# DYNAMIC DERIVATIVE BLOCK CHAIN TECHNIQUE

A THESIS
SUBMITTED IN PARTIAL FULFILLMENT OF THE RQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

**MASTER OF TECHNOLOGY
IN
INFORMATION SYSTEMS**

Submitted By:
**Nishant Garg**

**(2K17/ISY/12)**

Under the supervision
Of
**DR. KAPIL SHARMA (HOD)**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**Delhi Technological University**

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042.

**MAY - 2019**

# DECLARATION

I hereby certify that the work which is presented in the Thesis titled "**Dynamic Derivative Block chain Technique**" in fulfillment of the requirement for the award of the Degree of Master of Technology and submitted to the Department of Information Technology, Delhi Technological University (Formerly Delhi College of Engineering), New Delhi is an authentic record of my own work under the supervision of **Dr. Kapil Sharma (HOD, IT Department).**

The matter presented in this report has not been submitted by me for the award of any other degree of this or any other Institute/University.

**Signature**

**NISHANT GARG (2K17/ISY/12)**

# CERTIFICATE

This is to certify that Nishant Garg (2K17/ISY/12) has completed the project titled "**Dynamic Derivative Block Chain Technique**" under my supervision in partial fulfillment of the MASTER OF TECHNOLOGY degree in Information Systems at DELHI TECHNOLOGICAL UNIVERSITY.

**Signature: __________________________**

**Dr. Kapil Sharma**

**HOD**

**IT Department**

**DTU**

## ACKNOWLEDGEMENT

I am very thankful to **Dr. Kapil Sharma** (Professor, H.O.D Department of Information Technology) and all the faculty members of the Department of Information Technology of DTU. They all provided us with immense support and guidance for the project.

I would also like to express my gratitude to the university for providing us with the laboratories, infrastructure, testing facilities and environment which allowed us to work without any obstructions.

I would also like to appreciate the support provided to us by our lab assistants, seniors and our peer group who aided us with all the knowledge they had regarding various topics.

Nishant Garg

Roll No. 2K17/ISY/12

## ABSTRACT

Currently block chain is being used as the distributed ledger for various purposes. In systems that emphasize the requirement of authentication (esp. banking/crypto currencies) various techniques are used to maintain the sanity of the system. The predominant ones are (i) proof of work and (ii) proof of stake. Systems using proof of work tend to use increasingly large amounts of computing power and in-turn resources like energy and processor time. Proof of stake on the other hand uses the already present stake of miners* in the ledger.

While one can be controlled with more than half the computing power and the other can be controlled with excessive stake in the ledger. Both seem to have their disadvantages. We shall try to work on the proof of work technique to better it for application as a distributed ledger authentication system.

Since the proof of work system is the first authentication system we have come up with a few techniques to better the system. In order to work out the entire architecture of the new proposed system we have discussed in brief the structures of the newly proposed blocks and how the new methodologies will affect the block chain and the associated systems that rely on it. The applications of the proposed systems are primarily in crypto-currency and implications to such systems will be considered in detail.

# TABLE OF CONTENTS

# LIST OF FIGURES

## LIST OF SYMBOLS & ABBREVIATIONS

| | |
|---|---|
| 1. POW = Proof of work | |
| 2. POA = Proof of authority | |
| 3. POS = Proof of stake | |
| 4. PID controller = Proportional integral derivative | |
| | |

# CHAPTER 1

# INTRODUCTION

## 1.1 Basic Technology

A block chain is a digital concept to store data. This data comes in blocks, so imagine blocks of digital data. These blocks are chained together, and this makes the data immutable. When a block of data is chained to the other blocks, its data can never be changed again. It will be publicly available to anyone who wants to see it ever again, in exactly the way it was once added to the block chain. That is quite revolutionary, because it allows us to keep track records of pretty much anything we can think of (to name some: property rights, identities, money balances, medical records), without being at risk of someone tampering with those records. If I buy a house right now and add a photo of the property rights to a block chain, I will always and forever be able to prove that I owned those rights at that point. Nobody can change that information if it is put on the block chain. So, it is a way to save data and make it immutable. That sounds great, but the big question of course is: How does that work?

**Step 1—Transaction data**

Alright, let’s start off with an example: the Bit coin block chain. The Bit coin block chain is the oldest block chain in existence. The blocks on the Bit coin block chain are 1 MB of data each. At the time of writing it counts about 525,000 blocks, meaning roughly a total of 525,000 MB has been stored on this block chain. The data on the Bit coin block chain however, only exists out of transaction data in regard to Bit coin transactions. It is a giant track record of all the Bit coin transactions that have ever occurred, all the way back to the very first Bit coin transaction. This article refers to a block chain that stores transaction data.

**Step 2—Chaining the blocks (with a hash)**

Imagine a bunch of blocks of transaction data Issues with proof of work

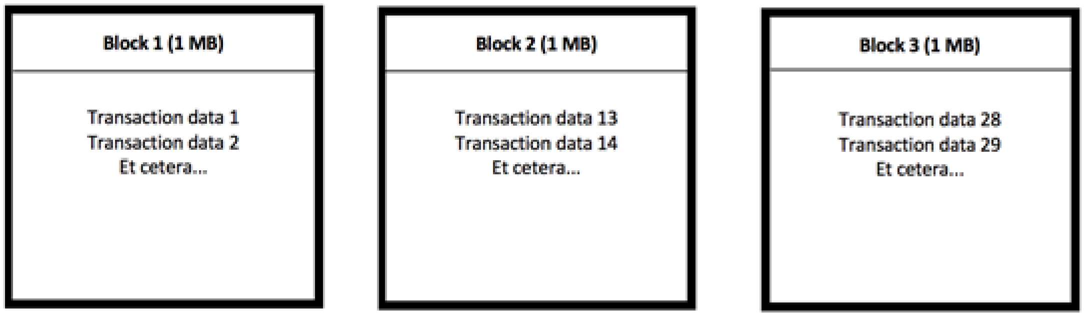


Fig. 1.1 Chaining

Not really special yet, you can compare it to some stand-alone word documents. As described in fig 1.1, Document 1 would then chronologically describe the first transactions that have occurred up to 1 MB clearly demonstrated in the , where after the next transactions would be described in document 2 up to another MB, and so on. These documents are the blocks of data. These blocks are now being linked (aka chained) together. To do this, every block gets a unique (digital) signature that corresponds to exactly the string of data in that block. If anything inside a block changes, even just a single digit change, the block will get a new signature. This happens through hashing and will be thoroughly explained in step 3.

