



# Bit-Ray Network

A Decentralized Transaction Ledger and Cloud  
Storage Infrastructure

white paper v1.0

[HTTPS://GITHUB.COM/BITRAY-PROJECT/GO-BITRAY](https://github.com/BitRay-project/go-bitray)

This research was done from August 2020 until now.

*First release, April 2021*

# Contents

<b>1</b>	<b>Introduction</b> .....	<b>5</b>
1.1	Background and history	5
1.2	Motivation	7
1.2.1	Existing storage solutions .....	7
1.2.2	Discussions .....	8
1.3	Objective	9
1.4	Outlines	10
<b>2</b>	<b>Network Mechanism</b> .....	<b>11</b>
2.1	The Bit-Ray Network	11
2.1.1	Account .....	13
2.1.2	State .....	14
2.1.3	Gas and payment .....	17
2.1.4	Transactions and messages .....	19
2.1.5	BitRay virtual machine (BVM) .....	22
2.2	The storage Network	24
2.2.1	The Decentralized Power of BitRay and IPFS .....	24
2.2.2	Storage market .....	26
2.2.3	Retrieval market .....	26
2.3	Network unification and mining economics	27
2.4	Conclusion	28

<b>3</b>	<b>Economics Model</b>	<b>29</b>
3.0.1	Initial BitRay token distribution	29
3.0.2	Initial BitRay token distribution usage	29
<b>4</b>	<b>Implementation and Iteration</b>	<b>31</b>
4.0.1	MileStones	31

# 1. Introduction

## 1.1 Background and history

The development of bitcoin (Satoshi, 2009) has been viewed as a radical development in money and currency. It is the first example of a digital asset which has no centralized issuer or controller. The significance of bitcoin experiment is the introduction of the underlying blockchain technology. It is the first type of money that is completely virtual. It's like an online version of cash. You can use it to buy products and services and more individuals, shops and companies begins to use bitcoin as a method of payment. In Oct 2021, for example, the online payment service, PayPal, announced that it would be allowing its customers to buy and sell Bitcoin. Recently, as talk of the currency has gone global, the Bank of Singapore has suggested that the 12-year-old currency could replace gold as its store of value. In Feb 2021, Tesla, led by Elon Musk, confirmed that it purchased about \$1.5 billion in bitcoin in January and expects to start accepting it as a payment in the future. Tesla has made roughly \$1 billion in paper profits from its investment into bitcoin, according to Daniel Ives, analyst at Wedbush Securities.

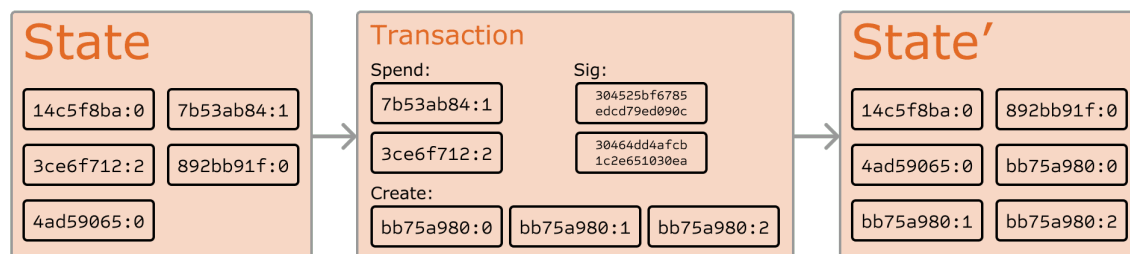
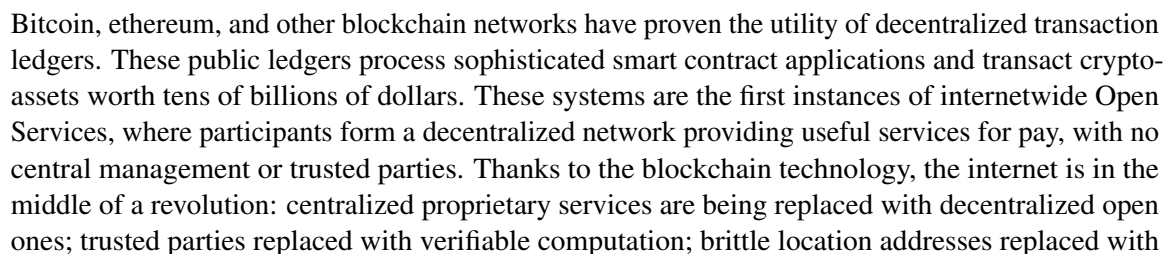


