D-RAM Distribution: A Popular Energy-Saving Memory Mining Blockchain Technology

Jerry Jing, Michael Freeman, David Stewart
with Wallace

Jan 7th, 2022

Abstract

Cryptocurrencies have been a massive surge since last decade leads to blockchain technology steps in epical milestone. Blockchain is a distributed database that is shared among the nodes of a computer network. Bitcoin, as a critical role in cryptocurrency system is known as a typical blockchain, maintaining a secure and decentralized record of transactions. The value and advantage of blockchain technology has been acknowledged by the public and applied in difference areas. The defect of blockchain technology is gradually revealed. Excessively wasting electricity have severely affect the utilities usage in many countries. Therefore, DRD Memory Mining is developed to solve the defects. This article illustrates the algorithm of DRD Memory Mining and how the memory consensus mechanism works.

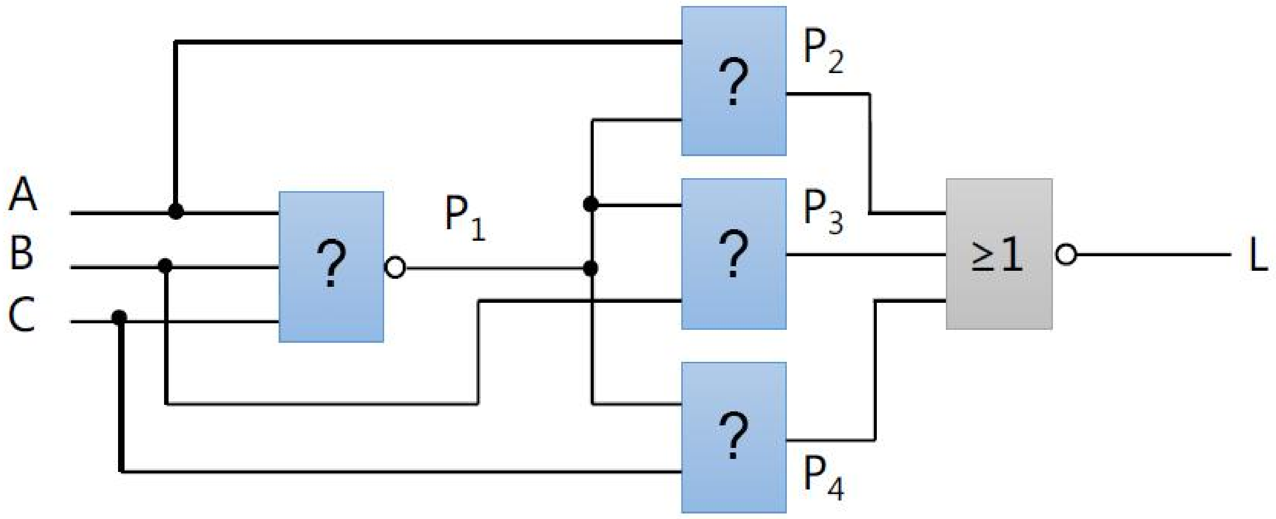
1 Introduction

The innovation with Bitcoin is that it guarantees the fidelity and security of a record of data and generates trust without the need for a trusted third party. With Bitcoin price surges higher, Bitcoin mining activities become more popular in many countries. It creates huge value and also bring negative effects. The Bitcoin network's spiraling energy needs are truly staggering compared with other potential needs. The production of just one Bitcoin consumes as much energy as 18 Americans or more than 1500 Nigerians per year. With less Bitcoin remaining, more electricity is needed to improve computing power. DRD creates a scenario that why not using RAM, an energy-saving hardware instead of CPU to validate POW. Therefore D-RAM Distribution technology has been established in Silicon Valley, a memory mining blockchain technology has been developed to provide an innovative, energy-saving mining era. The main concept in POW of DRD memory mining is to create a difficult NPC problem calculated in RAM instead of CPU or other computing power hardware.

