Modeling Bitcoin Protocols with Probabilistic Logic Programming

Damiano Azzolini, Fabrizio Riguzzi, Evelina Lamma, Elena Bellodi, and Riccardo Zese

Azzolini Damiano - damiano.azzolini@student.unife.it

1 September 2018

Università degli Studi di Ferrara

Bitcoin - Blockchain

Blockchain

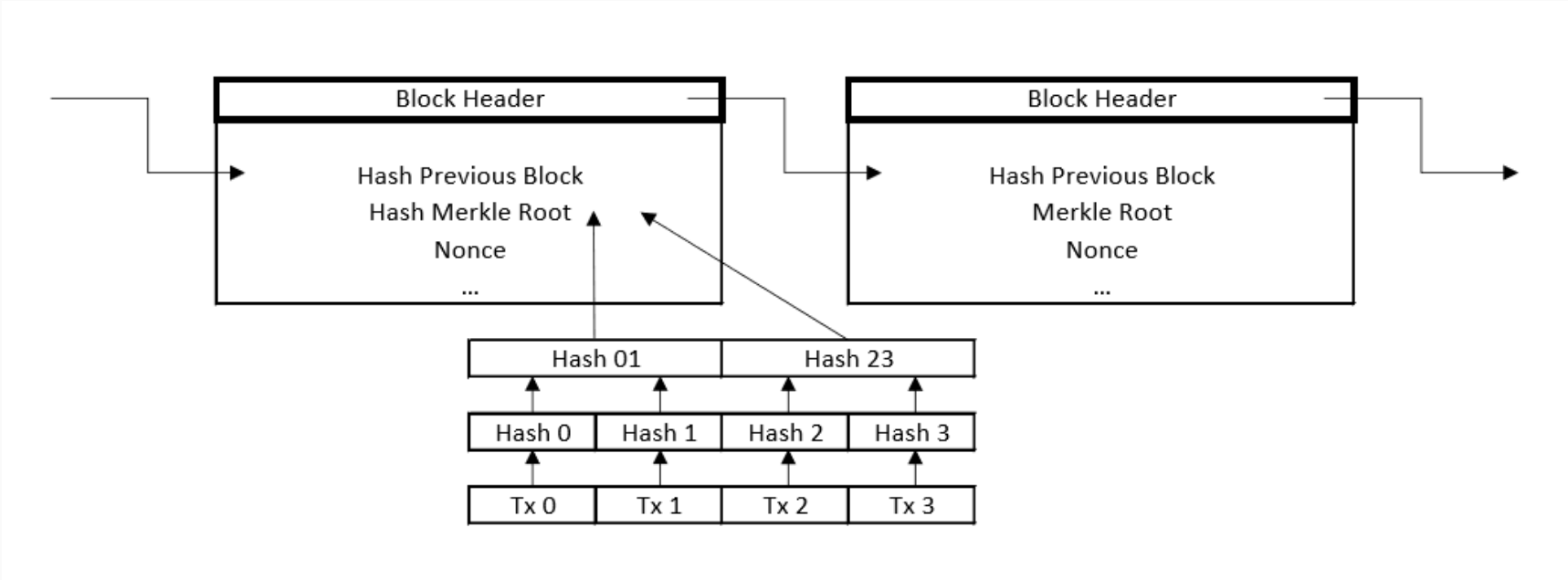
Distributed append-only public ledger: each peer in the network has a copy of this ledger.

- Blockchain contains blocks.
- Block contains transaction.
- User sends transaction.

No need of trusted third party.

Technology underlying cryptocurrencies such as Bitcoin.

Bitcoin Blockchain



Consensus - 1

Validation is required in order to append a new block to the chain. This process is called *mining*. If a peer is able to find the correct hash, it is rewarded with a certain number of bitcoins plus the fees of the transactions included in the block.

The chain can have different branches: the miners add the block to the **longest** branch because only the longest one is the valid one.

