Storage-centric Mutable Blockchain in the network of trusted nodes

Abhijith Rajeev (Abe)

This prototype is built according to the proposed model; **Reserved blocks** – Nodes get ownership of the blocks whenever they are moving out of strong connectivity, in that case; only the owner nodes are able to contribute to that particular block. Rest of the nodes will keep contributing to the blockchain apart from the reserved block (this part is not implemented in this prototype).

In this document we will discuss two prototypes, **first** is a simulation where all the blockchain operations which were discussed in the earlier proposal are performed over an API interface and the HTTP server acts as a master node which verifies and maintains the blockchain integrity. Since it's a simulation and API call, we will be manually entering the Owner ID and Node ID,

Second is where the more detailed networking is established over a TCP interface, which allows us to launch and communicate via multiple terminal windows. Since it's a TCP communication, the data is in the form of bytes and decoding the information such as Owner and Node ID are not achieved yet. This prototype will only have data (BPM data), provided by nodes and not Owner ID and Node ID

Data model: Set of information in each blocks are in the following order

Index – block index

Timestamp – time at which block is framed

Owner – owner ID, if assigned any (0 – in case of no owner)

Node – ID of the node which is contributing data to the block

BPM – data uploaded to the block, integer array data type is used in this prototype (BPM = beats per min, adopted from a blockchain repo)

Hash – hash generated from the entire block (in this prototype its generated by including all block information except for data (BPM array))

Previous Hash – hash of the previous block

Note: Prototype is built with **Golang**; for the availability of packages, boiler plate codes and code reusability.

Genesis block: Genesis block is the first block in the chain, and it's hard coded. In this prototype blockchain is an array or slice to be more precise (a Golang datatype)


```
{
    "Index": 0,
    "Timestamp": "2018-06-15 04:04:00.9346622 -0400 EDT m=+0.024374501",
    "Owner": 0,
    "Node": 0,
    "BPM": [
      65,
      23,
      54
    ],
    "Hash": "2ac9a6746aca543af8dff39894cfe8173afba21eb01c6fae33d52947222855ef",
    "PrevHash": ""
  }
```

BPM data size set to 3, for the purpose of this demonstration.

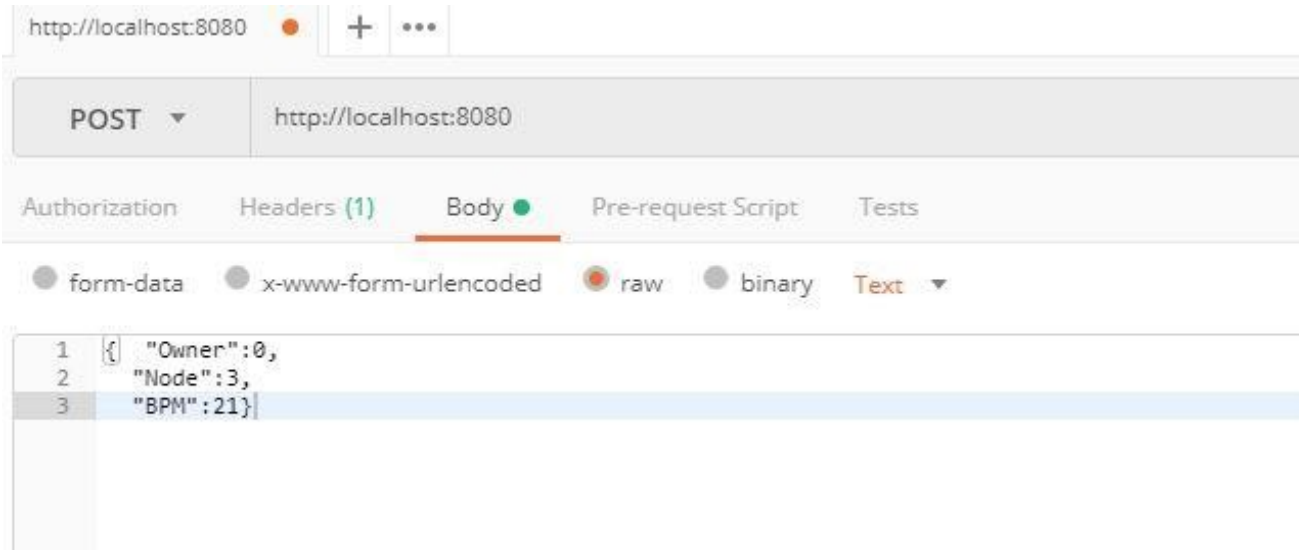
- Hash of every block is linked using the previous block, used to validate the integrity of the chain. In case conflict (when two nodes frames the block at nearly same time), longest chain rule is used to maintain the blockchain continuity.