Let's say block 1 registers two transactions, transaction 1 and transaction 2. Imagine that these transactions make up a total of 1 MB (in reality this would be much more transactions). This block of data now gets a signature for this specific string of data. Let's say the signature is 'X32'. Here is what this looks like:

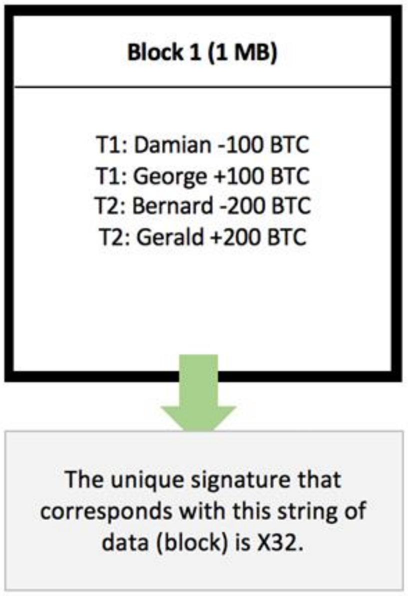


Fig. 1.2 Block

The signatures link the blocks together as shown in figure 1.2, making them a chain of blocks. Let's picture adding another block to this chain of blocks; block 3. Here is what this looks like:

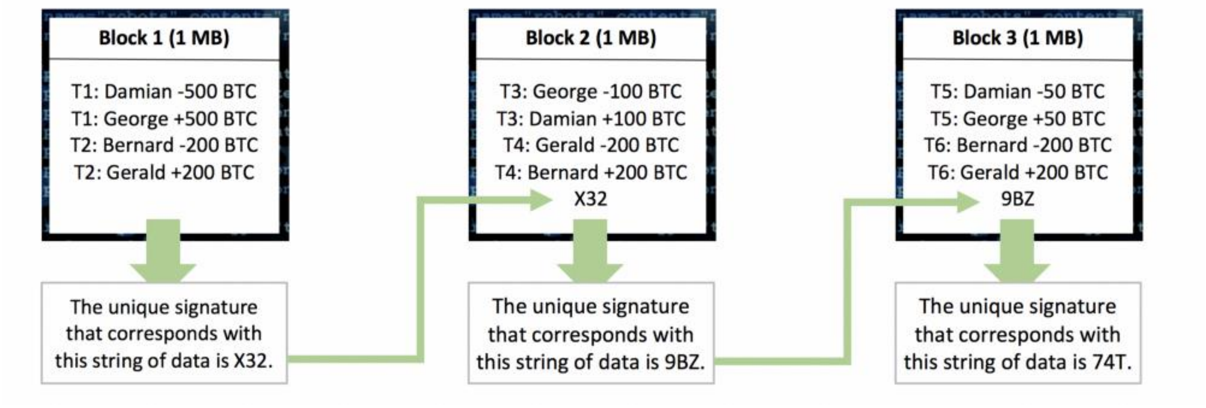


Fig. 1.3 Multiple blocks

Now imagine if the data in block 1 from fig 1.3 is altered.

Let's say that the transaction between Damian and George is altered and Damian now supposedly sent 500 Bit coin to George instead of 100 Bit coin. The string of data in block 1 is now different, meaning the block also gets a new signature. The signature that corresponds with this new set of data is no longer X32. Let's say it is now 'W10' instead. Here is what happens now:

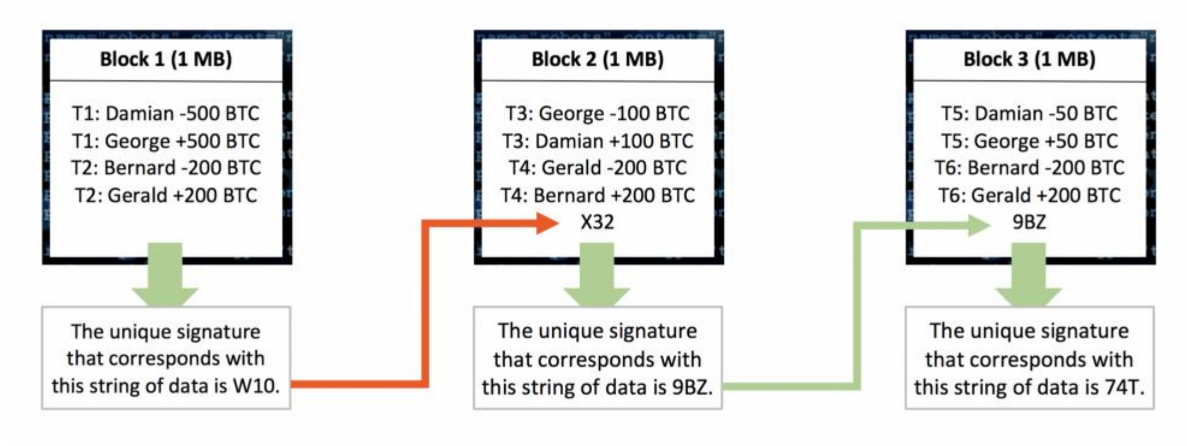


Fig. 1.4 Blocks

The signature W10 does not match the signature that was previously added to block 2 anymore. Block 1 and 2 are now no longer chained to each other as mentioned in fig 1.4. This indicates to other users of this block chain that some data in block 1 has been altered, and because the block chain should be immutable, they reject this change by shifting back to a previous record of the block chain where all the blocks are still chained together. The only way that an alteration can stay undetected, is if all the blocks stay chained together. This means for the alteration to go undetected, the new signature of block 1 must replace the old one in the data of block 2. But if the data of block 2 changes, this will cause block 2 to have a different signature as well. Let's say the signature of block 2 is now ‘PP4’ instead of 9BZ. Now block 2 and 3 are no longer chained together!

1. 51% attack.

A proof of work system is vulnerable to attack if more than half the computing power is used to develop an alternate chain. Networks like Bit coin are largely secure because they have a large number of nodes and it is practically impossible to buy 51% of bit coin’s total computing power. But networks which are relatively smaller in size can be attacked by this method.

“After Krypton, a Proof-of-work based network, was recently hacked, the Krypton development team announced its transfer to a Proof-of-stake system.”

2. Energy requirements.

As an example of energy requirements consider Bit coin. Bit coin needs 20.03 TWh energy every day. Each transaction takes on an average 221 Kwh of energy that is 221 units. The cost of mining the currency and the amount of currency obtained as a reward is always locked in a race. And this puts the value of currency in jeopardy. These heavy energy requirements are a major issue facing the current proof of make it less feasible and such calculations seem impossible for mobile equipment with limited memory and processing power.

3. Excessive Processing power requirement

The ledger can be called a completely distributed ledger only if every node can fairly participate in the ledger. Currently with a lot of competition the target values of the proof of work is rising so rapidly and essentially making certain nodes incapable of participating in the chain on their own accord. These nodes are forced to join other nodes in groups / communities or not participate.

Mining an entire block is increasingly difficult and the ledger hence tends to become centralized with power concentrating with nodes that have excessive computational power. This essentially leads to a race of purchasing computing power and cooling systems which in turn further increases energy requirements.

4. No Liquidity in Computing Power

The original motto of proof of work was one IP one vote, but the recent paradigm shift of most computing power resulting in the hard to get power to vote has reduced its distributed nature.

Since it is impossible to mine or get rewards with normal machinery for mining. It is increasingly seen that such nodes have no say in the system. People have come up with methods to mine the block chain by maintaining "mining farms" with mining machinery stacked up in rows of machines which run in parallel. Innovative methods to cool the machine are being deployed such as building the entire farms in places like Iceland!

If few of such nodes are hacked simultaneously, one can very simply generate an alternative chain which can again cause double spending problem which would make a nuisance. "In reality a 51% attack is feasible – especially with the rise of mining pools (groups of people mining together as a single unit). However the potential damage one could cause is small – though enough that it causes a panic that would seriously threaten bit coin's use as a currency. At current network mining difficulty levels, not even large-scale governments could easily mount a 51% attack."

# CHAPTER 2

# BACKGROUND WORK

The present alternatives to proof of work include proof-of-stake, proof of activity, proof of capacity, proof of burn, proof of storage etc.

The most common alternative is proof of stake. The present stakeholders set aside some amount of their stake (coins) to become a part of the mining pool and blocks are allotted to the stakeholder from the pool. If the miner is found to tamper with the chain, then the miner loses the stake which he froze for his chance to mine.