Figure 1.1: The Bitcoin Model (source:<https://ethereum.org/pt/whitepaper/>)

Apart from bitcoin being as a tool of distributed consensus, the research attention is rapidly starting to shift to other aspects of the blockchain industry by incorporating more functions and features into the network. In 2013, Vitalik Buterin, the founder of Ethereum, published the white paper of

Figure 1.2: Ethereum Interpretation (source:<https://ethereum.stackexchange.com/questions/268/ethereum-block-architecture>)



resilient content addresses; inefficient monolithic services replaced with peer-to-peer algorithmic markets.

## 1.2 Motivation

Dicentralized networks such as the ethereum can execute runnable script by incorporating a virtual stack machine into the network. However, in real applications, the cost of storing data will be significantly high in these types of network. According to Fig. 4.1 and ethereum yellow paper (Gavin, 2017), the fee is 20000 gas to store a 32-byte word. Let's assume the gas fee right now is around 50 Gwei (0.00000005 ETH). 1KB of storage costs 0.032 ETH. 1GB costs 32,000 ETH, which is \$57/KB given the current price of ethereum, making applications of larger scale programs almost impossible. Therefore, designing a separate storage system while incorporating smart contracts can be of great interest.

### 1.2.1 Existing storage solutions

#### **Centralized storage network (iCloud, GoogleDrive and Dropbox)**

These entities provide users 2GB-15GB of free storage and some bandwidth. If one requires larger storage then paid products is needed. These services are not integrated with open P2P network, nor are they decentralized. Each entity is implemented with central large disk clusters. If something terrible happens to one of these services, system down time may happen and in the worst case, users can potentially lose some of their valuable data.

#### **Decentralized storage network (Filecoin, Siacoin and Storj)**

Given the above discussions about centralized storage, decentralized ones can certainly play an important role in storing humanity's data, given their content distributions worldwide. Filecoin, as a decentralized storage network, would be a perfect example in this case. It is backed by IPFS (Juan, 2014) team for the purpose of incentivizing nodes to provide storage services on the network. The filecoin protocol creates a blockchain that utilizes the latest advancements of cryptographic proofs to generate succinct and trustless proofs (protocol labs, 2017). The protocol adapts proof-of-replication (a special type of proof-of-storage) and proof-of-spacetime, in order to guarantee miners store the specific file for a certain duration of time. The protocol requires nodes on the network to run auditors and validate storage providers.

Filecoin is incentivizing the network to provide storage services by paying storage miners to store and fetch files from the network. The idea is to utilize the large amount of unused storage on home computers and servers around the globe. Filecoin succeeds in its official implementation—*Lotus*<sup>1</sup>, which enables the owners of unused storage to monetize it, while eliminating any 3rd parties or central servers on the network, where different nodes can coordinate themselves by following the filecoin protocol.

Other decentralized storage solutions such as Siacoin and Storj, all attempt to collect micropayments for both storage and retrieval of data. They are similar to filecoin since they are using similar incentivizing ideals for data storage and data retrieval, and they are all targeting internet computer storage providers renting out space located behind connections.

---

<sup>1</sup><https://github.com/filecoin-project/lotus>



### 1.2.2 Discussions

For decentralized storage, a crucial drawback of the storage bandwidth model lies in the fact that they require users to continually pay for their storage for a certain range of time (file storage) and bandwidth (file retrieval), either by submitting transactions to the blockchain or initiating micro payment channel with retrieval miner (protocol lab, 2017). This means the files will never be available for the general public to access for free via browser, from a financial perspective. We give one example below.

#### Case study: DTube

Decentralized tube (DTube), implied by the name, is basically a Youtube running on blockchain based mainnet. It looks like Youtube in appearance, having almost the same interface but implemented in a decentralized way. There are a lot of potential benefits in which DTube can perform better than Youtube.