2 DRD POW Algorithm

The Proof of work(POW) of DRD is validated by solving a completed path in the random logic gate circuit. Random logic gate circuit is an Non-deterministic Polynomial(NPC) problem. For example the polynomial listed below: $Y = X^3 + 2x + 1$. If we know the value of X it will be very easy to get the value of Y. However if we only know the value of Y, it will be difficult to get the value of X. The only solution to get X is by guessing all the possibilities and validate in the polynomial to verify the value of Y is true or not. This method called exhaustive method.

We assume that there are millions of logic gate as below and we know the output Y but we have to find out the correct input X to get the right path from input to output. For example the simple logic circuit like below: if we know the value of L we need to calculate the value of A,B,C. In this case we have to list all the possibility such as what kind of logic gate(AND,OR,NOT,XOR etc) and also in each logic gate situation list all the possibilities of the value of A,B,C to validate output L is true or false. We realized that NPC problem cannot be optimized to find out answer easily. The answer maybe one or more than one but we need to list all possibilities and validates one by one. Simulates the logic gate circuit as below:



Initialization:
Within the data de-serialization requirement we have implemented the interface signals, which are the following signals: clock, reset, busWrite, busDataToLmac, busDataFromLmac, busAddr, pushSym, Sym, pushOut, dataOut, macState. The MAC state machine was also implemented that operates in 4 states, that are “Offline”, “Searching”, “Associating”, and “Working”. The MAC state machine is in the “Offline” state after a reset, after a error, or when the enable bit is 0. The state machine is in the “Searching” state when the enable bit is 1 and is actively looking for 64 framing nibbles in 6500 clocks. The state machine is in the “Associating” state until a type 3 packet is received. The state machine is in the “Working” state following the receipt of a type 3 packet and until the next reset or the enable signal is brought low.

```
timescale 1ns/10ps

module
lwmac(clk,reset,busWrite,busDataToLmac,busDataFromLmac,busAddr,pushSym,Sym,push
Out,dataOut,macState);
input clk, reset, busWrite;
input [31:0] busDataToLmac;
output [31:0] busDataFromLmac;
input [31:0] busAddr;
input pushSym;
input [4:0] Sym;
output pushOut;
output [8:0] dataOut;
output [3:0] macState;
reg SOH;
reg [3:0] lwcmd;
reg [63:0] D_MACID;
reg [7:0] Crypto_salt;

reg [31:0] _busDataFromLmac;
reg _pushOut;
reg [3:0] D_reg;
reg [8:0] _dataOut;
reg [3:0] _macState;
reg [3:0] next_state;
reg [12:0] clockcnt;
reg [6:0] frame_nib_cnt;
reg [4:0] packet_state;
reg [16:0] debugcnt;
reg [31:0]
MAClow,MAChigh,key0,key1,key2,key3,key4,key5,key6,key7,key8,key9,key10,key11,ke
y12,key13,key14,key15,enable,PktMask;

integer j,k,l;
reg [63:0]dest_id,source_id;
reg [67:0]_source_id;
reg [3:0] _Sym;
reg [4:0] D_length,S_length;
reg work_flag;
integer length, hlength;
reg [67:0] _id;
reg [63:0] id;
reg [15:0] hhash;

assign busDataFromLmac = _busDataFromLmac;
assign pushOut = _pushOut;
assign dataOut = _dataOut;
assign macState = _macState;
assign _Sym= Sym;

parameter [3:0] OFFLINE = 4'b0000;
parameter [3:0] SEARCHING = 4'b0001;
parameter [3:0] ASSOCIATING = 4'b0010;
parameter [3:0] WORKING = 4'b0011;
```