This alternative is excellent as it doesn’t require excess energy for mining. Nodes can simply set aside their coins for the mining process. More complex calculations can be avoided.

Most importantly it has a defense mechanism against the 51% attack. When a node tries to purchase 51% of the stake in such a system then the price of the coins will rise rapidly as a result of a jerk in demand.

Its major drawback is that nodes with greater stake in the ledger get to mine a greater share of the chain. Hence all the power in the chain gets centralized. This defeats the idea of a distributed network and leaves all the major decisions in the hands of people with the greatest share.

All of the above clearly point out to a clear lack of alternative in the true sense for the proof of work strategy. Since the major problem in the present strategy is just the race problem making the targets escalate and increase energy requirements and hence pooling of resources and mining farms, we must find ways to make the strategy more available to all nodes. In order to maintain the security of the block chain we must allow for the complex problems (facilitated mostly by harder targets), in order to make sure that no node can catch up with an alternate chain.

To incorporate both the above we must take more than one proofs for one block. This concept itself introduces numerous problems like the requirement of one node to collect all the proofs of works. This means we may need one central authority which itself defeats the entire block chain's concept. We also need to have a node remove duplicate records of proofs which have already been calculated.

The second issue which we faced was we now needed a method to obtain the dynamic values of the values of "n" (number of proofs accepted) and "t" (target). Previously in the white paper of Satoshi a simple p controller was used to make the value of target in order to maintain a fixed rate. Now the value of n may change dynamically (to encourage a faster rate we may need to reduce the number of miners to increase the rewards awarded).

Just changes in n may reduce the number of miners attempting at any time and hence affect the rate or it may not, in which case we need to vary the target value.

Another issue which we ran into was the fact that we may need to reward different number of miners which in many cases may make exceed the block size. Bumping the transactions to the next block can be a possible solution but it essentially eats up an awful lot of space inside a block and makes more frequent the ledger updates. Also the fact that any given miner would be totally unaware of the other miners it is impossible for him/her to add them all up.

A final issue that we came across was the fact that all the miner data had to be submitted and hence also could be stolen so we needed a second timestamp for all the proofs for each and every block.

Consequently our first attempt to make an algorithm was to keep a timestamp which tied all the proofs to their respective pay-to addresses. This timestamp would take up all the proofs which were valid and store them in a set, bind them with a timestamp and a pay-to address. The timestamp would simply serve the purpose of finding which proof ended up earlier. This method is efficient if we can guarantee a minimum distance of each node to a timestamp/ collection server. This server would generate the block once it gets a number of blocks that suffice.

A problem with such a setting is that we do not know for sure if certain number of proofs even exists for any given problem. This problem could essentially bring the entire work to a halt. Another server after a timestamp server for the entire chain and also the use of Use Nets to find transactions introduces unwanted latency in the system.

Another alternative which we worked out was each block could be mined after the miner address would be appended to it. This way we can be sure of the fact that we have enough number of proofs for each unique string generated by the combination of a different miner address. Also this can be done entirely without a collection server where the miner can release the standalone proofs which are useless with a different pay-to address. A final proof of work miner would definitely try to be the first to finish the mining process and hence would take up all the available proofs in the system or someone else would beat him to the race and finish himself.

This is the strategy of our choice and we shall be discussing the details of above in detail.

# CHAPTER 3

# ARCHITECTURE AND ALGORITHM

## 3.1 Algorithm for Dynamic decision of N and T

Dynamic decision of n and t can be made based on the rate of production of the blocks. In the original bit coin paper by Satoshi Nakamoto, a simple rate controller was used with only a term proportional to the error in rate. The rate required is 2016 blocks in 2 weeks. This totally amounts to a value of 6 blocks in an hour. The most general trend is to see an increase in the rate which is reduced by increasing the target value. The algorithm we deploy for this application is a PID controller.

"PID controllers are found in a wide range of applications for industrial process control. Approximately 95% of the closed loop operations of industrial automation sector use PID controllers. PID stands for Proportional-Integral-Derivative. These three controllers are combined in such a way that it produces a control signal as described in figure 3.1.1.

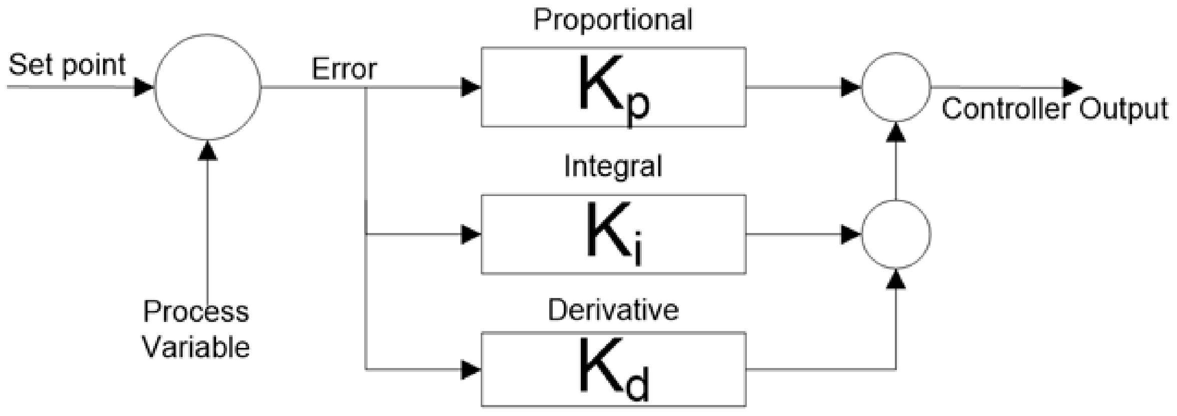


Fig. 3.1 Flow

As a feedback controller, it delivers the control output at desired levels. Before microprocessors were invented, PID control was implemented by the analog electronic components. But today all PID controllers are processed by the micro-processors. Programmable logic controllers also have the inbuilt PID controller instructions. Due to the flexibility and reliability of the PID controllers, these are traditionally used in process control applications."

In our implementation we did not use the I controller of the PID, since the integral value would continue to grow on and our error should settle near zero irrespective of previous error.

So we fix up values of the Kp and Kd (the co-efficient of the proportional part and differential part respectively) as values which would essentially not cause a lot of oscillations.

We obviously lack any definitive way to value out the value of rate as a random variable. We also do not have any historic data of bit coin itself. So we picked out 3 variable functions to map out the results of our PID algorithm. There is obviously no way to check the efficiency of this algorithm but we can check out the mappings by the functions for various types of random variable distributions.

## 3.2 Algorithm and Architecture of the new strategy

This algorithm essentially changed the entire structure of the block. Now each block holds its current n (number of proofs accepted) and t (target value for each block) and followed by n number of proofs separated by hashes (or preferably by some bit stuffed break sequence) etc. We are elaborating on the algorithm where we append the pay-to address of the miner to the block and then find a proof of work for the resulting string.

Each of the miners does the same with the one block in consideration. Since each miner wishes to be included in the "n" miners who are rewarded in the process. We can be sure that if a certain miner has made a correct calculation in time he/she is definitely rewarded. Clearly all the subsequent miners would use all the available proofs and it would be clearly uneconomical to hold a prejudice against one such fellow miner.

In order to support the above functionality we need a new list of proof of work block structure and a possibly a new architecture for the system.

The architecture can be in two different types:

1. With a Collection server.

2. With a Contention phase.

When using a separate collection server.