- Since DTube is based on the blockchain, it follows the same rules of secure, decentralized, list of record keeping. The data on the blockchain cannot be tempered. Once a user has uploaded the videos to DTube, he/she can be sure it can't be removed or edited.
- Unlike youtube, there won't be any advertisements due to its decentralization. Users can feel quite comfortable and lucky when watching videos without any disturbance.
- Everyone earns Money! Unlike Youtube, where only the uploaders can earn money, on DTube, content creators and content watchers will both be able to earn rewards in the shape of tokens, that can be later on exchanged in the open market, which is cool.

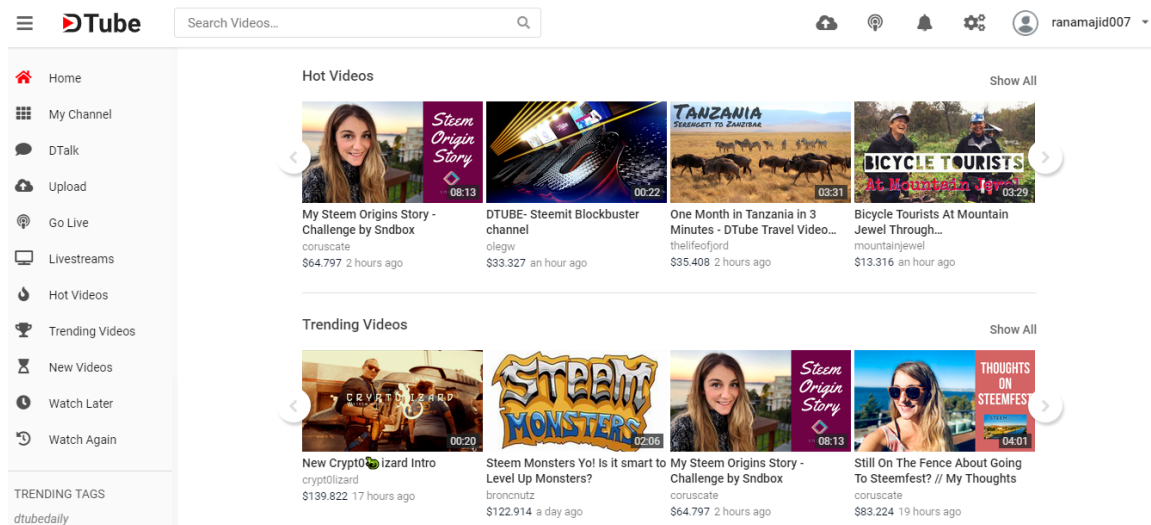


Figure 1.3: Dtube Website: <https://d.tube/>

So how does DTube works? Actually, DTube videos are not uploaded onto the blockchain. Rather, links to these videos are stored onto the blockchain. The real video file, in a specific format is stored using IPFS technology. IPFS is a protocol that enables storing of data in a decentralized manner on the internet (Juan, 2014). It works on the principle that follows “Distributed Hash Tables (DHT)”. In a similar fashion of other cryptocurrencies using asymmetric cryptography, DHT networks will have hash content to identify a file. This hash is basically the identity of the record, and it is so easy to repeat data and ensure that the hashes match to ensure that the file sent to us is the original.



Sounds pretty good so far? However, IPFS does not guarantee the availability of files. A file can disappear in the case that nodes storing it decline to make it available. Applications such as DTube can't simply rely on IPFS frameworks to be confident that the file will always exist and can be accessible when needed.

How about the existing decentralized storage solutions? According to filecoin protocol, the network will charge users for both storage and bandwidth which the network miners provided. The decentralized bandwidth solutions rely on micro-payments. This yields two major drawbacks. First, this transfer is difficult to guarantee clients will actually get the files they want because the network will not monitor on this process and the retrieval miner has to be trusted (protocol labs, 2017). Furthermore, micro-payments create transaction friction which discourages application promotion. In practice we typically see strong consumer resistance to micropayments in internet applications while most of them prefer one-time payments or free services<sup>2</sup>. In the case of DTube, the individual who uploads the videos may likely be very different from the individual who watched it (probably viewed by millions of people). It would be ideal for each individual to pay for their own bandwidth since the publisher of the video does not want to or is simply not able to pay for the bandwidth consumption of a million viewers. In this case, filecoin's network is not a viable solution because users will be upset with the micro-payments every time they watch the videos. Again, micro-payments will always hinder adoption. That being said, there are potential viable solutions possible in the commercial aspect, as we will present our solution in the following sections.