Within the Hash Checking requirement we have implemented the header hash and payload hash functional algorithm computes the hash checking of the header and payload and compares it against the provided header hash and payload hash from the input symbol signal. The hash checking is used to ensure the data sent to the head end comes from a valid user and to ensure the data has not been corrupted during the transmission process. The header hash is implemented using the hash checking equation, 16’h1021, and the hash is initialized to 16’h58a9. The payload hash is implemented in a similar manner as the header hash except through the use of different hash checking equations.

```
task extract_id;
@(Sym)
 begin
  length= _Sym[3:0]+1'b1;
  $display("length: %d",length);

  for(k=0;k<length+1'b1;k=k+1)begin
     _id[67:64]= _Sym[ 3:0];
     _id[67: 0]= #5 (_id[67:0]>>4);
     $display("_id:%h k:%d",_id,k);
  end
    if(k == length+1 || k == length+2)
      begin
      if(k == length+1)
         begin
           Crypto_salt[3:0] <= (_Sym[3:0]);
           $display("k val1:%d",k);
           k = k + 1;
           end
        if(k == length+2)
         begin
           Crypto_salt[7:4] <= (_Sym[3:0]);
           $display("k val2:%d",k);
           end
      end
  if(packet_state== 5'b00001)begin
     dest_id[63:0]=_id[63:0];
     $display("dest_id is %h",dest_id);
  end
  else if(packet_state== 5'b00010)begin
     source_id[63:0]=_id[63:0];
     $display("source_id is %h",source_id);
  end
  assign _id=68'b0;
 end
Endtask

task extract_hhash;
//output [63:0] id;
//begin
@(Sym)
 begin
  hlength= 4;
  $display("hlength: %d",hlength);

  for(l=0;l<hlength;l=l+1)begin
     hhash[15:12]= _Sym[3:0];
     if(l < 3)
      begin
```

```
        hhash[15: 0]= #5 (hhash[15:0]>>4);
        end
     $display("_id:%h l:%d",hhash,l);
  end
 end
//end
endtask
```

Within the Parsing of the low level header requirement we have implemented the low level packet type (lwcmd), destination MAC address, source MAC address, crypto information, and payload data. The parsing of the Low Level Header was performed within the low level MAC module. Data received from the input symbol signal is stored and read from registers during processing of the header and payload data. The destination MAC address, source MAC address, payload length, and payload functions were coded to account for variable length.

```
always @(Sym && work_flag==1)
//always @((Sym) && (work_flag))
begin
case(packet_state)
5'b00000: if (SOH==0)  begin
          packet_state <= 0;end
        else begin
          $display("packet_state: %h",packet_state);
          packet_state <= 5'b00001;
        end

5'b00001: if (Sym[4]==1)  begin
          packet_state <= 5'b00001;
        end
        else
          if (Sym[4]==0)  begin
          $display("packet_state: %h",packet_state);

          lwcmd[3:0] =  _Sym[3:0];/// ERROR!!
          $display("lwcmd: %2h",lwcmd[3:0]);
          packet_state <= 5'b00010;
        end

5'b00010: if (Sym[4]==1)  begin
            packet_state <= 5'b00010;end
        else begin
//        $display("packet_state: %h",packet_state);
           extract_id(); //source

/*
            length= _Sym[3:0]+1'b1;
             $display("length: %d",length);

              for(k=0;k<length+1'b1;k=k+1)begin
                _id[67:64]= _Sym[ 3:0];
                _id[67: 0]= #5 (_id[67:0]>>4);
                 $display("_id:%h k:%d",_id,k);
               end
                dest_id[63:0]=_id[63:0];
               $display("dest_id is %h",dest_id);
//            assign _id=68'b0;
```

```
*/
            packet_state <= 5'b00011;
        end

5'b00011: if (Sym[4]==1) begin
            packet_state <= 5'b00011;end
        else begin
            $display("test");
            extract_hhash(); //header hash
          //$ extract_id();display("packet_state: %h",packet_state);
          //extract_id();
/*
             length= _Sym[3:0]+1'b1;
             $display("length: %d",length);

              for(k=0;k<length+1'b1;k=k+1)begin
                _id[67:64]= _Sym[ 3:0];
                _id[67: 0]= #5 (_id[67:0]>>4);
                 $display("_id:%h k:%d",_id,k);
               end
                 source_id[63:0]=_id[63:0];
               $display("source_id is %h",source_id);
*/
//            assign _id=68'b0;

            packet_state <= 5'b00100;
        End
5'b00100:if (Sym[4]==1) begin
            packet_state <= 5'b00100;end
       else begin
         /*else if length begin
            Crypto_salt[3:0] <= Sym[3:0];
          $display("crypto_salt: %h",Crypto_salt);
      */
            packet_state <= 5'b00101;
         end
5'b00101:if (Sym[4]==1) begin
            packet_state <= 5'b00101;end
         else begin
            Crypto_salt[7:4] <= Sym[3:0];
            packet_state <= 5'b00110;
         end
```