A plausible issue which we must consider is the fact that the last few entries for the proof of work may be under fire since the order of entries would be dependant of the amount of time it takes to reach the timestamp server. This is also a problem in the actual bit coin and hence we must have more than one such collection servers. The server will simply receive the proofs till it has "n" number of proofs.

Once it reaches a limit of n correct proofs it will quit the whole process and publish the block with the respective proofs. Obviously the entire server can be used to lay restrictions on the number of proofs of varying difficulties (we could easily have proof of works for different target values). Also we could encourage a diverse geographic distribution of the proofs in some other applications which may need to support such functionalities.

When using a contention phase in this architecture we are not introducing a lot of major changes like a new server running between the timestamps of the chain but here again we are having a different packet structure and some changes in the way it is mined. Now the blocks will enter into a contention phase where the blocks will continually get proofs from various nodes.

Each node will try to append to the existing block by adding a proof of work. Now let us consider that 3 miners A, B and C are mining the block and both of A and B publish their proofs simultaneously. In this case the remaining number of proofs required for both of these is say k. If the proof by B is ignored for new 10 times then B can publish its proof again at the end of the new chain.

This time say C also simultaneously publishes both chain are of length k + 11 if C ignores the old proof by B but if he/she doesn't then we have C's chain as k+12 and it already includes the proof by B so both win. If not the possibility of B's proof being ignored for n consecutive tries is (1/2)**n which is quite clearly very slim.

This method requires the block to at least have all the proof and pay to address pairs separated by some escape flags or escape characters which are stuffed when published as a block chain and de-stuffed when being used by the miner.

This method doesn't change the existing architecture by a lot but it reduces the amount of prize received by a miner by a factor of n and hence highly discouraging the processor speed race which makes it ridiculously tough for node with moderate computing powers to calculate.

## 3.3 Flowchart for Proposed

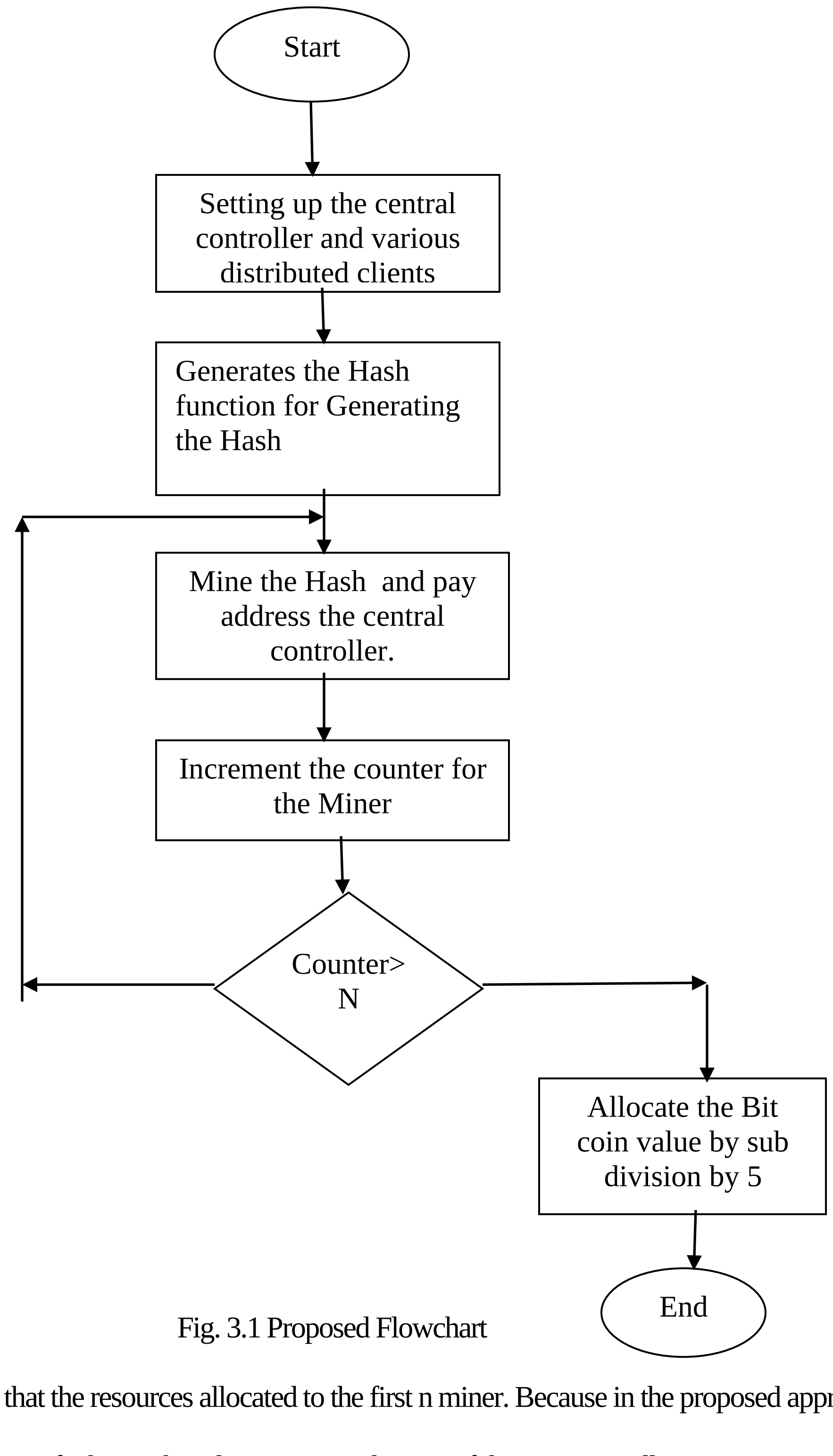

Start
Setting up the central controller and various distributed clients
Generates the Hash function for Generating the Hash
Mine the Hash and pay address the central controller.
Increment the counter for the Miner
Counter> N
Allocate the Bit coin value by sub division by 5
End

Fig. 3.1 Proposed Flowchart

In the Fig. 3.1 it is clear that the resources allocated to the first n miner. Because in the proposed approach the success rate per miner is far better than the existing technique of the resources allocation.