### 1.3 Objective

In this work we are motivated to develop a smart contract platform with a decentralized storage network that turns cloud storage into an algorithmic market. The platform runs on a blockchain with a native protocol token (also called "BitRay"), which miners earn by providing proof-of-work (POW) or by providing storage/bandwidth to clients, or both. Conversely, clients spend BitRay by hiring miners to store or distribute data. Just like bitcoin and ethereum, BitRay miners compete to mine blocks with block rewards, but BitRay mining power not only goes to POW, but is also related to active storage, which directly provides a useful service to clients in contrast to most other platforms whose usefulness is limited to maintaining blockchain consensus and security. This creates a powerful incentive for miners to not only contribute their hash power, but also contribute as much storage as they can, and rent it out to clients. The protocol abstracts those storage resources into a self-healing storage network that anybody in the world can rely on. In terms of storing content, the network achieves robustness by replicating and dispersing content across nodes, while automatically and periodically detecting and repairing content failures. Clients can select replication parameters to protect against different threat models such as Byzantine faults. The BitRay's cloud storage network also provides security of content, as content can be encrypted at the client side before uploading, while storage providers do not have access to decryption keys, they would never be able to recover the original file. BitRay's storage network works as an incentive layer on top of IPFS, which can provide storage infrastructure for the world's data. The aim of BitRay targets mainly at decentralizing data, implementing smart contracts, building and running distributed applications, and could storage. Ultimately, we wish to develop BitRay as a leading open-source platform for blockchain innovation and performance. Developers and businesses around the world can use BitRay to create secure, transparent, and deterministic digital infrastructures with scalability in storage.

---

<sup>2</sup>page 5 in <http://www.dtc.umn.edu/~odlyzko/doc/price.war.pdf>

## 1.4 Outlines

This work:

- Introduces the BitRay Network, dives into several components in detail and presents an overview of the network protocol.
- Introduces consensus schemes used in this network: *proof-of-work (PoW)*, *proof-of-storage (PoS)* and *proof-of-lock (PoL)*. PoW is used to guarantee the safety of the blockchain. PoS allows proving that any replica of data is stored in physically independent storage. PoL allows clients to consume bandwidth by locking some of their shares into a pre-compiled smart contracts in the network.
- Constructs two markets in the storage network, a Storage Market with PoS and a Retrieval Market with PoL, which govern how data is published to and read from BitRay storage network, respectively.
- Introduces the commercial incentive behind *proof-of-lock (PoL)* as the storage network bandwidth model.
- Discusses BitRay commercial values, including use cases, connections to other decentralized systems, and the protocol's potential in real applications.

## 2. Network Mechanism

### 2.1 The Bit-Ray Network

The Bit-Ray network is implemented with a blockchain data structure. It contains a cryptographically secure transactional machine with a shared-state, as well as a storage network that the blockchain manages.

- **"cryptographically secure"** means that the creation of network's currency (earned by creation of blocks) and the client's digital assets (network's currency they earned or received) are secured by complex mathematical algorithms that are obscenely hard to break. It is secure in a sense that it is impossible to cheat the system (e.g. create fake transactions, erase transactions, etc..) with limited computing power.
- **"transactional machine"** means that there's a single canonical state of the machine responsible for all the transactions being created in the system.
- **"with shared-state"** means that the distributed system is synchronized. The state stored on this machine is shared and open to every node. In other words, there's a single global truth that every honest node believes in.
- **"storage network"** is a part of the whole system and is combined with the network through BitRay tokens.

The Bit-Ray network is essentially a transaction-based state machine. This state machine will read a series of inputs and, based on those inputs, will transition to a new state. With BitRay's state machine, we begin with a "genesis state." This "genesis state" refers to a blank slate, before any transactions or mining activities have happened on the network. When transactions are executed, this genesis state transitions into some final state. At any point in time, this final state represents the current state of BitRay.