Prototype 1:

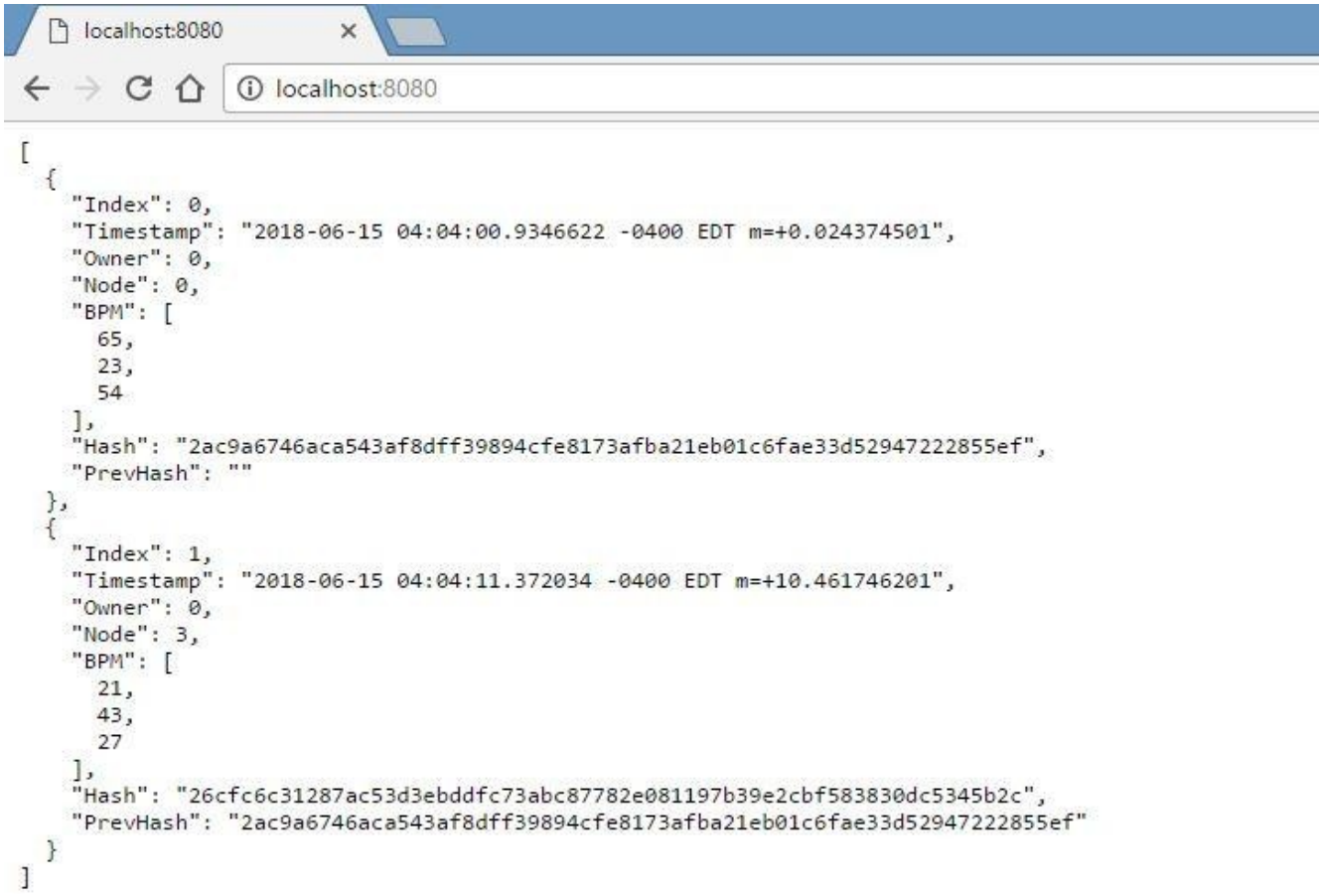
- A HTTP server is established and launched, popular golang package **Gorilla Mux** is used to support this feature. PORT 8080 which is provided in the .env file is read through the mux function and the port is made open for communication.
- The golang package **godotenv** is used here to read from the file and to operate through the port. The goalng package **spew** is used to print the JSON data in a well formatted way.
- Once the program is made running, the genesis block is visible on the browser window.