In the actual situation, the server sends a random logic gate question to the miner when it starts mining. Miner starts calculating and send the answer back to server to validate. When the answer is true the miner will be rewarded. The amount of rewarded DRD token daily is determined by the percentage of total valid proof associated with address in the whole network. The rewards may not be the same amount at the same time due to the calculation time.

3 DRD Consensus Algorithm

3.1 Overview

Initialization: $\Lambda \longleftarrow$ Hash.Init(λ,{N,N_input,S,P,H,φ}) Number of DAG nodes N, number of input nodes Ninput, output binary bits Noutput, random seed S. P is a pseudo-random number generating function （pseudo-random number generator). P(S, i) indicates when S is the random seed, the ith pseudo-random number generated. Auxiliary hash function H and its parameters φ, H_n(data, φ) indicates that the output length of H is set to n, other parameters are set to φ, put in the output of the function when the setting is "data".

DAG Initial steps : P_i=P(S,i)
DAG G ’definition:G=<V,E>,|V|=N
E_i,j={(1i>N_inputand(P_imodi=j)@0i≤N_input)┤
it is easy to know: ∑E=N−N_input

3.2 Calculation and Validation Stage

Calculation Stage
Definition:D=H_N_input(B,φ),B is the input data in binary format.
Define node status A
A={(D_ii≤N_input@A_jandA_i−1i>N_input,P_N+1+imod2=0,E_i,j=1@A_jorA_i−1i>N_input,P_N+1+imod2=0,E_i, j=1)
D_i represents D’s the ith binary digit
Define final status F
F_i={(F_ii≤N_inputor(i>N_input,P_2N+1+imod2=0)@notF_ii>N_input,P_2N+1+imod2=1)
Output as H_N_output(F,φ)

Validation Stage
The validation sequence is defined as follows:
V_i=P_2^i−1,0<i<Log_2N+1
V_N+i=F_2^i−1,0<i<Log_2N+1
In detail, we use the variant of the Mason rotation algorithm SIMD-oriented Fast Mersenne Twister (SFMT) as P,SHA algorithm as H.
In the proof step, generate and traverse sequences P, we only need to store the current generation result and generator status each time. When a checkpoint is encountered (that is, a point with an integer power of subscript 2), comparing with the verification sequence, returns NO immediately when a mismatched element is encountered. The time complexity of this operation is O(N), when verify V_i,i∈[0,N), the space complexity is O(1)， when verify V_N+i, i∈[0,N)， space complexity is O(N).
It is easy to know that for an initialized hash function Λ , the same input data can get the same output data.

3.3 Characteristics of Framework Consensus Algorithm

Unidirectional: The steps of the whole algorithm can be regarded as DAG (external hash function). Two external hash functions ensure that when the output is known, non polynomial time is required to get the input. A sub-problem of the inverse solution DAG part is to know the state of the last node, finding a feasible state of the whole network. This is the classical Boolean satisfiability (SAT) problem. Because the connection is random,it can cover the entire SAT domain,but the SAT problem is NPC problem.It cannot be solved in polynomial time. Therefore, the three steps of inverse solution of this function need to be solved by non polynomial algorithm, and the whole problem can not be solved in polynomial time.

Randomness: According to the definition of F sequence, regardless of the input, P (F_i = 1) = 50% and each F_i relatively independent. For any binary sequence, there is a random number sequence that can generate the sequence. Therefore, for any input, the output of DAG is a random value in [0,2 ^ noutput-1].