## 3.4 Flowchart for the existing

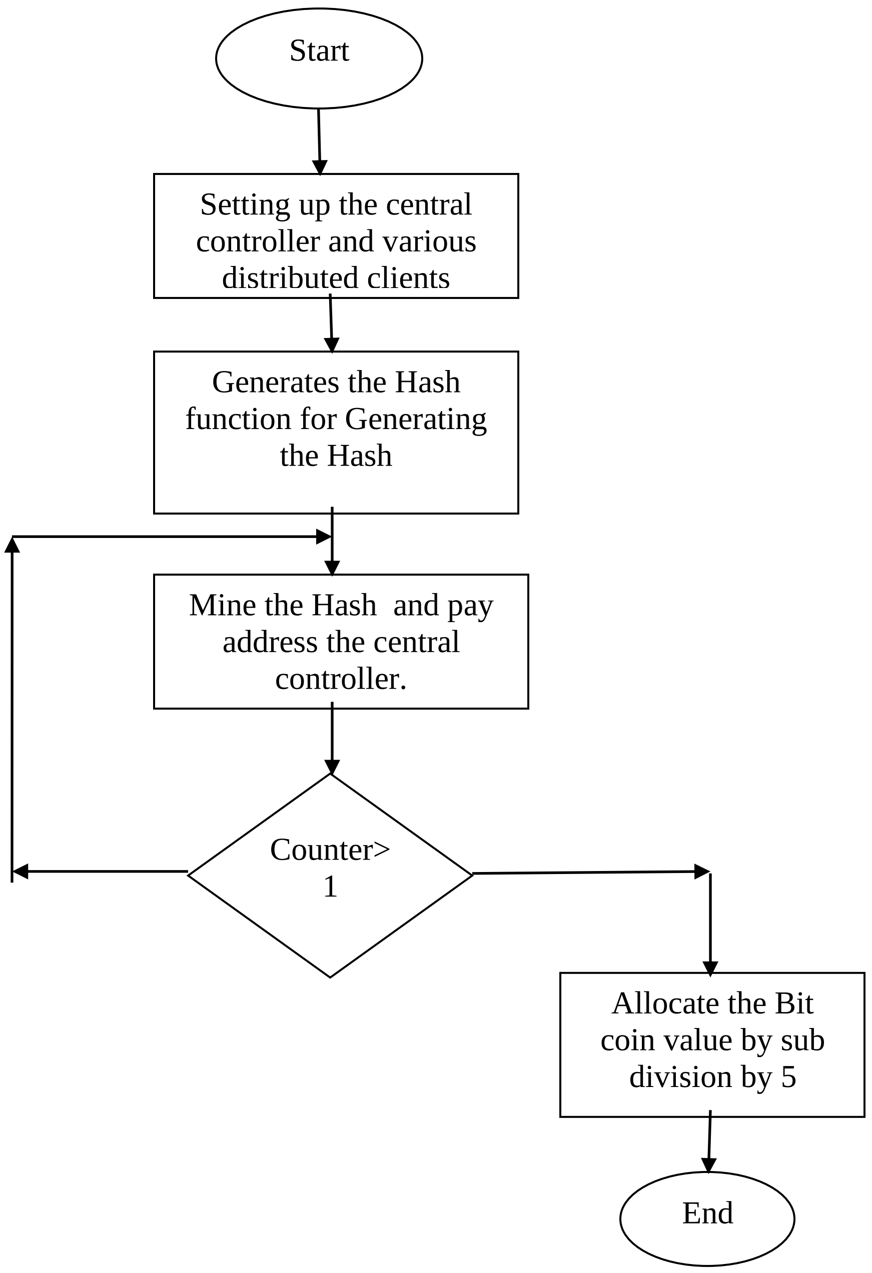

Start
Setting up the central controller and various distributed clients
Generates the Hash function for Generating the Hash
Mine the Hash and pay address the central controller.
Counter> 1
Allocate the Bit coin value by sub division by 5
End

Fig. 3.2 Existing Flowchart

Fig. 3.1 shows the existing technique for the bit coin mining. In the existing approach the first ever miner will be allocate with all the resources. Whoever will be submitting the has on second and third number will be neglected. So the effort per user will be reduced tremendously.

## 3.5 Performance parameters

There are various performance parameters on the basis of which the performance of the existing and proposed technique can be compared.

a. Throughput
b. Energy consumption.

# CHAPTER 4

# CODE AND SNAPSHOTS

## 4.1 Block Structure

```
function Hash = DataHash(Data, varargin)
if nargin == 0
   R = Version_L;
   if nargout == 0
      disp(R);
   else
      Hash = R;
   end

   return;
end

% Parse inputs: ---------------------------------------------------------------
[Method, OutFormat, isFile, isBin, Data] = ParseInput(Data, varargin{:});
% Create the engine: ----------------------------------------------------------
try
   Engine = java.security.MessageDigest.getInstance(Method);
catch ME  % Handle errors during initializing the engine:
   if ~usejava('jvm')
      Error_L('needJava', 'DataHash needs Java.');
   end
   Error_L('BadInput2', 'Invalid hashing algorithm: [%s]. %s', ...
      Method, ME.message);
end
% Create the hash value: ------------------------------------------------------
if isFile
   [FID, Msg] = fopen(Data, 'r');        % Open the file
   if FID < 0
```

```
    Error_L('BadFile', ['Cannot open file: %s', char(10), '%s'], Data, Msg);
  end
  % Read file in chunks to save memory and Java heap space:
  Chunk = 1e6;          % Fastest for 1e6 on Win7/64, HDD
  Count = Chunk;        % Dummy value to satisfy WHILE condition
  while Count == Chunk
    [Data, Count] = fread(FID, Chunk, '*uint8');
    if Count ~= 0      % Avoid error for empty file
      Engine.update(Data);
    end
  end
  fclose(FID);

elseif isBin            % Contents of an elementary array, type tested already:
  if ~isempty(Data)     % Engine.update fails for empty input!
    if isnumeric(Data)
      if isreal(Data)
        Engine.update(typecast(Data(:), 'uint8'));
      else
        Engine.update(typecast(real(Data(:)), 'uint8'));
        Engine.update(typecast(imag(Data(:)), 'uint8'));
      end
    elseif islogical(Data)               % TYPECAST cannot handle LOGICAL
      Engine.update(typecast(uint8(Data(:)), 'uint8'));
    elseif ischar(Data)                  % TYPECAST cannot handle CHAR
      Engine.update(typecast(uint16(Data(:)), 'uint8'));
      % Bugfix: Line removed
    elseif myIsString(Data)
      if isscalar(Data)
        Engine.update(typecast(uint16(Data{1}), 'uint8'));
      else
        Error_L('BadBinData', 'Bin type requires scalar string.');
      end
    else  % This should have been caught above!
```

```
      Error_L('BadBinData', 'Data type not handled: %s', class(Data));
    end
  end
else                 % Array with type:
  Engine = CoreHash(Data, Engine);
end

% Calculate the hash: -------------------------------------------------------
Hash = typecast(Engine.digest, 'uint8');

% Convert hash specific output format: -----------------------------------------
switch OutFormat
  case 'hex'
    Hash = sprintf('%.2x', double(Hash));
  case 'HEX'
    Hash = sprintf('%.2X', double(Hash));
  case 'double'
    Hash = double(reshape(Hash, 1, []));
  case 'uint8'
    Hash = reshape(Hash, 1, []);
  case 'short'
    Hash = fBase64_enc(double(Hash), 0);
  case 'base64'
    Hash = fBase64_enc(double(Hash), 1);

  otherwise
    Error_L('BadOutFormat', ...
      '[Opt.Format] must be: HEX, hex, uint8, double, base64.');
end

end

%
*******************************************************************************
function Engine = CoreHash(Data, Engine)
```

```
% Consider the type and dimensions of the array to distinguish arrays with the
% same data, but different shape: [0 x 0] and [0 x 1], [1,2] and [1;2],
% DOUBLE(0) and SINGLE([0,0]):
% <  v016: [class, size, data]. BUG! 0 and zeros(1,1,0) had the same hash!
% >= v016: [class, ndims, size, data]
Engine.update([uint8(class(Data)), ...
         typecast(uint64([ndims(Data), size(Data)]), 'uint8')]);

if issparse(Data)              % Sparse arrays to struct:
  [S.Index1, S.Index2, S.Value] = find(Data);
  Engine                = CoreHash(S, Engine);
elseif isstruct(Data)          % Hash for all array elements and fields:
  F = sort(fieldnames(Data));     % Ignore order of fields
  for iField = 1:length(F)       % Loop over fields
    aField = F{iField};
    Engine.update(uint8(aField));
    for iS = 1:numel(Data)       % Loop over elements of struct array
      Engine = CoreHash(Data(iS).(aField), Engine);
    end
  end
elseif iscell(Data)            % Get hash for all cell elements:
  for iS = 1:numel(Data)
    Engine = CoreHash(Data{iS}, Engine);
  end
elseif isempty(Data)           % Nothing to do
elseif isnumeric(Data)
  if isreal(Data)
    Engine.update(typecast(Data(:), 'uint8'));
  else
    Engine.update(typecast(real(Data(:)), 'uint8'));
    Engine.update(typecast(imag(Data(:)), 'uint8'));
  end
elseif islogical(Data)         % TYPECAST cannot handle LOGICAL
```