- It's also visible in the terminal window;

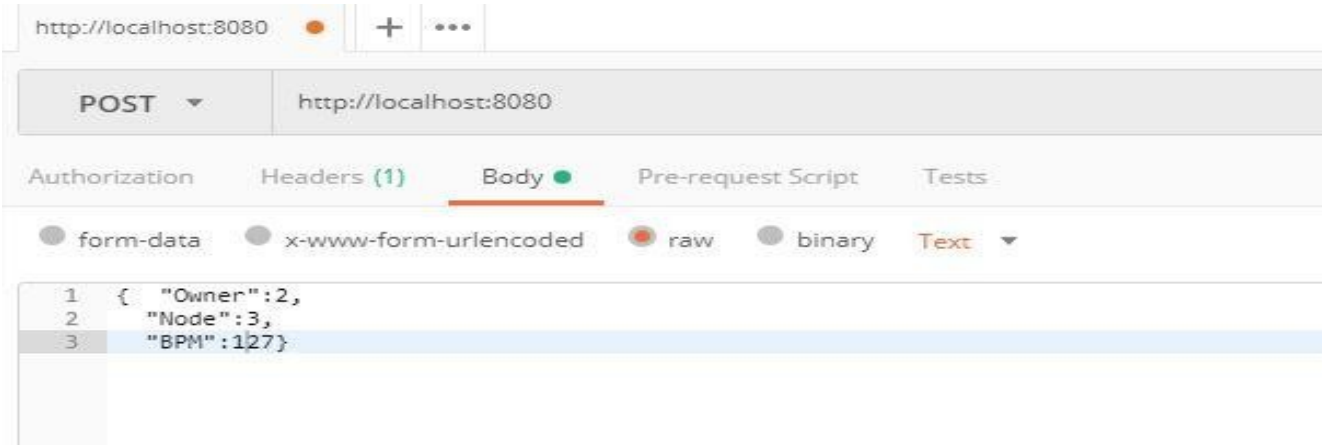
- Now open an API testing tool, we can start playing with our blockchain. Here we can see the manually given information such as Node ID, Owner ID has been sent along with the actual data.