Consensus - 2

Bitcoin uses Proof-of-Work (hashcash): find a *nonce* value such as:

$$SHA256(SHA256(blockHeader)) \leq T$$

T is a value called target and SHA-256 is the hash function "Secure Hash Algorithm" that returns a 256 bit hash of the input. This value is dynamically changed every 2016 blocks found to ensure that new blocks will be generated at regular intervals, one every 10 minutes on average.

Drawbacks

- Centralization of hashing power.
- Forks.
- More...

Hashing Power Distribution

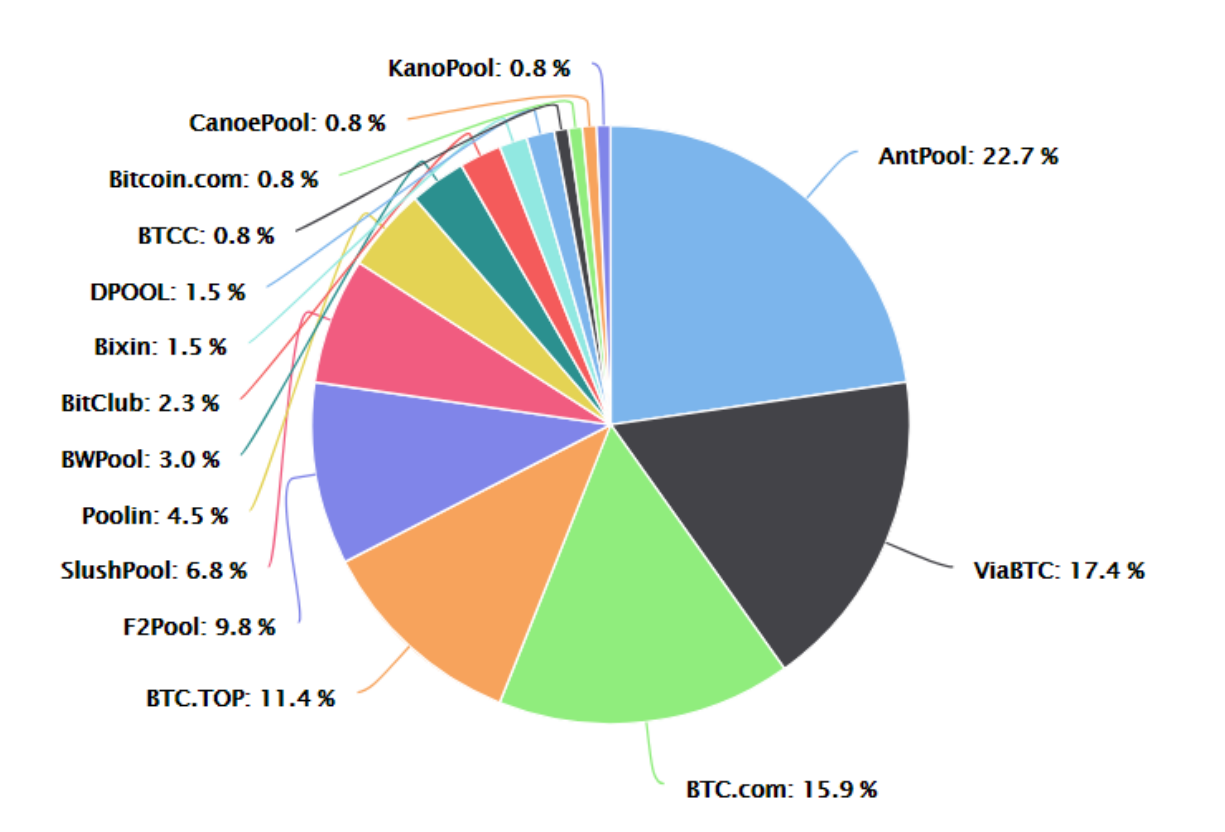


Figure 1: Hashing power distribution - 25/08/2108

Source: `https://btc.com/stats/pool?pool_mode=day`

Preventing Large Pools Formation

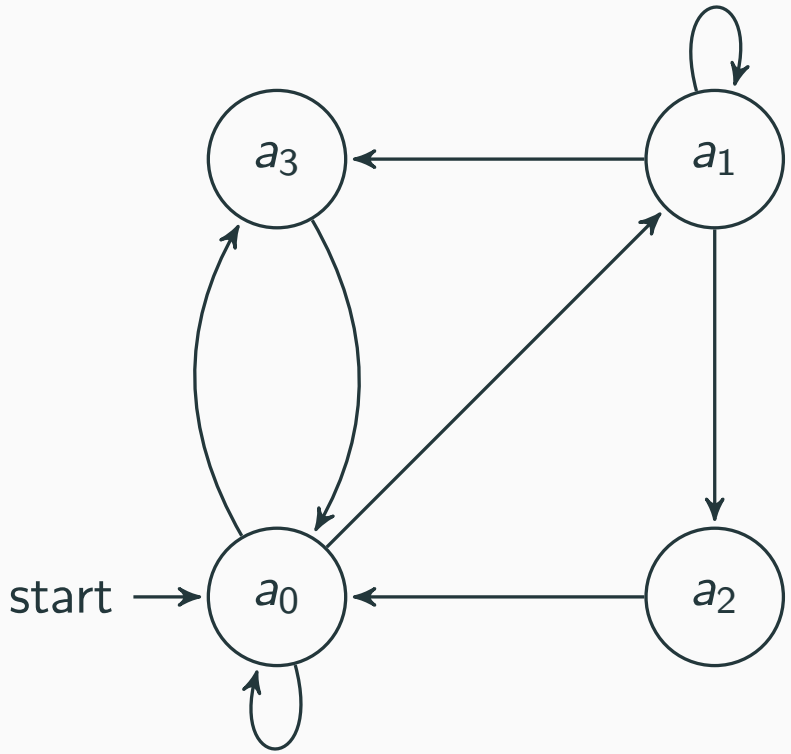
Problem

To increase the probability to find the correct hash of a block, miners usually organize themselves into a mining pool: this situation can generate a super pool that holds more than 50% of the total hash power.

Possible solution: 2-phase PoW. Traditional PoW + signing the header with the private key of the address that will receive the mining reward and then finding a new hash smaller than a certain value.

2-phase PoW