Collision resistance: According to the randomness, the collision probability of any two sequences is 1/2 ^ noutput.

Anti parallelism: The problem of the network is to select a random seed K so that the graph generated by the seed meets specific conditions after the above calculation (for example, the first N consecutive bits are 0). Under this problem, it is difficult for different processors to share the same DAG, so it is difficult to share memory between problems/proofs. A complete set of calculation and fast access resources are required for parallelism. Since each calculation needs the last result, it is difficult to calculate in parallel.

Conclusion: DRD Consensus Algorithm is to build a NPC problem that could not be optimized to calculate. And this NPC problem is random logic gate as we explained above. Each miner in DRD network reaches a consensus agreement that who solves the problem first is trusted and rewarded.

4. DRD Features & Prospects

4.1 Overview

The traditional miner is expensive and wasting electricity. Electricity resource should not be considered as a necessary mining investment, POW process should not waste resource to achieve their goals. Therefore, DRD memory mining adopts memory stick as hardware device, a much cheaper hardware with low power consumption.

4.2 DRD Halving Model

The total amount of DRD token is 21 Million. With respect to BTC halving model, we proposed an advanced high-frequency halving model in order to ensure the value of DRD token. BTC halves every 4 years however DRD halves in 7.5 min. Simultaneously, each halving action leads to a new block generated and the height (H) of block add one more layer. According to this halving model, the function between daily quantity(Q) of DRD and the height(H) of the block explained as below:

$$Q = -2.12 * 10^{7} * e^{(-5*10^{-6*H})} - (-2.12 * 10^{7} * e^{(-5*10^{-6*0})})$$

$$H = (24 * 600 / 7.5) * Day$$

Based on the equation we can know that the daily quantity of DRD is decreasing everyday. With such a frequent halving model, will greatly increase DRD value and its appreciation.

4.3 Affordable & Energy-Saving

Bitcoin miner requires high demand hardware device. As the popularity of Bitcoin mining grows, the price of Bitcoin miner is rapid rising. RAM is a much cheaper material as DRD miner hardware support. Make sure that every DRD user can afford it to enjoy the mining experience.

An energy-saving is another feature that DRD owns. According to our precise test results prove that daily power consumption of RAM is less than 0.05kwh, which equals to a router. Bitcoin miner daily power consumption is 78kwh per day, and the daily utility cost is 78kwh*0.07usd/kwh = 5.46 USD/day. DRD daily utility cost is only 0.007USD.

4.4 DRD Road-Map & Prospects

- The first generation of DRD miner will be manufactured in December 2021. To ensure the DRD quality we will test first 100 units.
- According to the feature of DRD blockchain technology, our goal is to step in the top crypto exchanges within 6 month to enhance the DRD liquidity and development.
- DRD team will develop branched chain based on DRD public chain, Such as Metaverse, Digital Assets, NFT and application in the field of plastic surgery and hospital, etc.
- DRD team will be committed to optimize the algorithm and upgrade the hardware to improve DRD miner quality into Mass-Production.
- DRD team has strong resources to promote DRD Memory Mining technology globally and become the Top 20 blockchain technology within one year.

References

[1] Richard Craib,Georey Bradway, Xander Dunn with Joey Krug. Numeraire: A Cryptographic Token for Coordinating Machine Intelligence and Preventing Overfitting. https://numer.ai/whitepaper.pdf

[2] L.M.Bach, B.Milhaijevic, M.Zagar https://ieeexplore.ieee.org/abstract/document/8400278/

[3] V. Buterin, “On public and private blockchains”, Ethereum., vol.7, Aug, 2015.

[4] C.Natoli and V. Gramoli, “The blockchain anomaly”, Proc, 15th IEEE int.Symp, pp. 310-317, 2016.

[5] G. Karame and E. Androulaki, Bitcoin and Blockchain Security, Norwood, MA: Artech House, 2016

[6] S.Nakamoto, Bitcoin: A peer-to-peer Electronic Cash System https://bitcoin.org/bitcoin.pdf