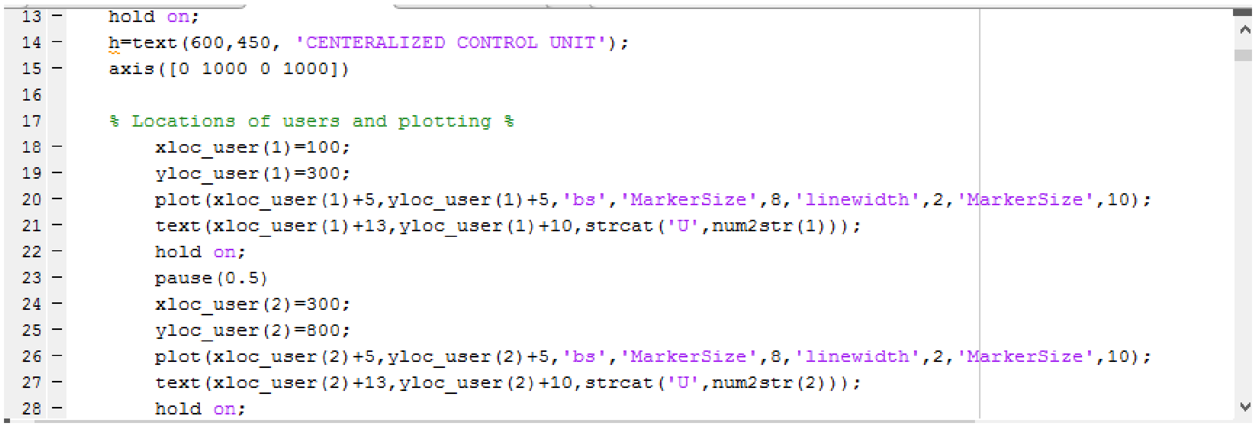
```
  Engine.update(typecast(uint8(Data(:)), 'uint8'));
elseif ischar(Data)                  % TYPECAST cannot handle CHAR
  Engine.update(typecast(uint16(Data(:)), 'uint8'));
elseif myIsString(Data)              % [19-May-2018] String class in >= R2016b
  classUint8 = uint8([117, 105, 110, 116, 49, 54]);  % 'uint16'
  for iS = 1:numel(Data)
    % Emulate without recursion: Engine = CoreHash(uint16(Data{iS}), Engine)
    aString = uint16(Data{iS});
    Engine.update([classUint8, ...
      typecast(uint64([ndims(aString), size(aString)]), 'uint8')]);
    if ~isempty(aString)
      Engine.update(typecast(uint16(aString), 'uint8'));
    end
  end

elseif isa(Data, 'function_handle')
  Engine = CoreHash(ConvertFuncHandle(Data), Engine);
elseif (isobject(Data) || isjava(Data)) && ismethod(class(Data), 'hashCode')
  Engine = CoreHash(char(Data.hashCode), Engine);
else  % Most likely a user-defined object:
  try
    BasicData = ConvertObject(Data);
  catch ME
    error(['JSimon:', mfilename, ':BadDataType'], ...
      '%s: Cannot create elementary array for type: %s\n  %s', ...
      mfilename, class(Data), ME.message);
  end

  try
    Engine = CoreHash(BasicData, Engine);
  catch ME
    if strcmpi(ME.identifier, 'MATLAB:recursionLimit')
      ME = MException(['JSimon:', mfilename, ':RecursiveType'], ...
        '%s: Cannot create hash for recursive data type: %s', ...
        mfilename, class(Data));
    end
    throw(ME);
  end
end
end
%
*****************************************************************************
function [Method, OutFormat, isFile, isBin, Data] = ParseInput(Data, varargin)
% Default options: ------------------------------------------------------
Method    = 'MD5';
OutFormat = 'hex';
isFile    = false;
isBin     = false;
% Check number and type of inputs: ---------------------------------------
nOpt = nargin - 1;
Opt  = varargin;
if nOpt == 1 && isa(Opt{1}, 'struct')   % Old style Options as struct:
  Opt  = struct2cell(Opt{1});
  nOpt = numel(Opt);
end
% Loop over strings in the input: ----------------------------------------
for iOpt = 1:nOpt
  aOpt = Opt{iOpt};
  if ~ischar(aOpt)
    Error_L('BadInputType', '[Opt] must be a struct or chars.');
  end
  switch lower(aOpt)
    case 'file'              % Data contains the file name:
      isFile = true;
    case {'bin', 'binary'}  % Just the contents of the data:
      if (isnumeric(Data) || ischar(Data) || islogical(Data) || ...
          myIsString(Data)) == 0 || issparse(Data)
        Error_L('BadDataType', ['[Bin] input needs data type: ', ...
          'numeric, CHAR, LOGICAL, STRING.']);
```

```
         end
         isBin = true;
      case 'array'
         isBin = false;      % Is the default already
      case {'asc', 'ascii'}  % 8-bit part of MATLAB CHAR or STRING:
         isBin = true;
         if ischar(Data)
            Data  = uint8(Data);
         elseif myIsString(Data) && numel(Data) == 1
            Data  = uint8(char(Data));
         else
            Error_L('BadDataType', ...
               'ASCII method: Data must be a CHAR or scalar STRING.');
         end
      case 'hex'
         if aOpt(1) == 'H'
            OutFormat = 'HEX';
         else
            OutFormat = 'hex';
         end
      case {'double', 'uint8', 'short', 'base64'}
         OutFormat = lower(aOpt);
      otherwise  % Guess that this is the method:
         Method = upper(aOpt);
   end
end

end

%
******************************************************************************
function FuncKey = ConvertFuncHandle(FuncH)
%   The subfunction ConvertFuncHandle converts function_handles to a struct
%   using the Matlab function FUNCTIONS. The output of this function changes
%   with the Matlab version, such that DataHash(@sin) replies different hashes
%   under Matlab 6.5 and 2009a.
%   An alternative is using the function name and name of the file for
%   function_handles, but this is not unique for nested or anonymous functions.
%   If the MATLABROOT is removed from the file's path, at least the hash of
%   Matlab's toolbox functions is (usually!) not influenced by the version.
%   Finally I'm in doubt if there is a unique method to hash function handles.
%   Please adjust the subfunction ConvertFuncHandles to your needs.

% The Matlab version influences the conversion by FUNCTIONS:
% 1. The format of the struct replied FUNCTIONS is not fixed,
% 2. The full paths of toolbox function e.g. for @mean differ.
FuncKey = functions(FuncH);

% Include modification file time and file size. Suggested by Aslak Grinsted:
if ~isempty(FuncKey.file)
   d = dir(FuncKey.file);
   if ~isempty(d)
      FuncKey.filebytes = d.bytes;
      FuncKey.filedate  = d.datenum;
   end
end

% ALTERNATIVE: Use name and path. The <matlabroot> part of the toolbox functions
% is replaced such that the hash for @mean does not depend on the Matlab
% version.
% Drawbacks: Anonymous functions, nested functions...
% funcStruct = functions(FuncH);
% funcfile   = strrep(funcStruct.file, matlabroot, '<MATLAB>');
% FuncKey    = uint8([funcStruct.function, ' ', funcfile]);

% Finally I'm afraid there is no unique method to get a hash for a function
% handle. Please adjust this conversion to your needs.

end
```

```
% 
********************************************************************************
function DataBin = ConvertObject(DataObj)
% Convert a user-defined object to a binary stream. There cannot be a unique
% solution, so this part is left for the user...

try    % Perhaps a direct conversion is implemented:
   DataBin = uint8(DataObj);

   % Matt Raum had this excellent idea - unfortunately this function is
   % undocumented and might not be supported in te future:
   % DataBin = getByteStreamFromArray(DataObj);

catch  % Or perhaps this is better:
   WarnS   = warning('off', 'MATLAB:structOnObject');
   DataBin = struct(DataObj);
   warning(WarnS);
end

end

% 
********************************************************************************
function Out = fBase64_enc(In, doPad)
% Encode numeric vector of UINT8 values to base64 string.
B64 = org.apache.commons.codec.binary.Base64;
Out = char(B64.encode(In)).';
if ~doPad
   Out(Out == '=') = [];
end
% Matlab method:
% Pool = [65:90, 97:122, 48:57, 43, 47];  % [0:9, a:z, A:Z, +, /]
% v8   = [128; 64; 32; 16; 8; 4; 2; 1];
% v6   = [32, 16, 8, 4, 2, 1];
```

```
%
% In = reshape(In, 1, []);
% X  = rem(floor(bsxfun(@rdivide, In, v8)), 2);
% d6 = rem(numel(X), 6);
% if d6 ~= 0
%    X = [X(:); zeros(6 - d6, 1)];
% end
% Out = char(Pool(1 + v6 * reshape(X, 6, [])));
%
% p = 3 - rem(numel(Out) - 1, 4);
% if doPad && p ~= 0  % Standard base64 string with trailing padding:
%    Out = [Out, repmat('=', 1, p)];
% end
end
%
*******************************************************************************
function T = myIsString(S)
% isstring was introduced in R2016:
persistent hasString
if isempty(hasString)
   matlabVer = [100, 1] * sscanf(version, '%d.', 2);
   hasString = (matlabVer >= 901);  % isstring existing since R2016b
end
T = hasString && isstring(S);  % Short-circuting
end
%
*******************************************************************************
function R = Version_L()
% The output differs between versions of this function. So give the user a
% chance to recognize the version:
% 1: 01-May-2011, Initial version
% 2: 15-Feb-2015, The number of dimensions is considered in addition.
%    In version 1 these variables had the same hash:
%    zeros(1,1) and zeros(1,1,0), complex(0) and zeros(1,1,0,0)
% 3: 29-Jun-2015, Struct arrays are processed field by field and not element
%    by element, because this is much faster. In consequence the hash value
%    differs, if the input contains a struct.
% 4: 28-Feb-2016 15:20, same output as GetMD5 for MD5 sums. Therefore the
%    dimensions are casted to UINT64 at first.
%    19-May-2018 01:13, STRING type considered.
R.HashVersion = 4;
R.Date       = [2018, 5, 19];
R.HashMethod  = {};
try
  Provider = java.security.Security.getProviders;
  for iProvider = 1:numel(Provider)
    S     = char(Provider(iProvider).getServices);
    Index = strfind(S, 'MessageDigest.');
    for iDigest = 1:length(Index)
      Digest       = strtok(S(Index(iDigest):end));
      Digest       = strrep(Digest, 'MessageDigest.', '');
      R.HashMethod = cat(2, R.HashMethod, {Digest});
    end
  end
catch ME
  fprintf(2, '%s\n', ME.message);
  R.HashMethod = 'error';
end
end
%
*******************************************************************************
function Error_L(ID, varargin)
error(['JSimon:', mfilename, ':', ID], ['*** %s: ', varargin{1}], ...
  mfilename, varargin{2:nargin - 1});
end
```

