Block: Transactions

Field	Description	Size
Version no	currently 1	4 bytes
In-counter	positive integer VI = VarInt	1 - 9 bytes
list of inputs	the first input of the first transaction is also called "coinbase" (its content was ignored in earlier versions)	<in-counter>-many inputs
Out-counter	positive integer VI = VarInt	1 - 9 bytes
list of outputs	the outputs of the first transaction spend the mined bitcoins for the block	<out-counter>-many outputs
lock_time	if non-zero and sequence numbers are < 0xFFFFFFFF: block height or timestamp when transaction is final	4 bytes

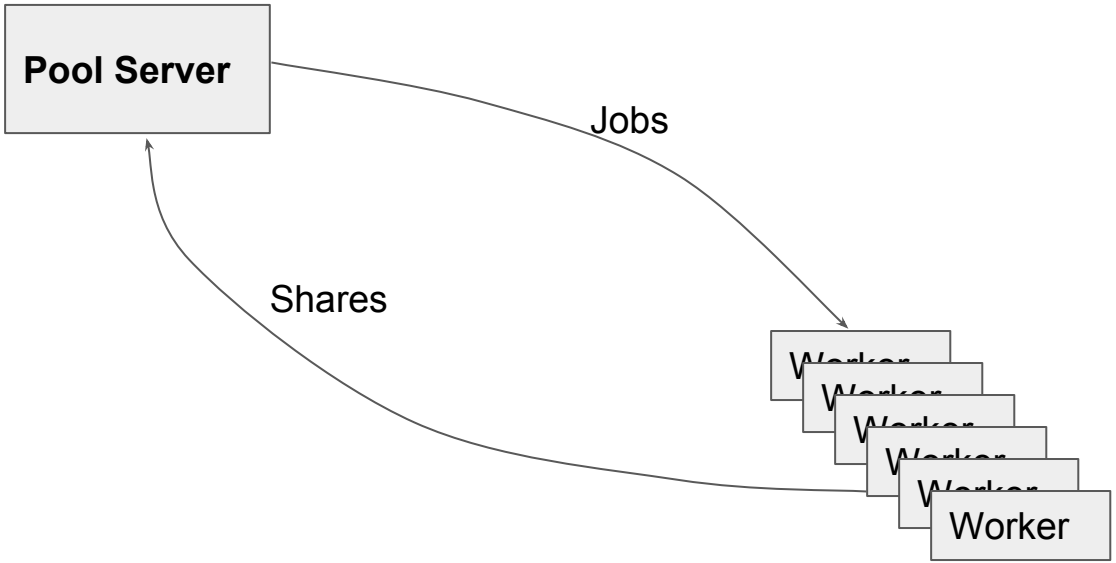
```
Input:
Previous tx: f5d8ee39a430901c91a5917b9f2dc19d6d1a0e9cea205b009ca73dd04470b9a6
Index: 0
scriptSig: 304502206e21798a42fae0e854281abd38bacd1aeed3ee3738d9e1446618c4571d10
90db022100e2ac980643b0b82c0e88ffdfec6b64e3e6ba35e7ba5fdd7d5d6cc8d25c6b241501

Output:
Value: 5000000000
scriptPubKey: OP_DUP OP_HASH160 404371705fa9bd789a2fcd52d2c580b65d35549d
OP_EQUALVERIFY OP_CHECKSIG
```

Block: Transactions: Money moving



Pool and workers: stratum protocol



Pool: Jobs

1. Authenticate miner. Client sends:
`mining.authorize("username", "password")`

2. Get subscription:
client sends: `mining.subscribe("user agent/version", "extranonce1")`
client receives: `[[["mining.set_difficulty", "subscription id 1"], ["mining.notify", "subscription id 2"]], "extranonce1", extranonce2_size]`

3. Server sends:
`mining.notify(...):` Fields in order:

1. *Job ID*. This is included when miners submit a results so work can be matched with proper transactions.
2. *Hash of previous block*. Used to build the header.
3. *Generation transaction (part 1)*. The miner inserts ExtraNonce1 and ExtraNonce2 after this section of the transaction data.
4. *Generation transaction (part 2)*. The miner appends this after the first part of the transaction data and the two ExtraNonce values.
5. *List of merkle branches*. The generation transaction is hashed against the merkle branches to build the final merkle root.
6. *Bitcoin block version*. Used in the block header.
7. *nBits*. The encoded network difficulty. Used in the block header.
8. *nTime*. The current time. nTime rolling should be supported, but should not increase faster than actual time.
9. *Clean Jobs*. If true, miners should abort their current work and immediately use the new job. If false, they can still use the current job, but should move to the new one after exhausting the current nonce range.

Pool: Shares

Client sends whenever it founds PoW with difficulty > Job difficulty:

```
mining.submit("username", "job id", "ExtraNonce2", "nTime", "nOnce")
```

Miners submit shares using the method "mining.submit". Client submissions contain:

1. *Worker Name*.
2. *Job ID*.
3. *ExtraNonce2*.
4. *nTime*.
5. *nOnce*.

Block: Difficulty

Target: 0x0000000000000000FF...FF -> 64 symbols = 32 bytes = 256 bits

Target is for PoW!