- State 0 (a0): a miner generates a hash using a certain nonce. If this hash is correct, it will move to state 1, if it's not, it will stay in a0.
- State 1 (a1): the miner has already solved the first hash puzzle and now needs to solve the second one.
- State 2 (a2): the miner has solved both hashes, is rewarded and now is ready to mine another block (back to state 0).
- State 3 (a3): another miner has solved the second hash, so the first miner can now stop working on this hash and move to another one (back to state 0).



CPLINT Code - 1

```
a_found_y(_):0.15.
b_found_y(_):0.25.
b_found_x(_):0.10.
found_y(S):- a_found_y(S); b_found_y(S).

trans(a0,S,a1):1.0/50; trans(a0,S,a0):1.0-1.0/50:- \+b_found_x(S).
trans(a1,S,a2):0.15;trans(a1,S,a1):1.0-0.15:- \+b_found_y(S).

trans(a0,S,a3):- b_found_x(S).
trans(a1,S,a3):- b_found_y(S).

trans(a2,S,a0):- found_y(S).
trans(a3,S,a0):- found_y(S).
```

CPLINT Code - 2

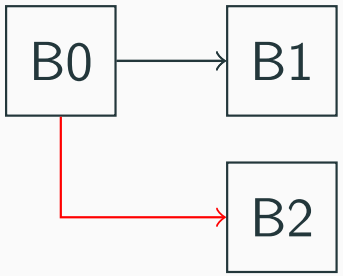
```
% starting at state S at instance I,
% state T is reachable.
reach(S, I, T) :-
   trans(S, I, U),
   reach(U, next(I), T).
reach(S, _, S).

?- mc_sample(reach(a0,0,a2),1000,P).
P = 0.068
```

`mc_sample/3` samples `reach/3` 1000 times and returns the estimated probability that a sample is true.