**4.2 Proof of Work**



```
hold on;
h=text(600,450, 'CENTERALIZED CONTROL UNIT');
axis([0 1000 0 1000])

% Locations of users and plotting %
    xloc_user(1)=100;
    yloc_user(1)=300;
    plot(xloc_user(1)+5,yloc_user(1)+5,'bs','MarkerSize',8,'linewidth',2,'MarkerSize',10);
    text(xloc_user(1)+13,yloc_user(1)+10,strcat('U',num2str(1)));
    hold on;
    pause(0.5)
    xloc_user(2)=300;
    yloc_user(2)=800;
    plot(xloc_user(2)+5,yloc_user(2)+5,'bs','MarkerSize',8,'linewidth',2,'MarkerSize',10);
    text(xloc_user(2)+13,yloc_user(2)+10,strcat('U',num2str(2)));
    hold on;
```

Fig. 4.1 Code

## 4.3 Interface

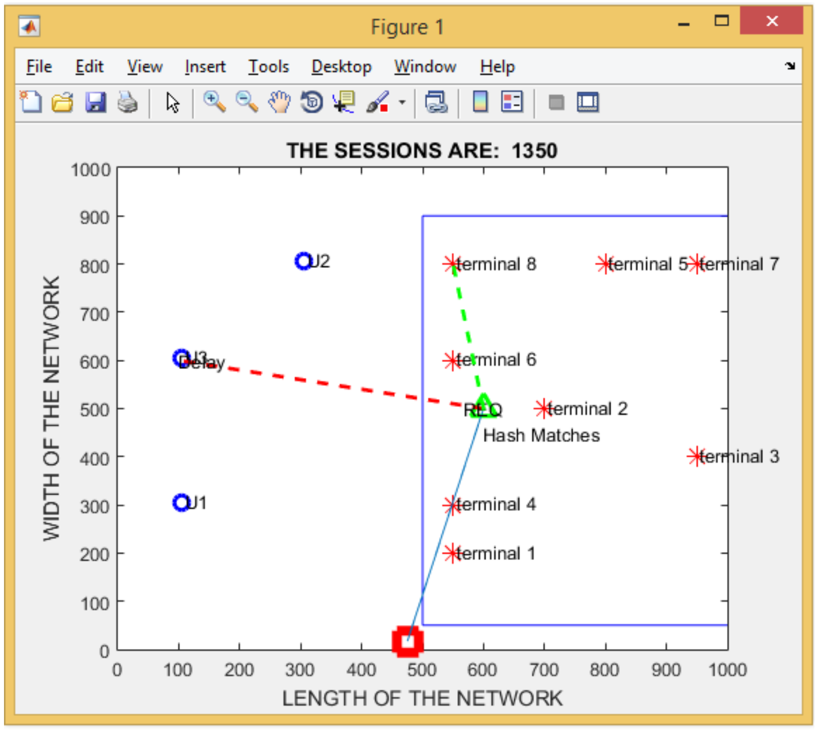


Fig. 4.2 Interface

## 4.4 Analysis

Now with say n= 500 people calculating H hardness proofs with larger target t have to compute 500.H computations as compared to one miner mining 1000H worth of computations.

Also clearly the reward taken by any one wallet is significantly lowered down and so people would be discouraged to get standalone machines which can mine extensively and would rather use existing machines.

Clear achievement of this project is a new strategy which enhances the distributed feature of entire block chain operation.