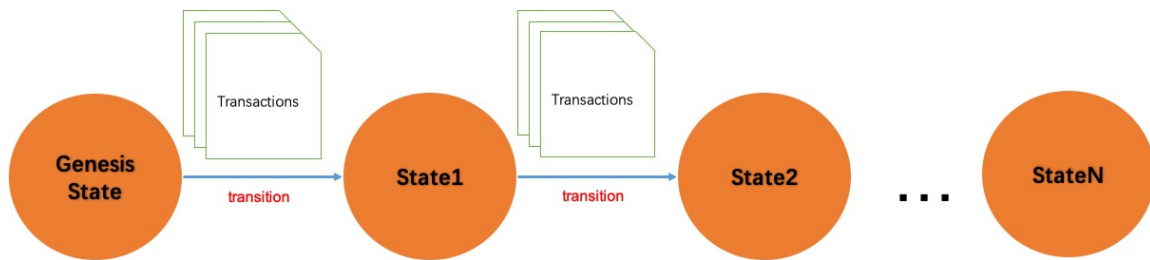


Figure 2.1: BitRay state transition diagram

The state of BitRay is modified by millions of transactions. These transactions are grouped into blocks. A block contains a series of transactions, and each block is chained together with the hash of its previous block, as shown in Fig. 4.1.

To cause a transition from one state to the next, a transaction must be valid. For a transaction to be considered valid, it must be validated by the BitRay consensus machine and must go through a the mining process. Mining is when a group of nodes spend their computing resources (hash powers) to create a block of valid transactions. If one transaction is invalid, miners will simply discard the transaction.

Any node on the network that regards itself as a miner can attempt to create and validate a block. Miners from around the world try to create and validate blocks at the same time. Each miner provides a mathematical proof when submitting a block to the blockchain, demonstrating enough computing power has been dedicated to the proof, which prevents sybil-attack. This proof is called *proof-of-work*, with the earliest version shown in the bitcoin network.

One might be asking: what guarantees that everyone sticks to one blockchain (the canonical chain)? Things can happen that two miners found one block at almost the same time. Also, how can we be sure that there doesn't exist a subset of miners who will decide to fork their own chain of blocks?

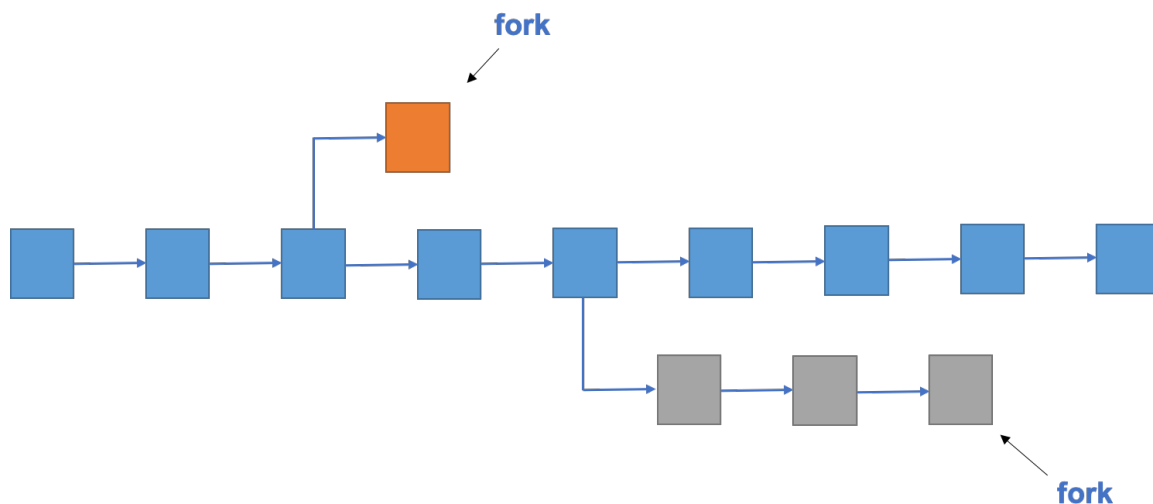


Figure 2.2: Multiple chains (canonical (shown in blue) chain+side chains)

At the beginning of this chapter, we introduced the BitRay blockchain as a transactional virtual machine with shared-state. We must be aware that the correct current state is a single global truth,

which everyone must acknowledge. Consensus engine would be compromised by having multiple states (or chains)

As shown in Fig. 2.2, whenever side chains are generated, a process called "chain fork" occurs. We typically want to avoid forks, because they interrupt the system and force nodes on the network to choose which chain they work on. To determine which chain is our canonical chain, BitRay borrows a mechanism called the Greedy Heaviest Observed Subtree (GHOST) protocol. The GHOST protocol demands that participants on the network pick the chain that has had the most computation done upon it. In BitRay network we determine the canonical chain by using the block number of the most recent **valid** block, which represents the total number of blocks in the current chain (not counting the genesis block). The higher the block number, the longer the chain and the greater the mining effort that must have gone into arriving at the head block. This allows us to agree on the canonical version of the current state.