Double Spending Attack

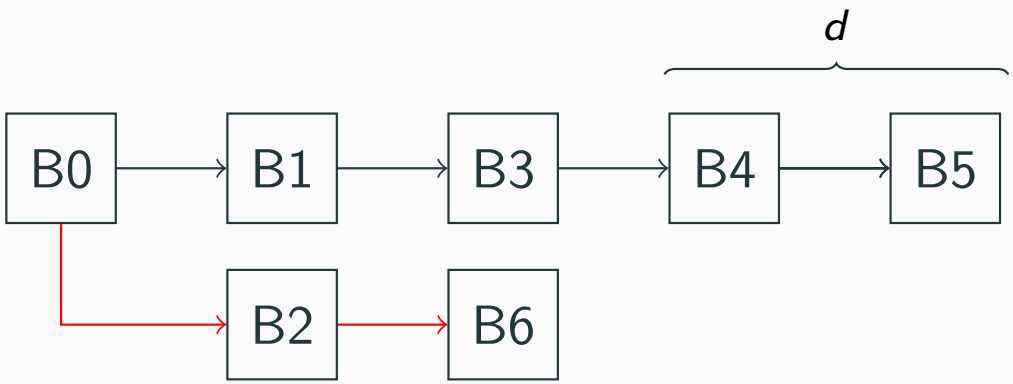
Initial State

Goal: spend the same bitcoin twice through the creation of an alternative chain.



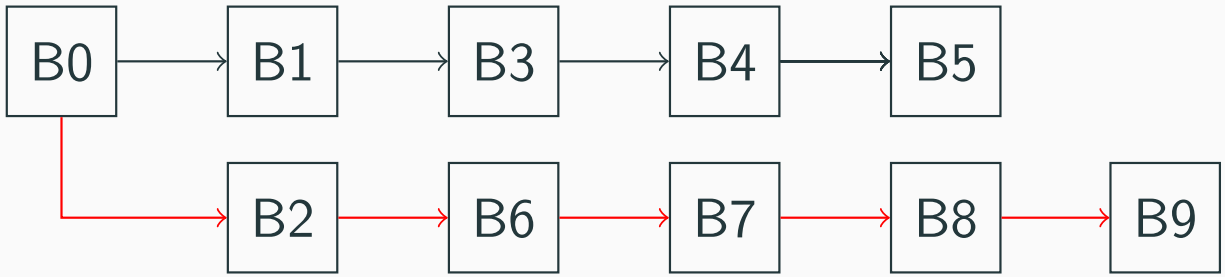
Initial state of the double spending attack. Block $B1$ with transaction $T1$ is inserted in the chain after $B0$, while the attacker starts mining another block ($B2$) with $B0$ as ancestor, without $T1$ inside. $T1$ contains the amount the attacker wants to double spend.

General State



General case. The "honest" chain has built 3 **confirmation** blocks on top of $B1$ ($B3, B4, B5$) while only one block ($B6$) has been built on top of $B2$ by the attacker. In this figure, d represents the distance between the honest and the secret chain and is used to evaluate the advantage of the honest chain over the attacker.

Successful Attack



Successful attack. The attacker has built a longer chain (marked in red). The attacker will now publish all blocks from $B2$ to $B9$ and so all blocks from $B1$ to $B5$ in the black chain will not be considered valid because they are part of a chain which is not the longest one.

Model

Two functions: attacker's potential progress function and the catch up function.

The attacker's potential progress function relates the number m of blocks that are mined by the attacker, while the honest chain has mined n (confirmation) blocks. Modelled with Pascal distribution or Poisson distribution.

The catch up function describes the probability that the attacker can create a chain longer than the honest one. Modelled with binomial random walk.