The chain is secure as if we consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain.

Now we consider the energy savings. These can be estimated to the follows:

Let the hardness of a certain problem be H with target t. The hardness of the zero knowledge proofs increases exponentially with the target value for a target 0.1t is say 1000H (as are the values from the multi proofs interaction: Shamir, Feige).Now with say n= 500 people calculating H hardness proofs with larger target t have to compute 500.H computations as compared to one miner mining 1000H worth of computations.

Also clearly the reward taken by any one wallet is significantly lowered down and so people would be discouraged to get standalone machines which can mine extensively and would rather use existing machines.

## 4.5 Conclusion

The proof of work strategy can be improved by the use of a multi miner approach per block. We have achieved simultaneous existence of all proofs by using the pay to address as a differentiating factor. The limitations which applied to previous proof of work strategy have been removed or weakened.

- The 51% attack can is now weakened since each node can have a say in mining the block chain and hence the chain will grow faster and with it being increasingly harder to control 51% of the entire chain.

- Processing power and energy requirements will be reduced as discussed above since both the reduction in the awarded values and also the easiness of the problems.

- The available processing power on the network has now been made into a more liquid form clearly since to be able to mine the amount of processing power requirement have been brought down significantly.

**4.6 Future work and limitations**

The length requirement for proofs in blocks has been increased and hence the scalability of certain existing systems like legacy bit coin which have fixed size of blocks. Legacy systems will need to change entire block structures because of the change and hence require what we call a hard reset.

What we can do more is generate a working smart contract engine or a crypto currency but due to the limited scope of the project we have stuck with a practical demonstration.

# CHAPTER 5

# RESULTS

The existing and proposed technique has been compared on two basic parameters. One is the throughput and another is the energy consumptions. The energy consumption for the proposed technique can be measured. Because the number of the users who will submit the hash with pay key is fixed. All the persons with in the count will get the bit coins. But in the existing technique only one person will be allocated with the bit coin who ever will submit the hash first. Everyone else will be discarded irrespective of the position and correctness. This will waste large amount of the effort of the people and also will reduces the interest of the people. For the understanding point of the view the system correct at the number of the users who will get the reward will be increased will distribute the energy amongst the people.

## 5.1 Energy consumption

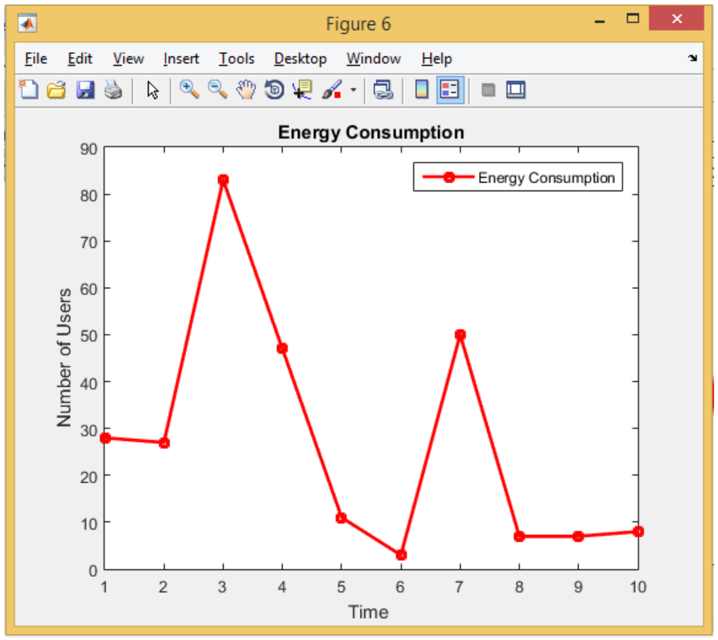


Fig. 5.1 Energy consumption

In Fig. 5.1 the energy consumption for the proposed technique has been define. At the initial instance the energy consumption is at the higher level. as the number of users get submitting the hash the energy consumption will be reduced there on.

## 5.2 Throughput

This will measure the rewards against the energy. As the multiple persons get rewards against the submission. This will increase the throughput for the overall system.

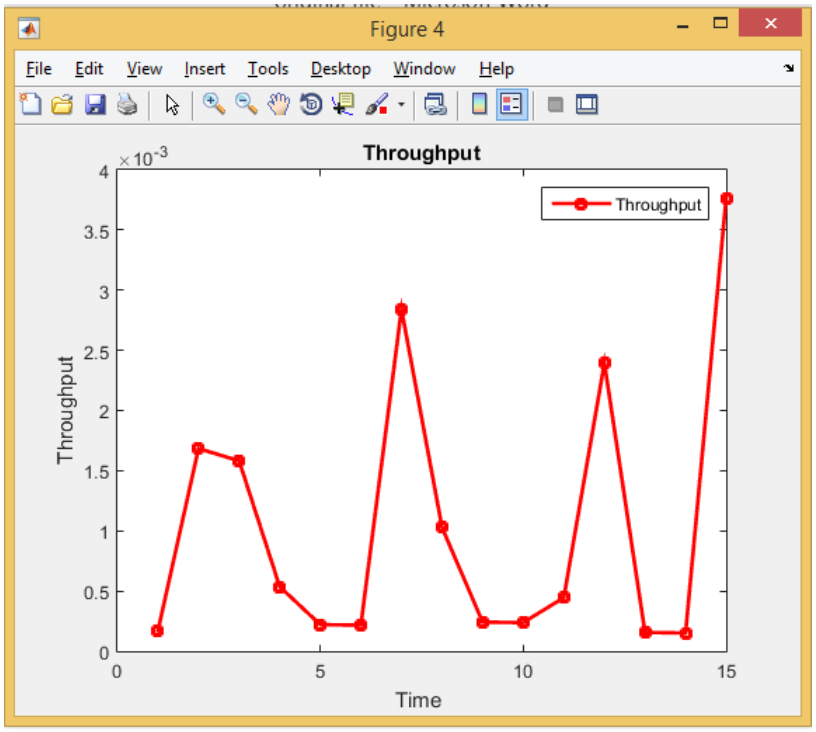


Fig. 5.2 Throughput

## REFERENCES

[1] Bentov, I., Gabizon, A., and Mizrahi, A. Crypto currencies without proof of work. Paper presented at the International Conference on Financial Cryptography and Data Security (pp. 142-157). Springer, Berlin, Heidelberg, February 2016.

[2] Hajdarbegovic, N. Bit coin Miners Ditch Ghash.io Pool over Fears of 51% Attack.CoinDesk. 2014

[3] Don Tapscott and Alex Tapscott, Block chain Revolution: How the Technology Behind Bit coin Is Changing Money, Business, and the World, 1st ed. New York, USA: Penguin Publishing Group, 2016.

[4] Bentov, I., Lee, C., Mizrahi, A., and Rosenfeld, M. Proof of Activity: Extending Bit coin's Proof of Work via Proof of Stake [Extended Abstract] y. ACM SIGMETRICS Performance Evaluation Review, 2014. 42(3), 34-37.

[5] Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bit coin. In: 2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, 18–21 May 2014.

[6] Nakamoto, S. Bit coin: A peer-to-peer electronic cash system 2008.

[7] King, S., and Nadal, S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. Self-published paper 2012.

[8] Bentov, I., Lee, C., Mizrahi, A., and Rosenfeld, M. Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake [Extended Abstract] y. ACM SIGMETRICS Performance Evaluation Review, 42(3), 34-37. 2014.