Apart from blockchain, the main components of BitRay network consists of:

- Accounts
- State
- Gas and fees
- Transactions
- Blocks
- Virtual stack machine
- Mining and proof-of-work
- Storage network

### 2.1.1 Account

The global “shared-state” of Ethereum is comprised of many small objects (“accounts”) that are able to interact with one another through a message-passing framework. Each account has a state associated with it and a 20-byte address. An address in Ethereum is a 160-bit identifier that is used to identify any account.



Figure 2.3: Externally owned accounts (controlled by private key) and contract accounts (controlled by code)

Borrowing the account structure in (Vitalik, 2013), the global state of BitRay is comprised of many accounts which are able to interact with one another through a message-passing framework. Each account has a state associated with it and a 20-byte (160bit) address. There are two types of accounts:

- The most common type of address is externally owned account, which is controlled by private keys and have no code associated with them.
- In order to create decentralized applications, there is also another type called contract account, which is controlled by its code segment associated with them.

### Externally owned accounts & contract accounts

In this paragraph we explain the fundamental difference between externally owned accounts and contract accounts. An externally owned account can:

- Send network messages by creating and signing a transaction using its private key.
- Send network messages to other externally owned accounts. A message between two externally owned accounts is simply BitRay coins transfer.
- Send network messages to other contract accounts. A message from an externally owned account to a contract account activates the contract account's code, allowing it to execute various contract codes.

Any state changes which occur on the BitRay blockchain are always set in motion by transactions fired from externally controlled accounts. Contract accounts cannot initiate new transactions on their own and can only fire transactions in response to other transactions they have received from an externally owned account or from a contract account.

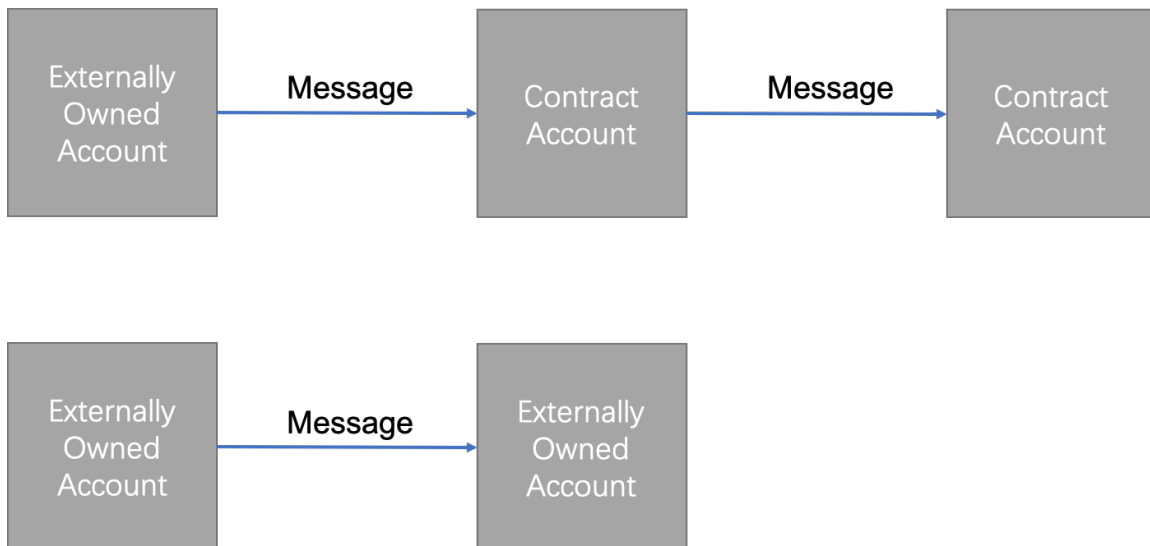


Figure 2.4: Message passing mechanism between externally owned accounts (controlled by private key) and contract accounts (controlled by code)