CPLINT Code - 1

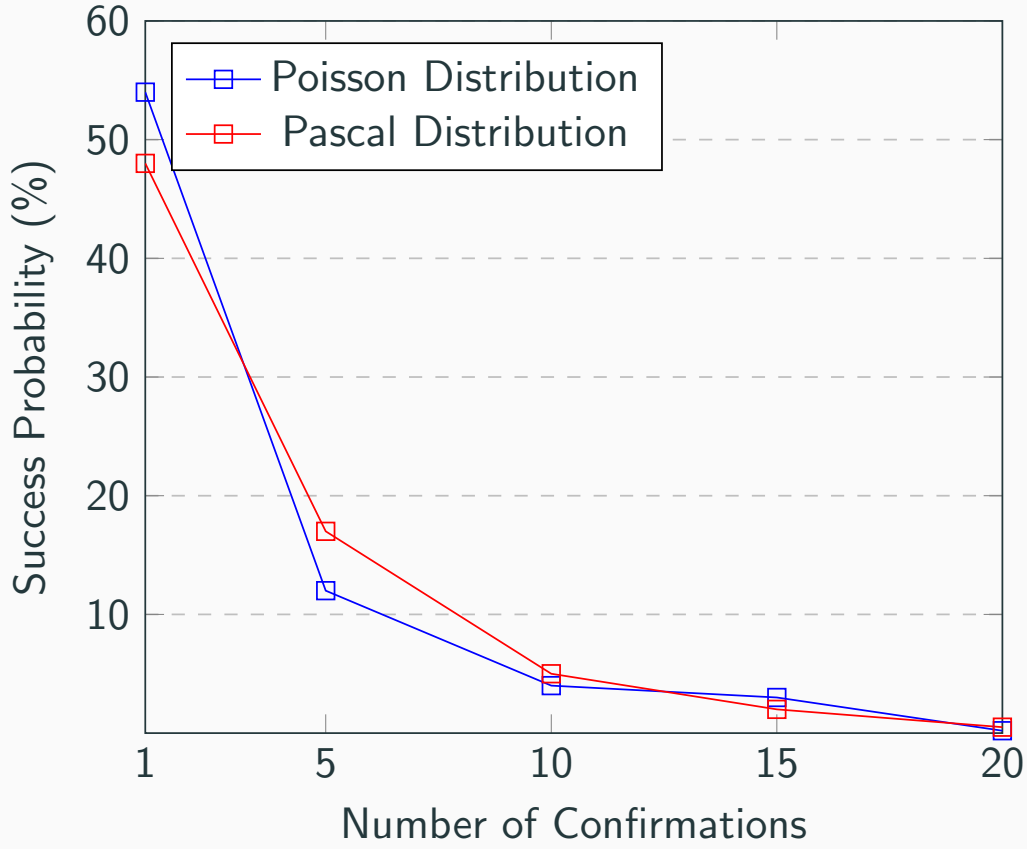
```
% attacker's potential progress function
% 10 confirmation blocks
% 30% of the total hashing
% power controlled by the attacker
attacker_progress_poisson(X):poisson(X,Lambda):-
    Lambda is 10*0.3/0.7.

% catch up function
move(T,1):0.7; move(T,-1):0.3.

walk(InitialPosition):-
   walk(InitialPosition,0).

walk(0,_).
walk(X,T0):-
    X > 0,
    X < 100, % threshold for not winning
    move(T0,Move),
    T1 is T0+1,
    X1 is X+Move,
    walk(X1,T1).
```

CPLINT Code - 2

```
success_poisson:-
    attacker_progress_poisson(A),
    V is 10 - A,
    (   V = 0 ->
     true;
     walk(V)
    ).

?- mc_sample(success_poisson,1000,P).
Prob = 0.036.
```

`mc_sample/3` samples `success_poisson/0` 1000 times and returns the estimated probability that a sample is true.

Success Probability - Double Spending Attack



Values obtained considering that the attacker holds 30% of the total hashing power of the network.

Conclusions and Future Work

Probabilistic Logic Programming can be used both to model the current Bitcoin protocol and to analyze some proposals for improvements.

The model can be extended also to other consensus algorithms in order to get, for instance, expected reward or expected block validation time.

PLP can also be used to model the behaviour of consensus algorithms in presence of Byzantine faults.