Target_Max = 2**236 = 0x00000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Target_Min = 2**0 = 0x0000000000000000000000000000000000000000000000000000000000000001

Often truncated: 0x00000000FFFF0000000000000000000000000000000000000000000000000000

Difficulty = 0xFFFF * 2**208 / target = 2**224/target

difficulty_min= 2**(-12)

Network performance = 2**256 / target / Tb = D * 2**256 / (2**224) / Tb = D * 2**32 / Tb
where
D - difficulty,
Tb - block issuance period in seconds

Dash

Retargetting **every ~~288 blocks~~ single block ?**

Network Difficulty:	6 621 029.926 111 49
Tb (target):	150 сек
Network Hash Rate (current):	161.13 TH

=>

Tb (current) = 176 sec

Average Network Hash Rate = 189 TH

Block: probability

1. Simple probability calculations:
   a. p = target / 2**256 = 2**224 / D / 2**256 = 1 / (D * 2**32) - probability of getti
      (difficult enough) block hash per one try (iteration/every single block hash calculation)
   b. ph = H / (D * 2**32) = 2.3 * 10**(-10) * H / D - probability of getting one valid
      as a function of Network Difficulty D and miner's hashrate H [h/s],
   c. ph = H / (D * 2**32) = H / (TH * Tb) - probability of getting one valid hash pe
      function of Total Network Hashrate TH [h/s], miner's hashrate H [h/s] and bloc
      period Tb [s]
   d. pB = H / TH - probability of getting valid hash per Block ROUND (lasts Tb sec
      function of Total Network Hashrate TH [h/s] and miner's hashrate H [h/s]

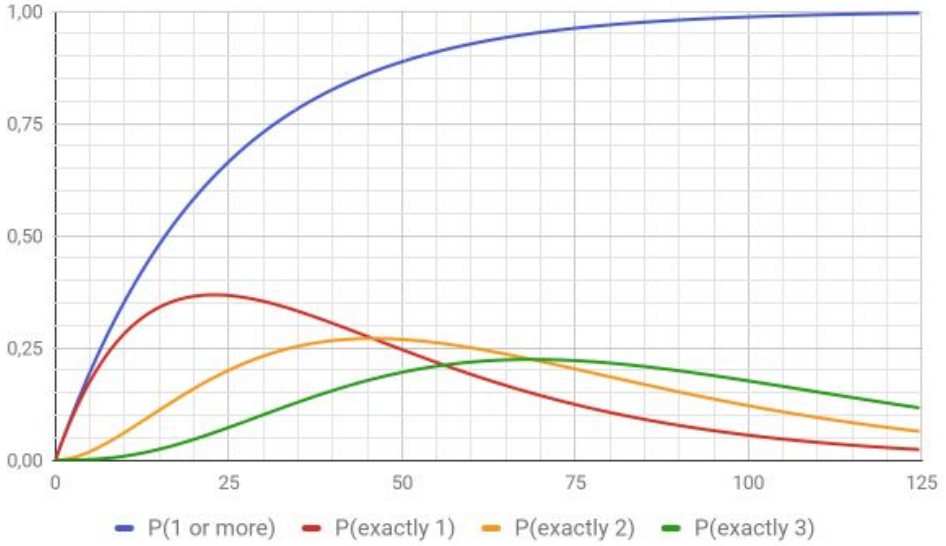
Block: probability more precisely

26. **Probability of getting one block as a function of time**:
    a. Probability of getting valid hash (PoW algorithm) per Block ROUND (Tb ~2.5 minutes for LTC/DASH, ~10 minutes for BTC): **p = H/TH**, where H - miner's hashrate [h/s], TH - Total Network Hashrate [h/s]
    b. In accordance with Binomial Distribution the probability of getting exactly **k (no more or less, exactly k) blocks per n Block ROUNDS**: **P(k,n) =** $C_n^k \times p^k \times q^{n-k}$, where $C_n^k = \frac{n!}{k!(n-k)!}$ - binomial coefficients, p - probability of getting valid hash (PoW algorithm) per Block ROUND, $q = 1 - p$.
    c. k=1,n: P(1,n) = $n \times p \times (1-p)^{n-1}$
    d. k=2,n: P(2,n) = $\frac{n(n-1)}{2} \times p^2 \times (1-p)^{n-2}$
    e. k=3,n: P(3,n) = $\frac{n(n-1)(n-2)}{6} \times p^3 \times (1-p)^{n-3}$ and so on.
    f. Probability of getting **1 or more blocks** per n Block ROUNDS is a summ of all binomial elements (which equals 1) minus only one element C_n^0, i.e. P(1..n,n) = $1 - C_n^0 \times p^0 \times (1-p)^{n-0}$, i.e. **P(1..n,n) =** $1 - (1-p)^n$
    g. To get probability of getting one block as a function of time we should multiply n on Tb (Block ROUND time aka block issuance period).

Block: probability: graph example

h. Example of LTC: Tb = 2.5 minutes = 150 seconds, initial data is in the table below. Graph represents single probabilities and "1 or more block" probability:

Parameter	Symbol	Value	Units
Pool hashrate	H	20	GH/s
Total network hashrate	TH	11	TH/s
Block ROUND time aka block issuance period	Tb	150	sec



Pool: Difficulty settings


```
"diff": 24,
        "varDiff": {
            "minDiff": 24,
            "maxDiff": 64,
            "targetTime": 15,
            "retargetTime": 90,
            "variancePercent": 10
        }
```