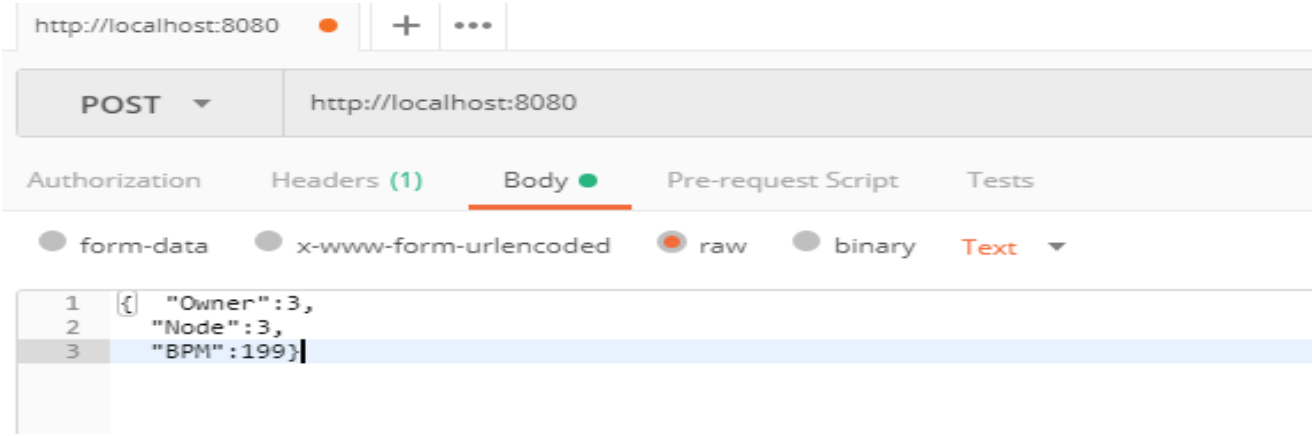
- Now we can see the same data reflected in our blockchain, here Owner = 0; means that there are no reserved blocks. Any nodes can contribute to the block and BPM is the data being sent to the chain from node 3.
- We'll do more post requests and see which are the data being added and being rejected from the blockchain. And we can also see that after three requests; i.e, after three integer elements being sent block is framed and the chain starts to accept data for the new block. (this is the limit set for the purpose of this demonstration)



- Let's see when the blocks are being reserved, in this case only the owner node will be able to contribute to the block. Meaning that the Owner ID and the Node ID must be same in order to contribute to the node.
- Ideally rest of the nodes should be able to contribute to the unreserved – highest block, but in this demonstrational prototype only single mutex operation was performed.



- This data will not be posted onto our blockchain as there is conflict in the Owner ID and Node ID.



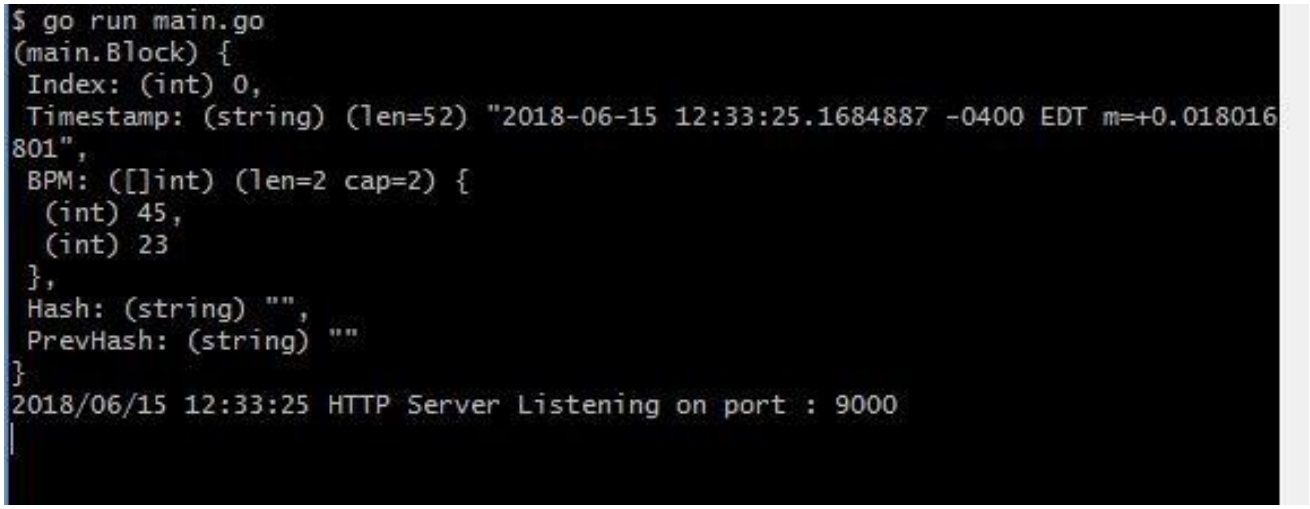
- Now with a modified Owner and the Node IDs, the new data to be added to the blockchain is sent to the master node for verification. And the chain with new block is below.


```
{
  "Index": 1,
  "Timestamp": "2018-06-15 04:04:11.372034 -0400 EDT m=+10.461746201",
  "Owner": 0,
  "Node": 3,
  "BPM": [
    21,
    43,
    27
  ],
  "Hash": "26cfc6c31287ac53d3ebddfc73abc87782e081197b39e2cbf583830dc5345b2c",
  "PrevHash": "2ac9a6746aca543af8dff39894cfe8173afba21eb01c6fae33d52947222855ef"
},
{
  "Index": 2,
  "Timestamp": "2018-06-15 07:22:15.7483113 -0400 EDT m=+11894.842952901",
  "Owner": 3,
  "Node": 3,
  "BPM": [
    199,
    19,
    67
  ],
  "Hash": "ff7db99d7e563ad17925a6922e9fda380d352468412556b060c4ece0a4dd5a81",
  "PrevHash": "26cfc6c31287ac53d3ebddfc73abc87782e081197b39e2cbf583830dc5345b2c"
}
```

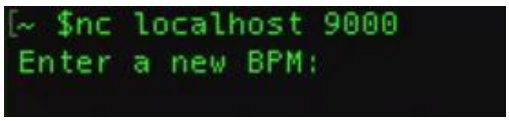
Note: Source code and readme file to run the prototype application will be given along with this document.

Prototype 2:

- A TCP server is established over the PORT 9000 along with initiating the genesis block, any nodes can come in connection with the established server by using the command: nc localhost 9000.
- Once the connection is established, nodes will be able to contribute to the blockchain, as in the prototype 1 the data in this blockchain is kept mutable and of length two integer arrays.
- Since is the data is transmitted over the TCP, it's in the form of bytes and we are not doing the bytes segmentation here so only data (BPM) is sent over the established channel (No owner id or node id).
- This prototype almost mimics the operation of network of nodes in a blockchain environment. Whereas in the actual live blockchain network; nodes conmmunicate via RPCs (remote procedural calls), here as its build on a single machine and instead of having multiple machines as nodes we have multiple terminals; the communication is via IPCs (Inter Process Communication).



- The server starts on the port 9000, along with the genesis block. Now let's connect to this by opening as many terminals as we want by entering the command; **nc localhost 9000**



- By entering a new BPM value any node can contribute to the blockchain. And the initiator node as a master node and handles all the verification and validation part.

Future work:

- **IPFS/storage integration**: Since it's a storage centric blockchain adding IPFS or any other sort of storage (cloud storage like S3) part will be done in the future.

- **Multi-node validating capacity**: As this is a simulation, only a master node is capable of validating and adding blocks to the chain. A much sophisticated consensus algorithm with application and rules based in all connected nodes makes it a fully functional blockchain.

- **Remote procedural calls**: In this simulation the inter-nodal communication (inter-process communication) is established over IPCs. The actual network would require RPC capability.

- **Network Mem pool**: The transactions contributed by the nodes of network are stored in a memory buffer before adding to the blockchain. Here in this prototype we are using an array buffer on the master node, but in the actual blockchain it's going to be a distributed unit.

- **Distributed hash table or Merkle tree integration**: For the purpose of demonstration and keep the prototype simple the data type used was a simple integer array. The actual blockchain would have wither distributed hash table or a merkle tree as the storage part of the block.

- **Secure access of data**: The encryption techniques like proxy re-encryption or circular encryption can be integrated with the blockchain to delegate access only required nodes.

Note: Source code of both the prototypes are sent along with this document. With a minimal environmental setup anyone would be able to run the prototype model on their machine.

Setup:

- Once Golang is on the machine, we would need the packages by entering the command
  ```
  go get github.com/davecgh/go-spew/spew
  go get github.com/gorilla/mux
  go get github.com/joho/godotenv
  ```
- In case .env is missing, create a file with the name ".env" with the content "PORT=8080". This would help in establishing the master nodal server.
- Then run the source code file;
  ```
  Go run main.go
  ```
- And contribute to the blockchain via, Postman. Data sent is in the form of JSON.
- For the protype 2: after running the main.go
  - Open as many terminal as wanted by entering the command: nc localhost 9000
- Contribute to the blockchain by sending over the BPM data.