## 2.1.2 State

### Account State

The account state (present in externally owned accounts and contract accounts) consists of four components

- **Nonce:** If the account is an externally owned account, this number represents the number of transactions sent from the account's address. If the account is a contract account, the nonce is the number of contracts created by the account.
- **Balance:** The number of BitRay coins owned by this address.
- **StorageRoot:** A hash of the root node of a Merkle Patricia Tree (MPT). The tree's structure will be explained in the following sections. This tree encodes the hash of the storage contents of this account, and is empty by default.
- **CodeHash:** The hash of the BVM (BitRay virtual machine) code of this account. In the case

of externally owned accounts, the codeHash field is the hash of the empty string. In the case of contract accounts, codehash is the hash of the code that gets stored .

### World state

We now know that the BitRay's global state consists of a mapping between account addresses and the account states. This key-value mapping is stored in a data structure known as a Merkle Patricia Tree (MPT). For more detailed explanations of MPT we encourage the readers to refer to the ethereum yellow paper<sup>1</sup>, appendix D. MPT is basically a key-value database for the BitRay system. Beginning from the root node of the tree, the key will inform which child node to follow to get to the corresponding value, which is stored in the leaf nodes. In BitRay's case, the key-value mapping for the state tree is between addresses and their associated accounts, including the nonce, balance, codeHash, and storageRoot for each account, where the storageRoot is a tree itself.

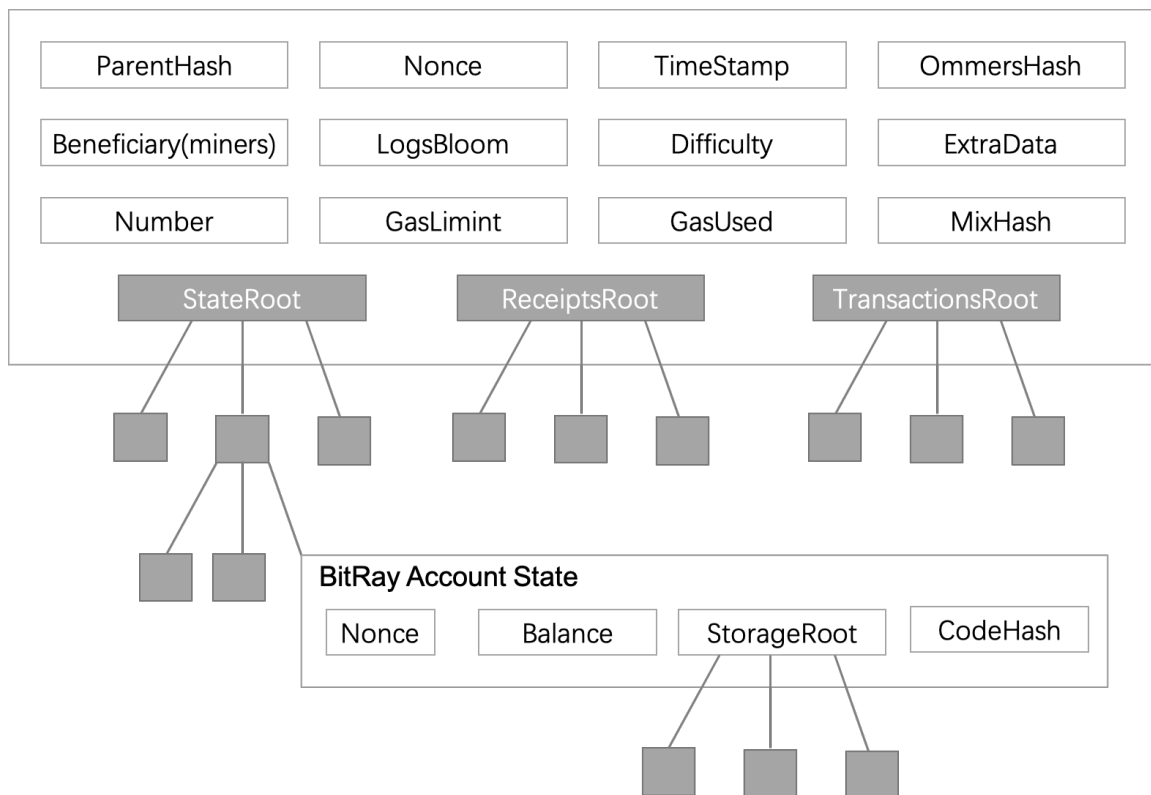


Figure 2.5: BitRay block header

As shown in Fig. 2.5, the same tree structure is used also to store transactions and receipts. More specifically, every block has a “header” which stores the hash of the root node of three different MPT structures, including:

- State tree
- Receipts tree
- Transactions tree

<sup>1</sup><https://ethereum.github.io/yellowpaper/paper.pdf>



Recalling that a blockchain is maintained by a bunch of nodes. In the BitRay network, there are two types of nodes:

- Full nodes
- Light nodes

A full archive node synchronizes the blockchain by downloading the full chain, including all the transactions, from the genesis block to the current head block, executing all of the transactions contained within. Typically, miners store the full archive node, because they need the information in order to proceed with the mining process. While it is also possible to download a full node without executing every transaction, any full node contains the entire chain.

But unless a node needs to execute every transaction or easily query historical data for mining purposes, there's really no need to store the entire chain. This is where the concept of a light node comes in. Instead of downloading and storing the full chain and executing all of the transactions, light nodes download only the chain of headers, from the genesis block to the current head, without executing any transactions or retrieving any associated state. In the BitRay blockchain design, the size of the block head is around 500 byte (0.5 Kb). This makes the whole chain (6 million height after 6 years of running) costing around 300Mb of storage size, which can be easily stored on mobile devices. One of the benefits of storing information in MPT in BitRay network is enabling the type of nodes of what we call "light clients" or "light nodes". Since light nodes have access to block headers, which contain hashes of three tries, they can still easily generate and receive verifiable answers about transactions, events, balances, etc. The reason that this can work is because we hash contents in the MPT upwards, whose validations can be demonstrated by **merkel proof**. Consider the case that a malicious user attempts to modify some states at certain position of a MPT, this change will cause a change in the hash of the node above, which will change the hash of the node above that, and so on, until it eventually changes the root of the tree, rendering the whole state invalidated. Merkle proofs are established by hashing a hash's corresponding hash together and climbing up the MPT until you obtain the root hash which is publicly known by the blockchain. Given that one way hashes are intended to be collision free, no two plain-text hashes can be the same.

### Merkel proof

Merkle proofs are used to decide upon the following factors:

- Decide if the data belongs in the merkle tree
- Decide the validity of data being part of a dataset without storing the whole data set
- Decide validity of a certain data set being inclusive in a larger data set without revealing either the complete data set or its subset.

Any node interested to verify a piece of data can use something called a "Merkle proof" to do so. A Merkle proof consists of:

- A chunk of data to be verified and its hash
- The root hash of the tree
- **Branch**—all of the partner hashes going up along the path from the data chunk to the MPT root.

Any node on the network reading the proof can verify that the hashing for that branch is consistent all the way up the tree, and therefore that the given chunk is actually at that position in the tree, guaranteeing statistical reliability.

In summary, the benefit of using a MPT is that the root node of this structure is cryptographically dependent on the data stored in the tree, and so the hash of the root node can be used as a secure identity for this data. Since the block header includes the root hash of the state, transactions, and

receipts trees, any node can validate a small part of state of BitRay without needing to store the entire state, which can be potentially unbearable in size for mobile device.

### 2.1.3 Gas and payment

One very important concept in BitRay is the concept of fees. There is no free lunch. Every computation or operation that occurs on the BitRay network requires fees—GAS. Gas is the unit used to measure the fees required for a particular computation. Remember that if the client has enough BTRC<sup>2</sup> in their account balance to cover this maximum, then all the operations are good to go. Furthermore, the sender is refunded for any unused gas at the end of the transaction, exchanged at the original rate. In terms of the unit of gas, we refer to the smallest unit as **rb**<sup>3</sup> and there is also the notion of gas price<sup>4</sup> and gas limit. With every transaction, a sender sets a gas limit and gas price. Gas price and gas limit imply the maximum amount of BTRC that the sender is willing to pay for executing a transaction<sup>5</sup>.

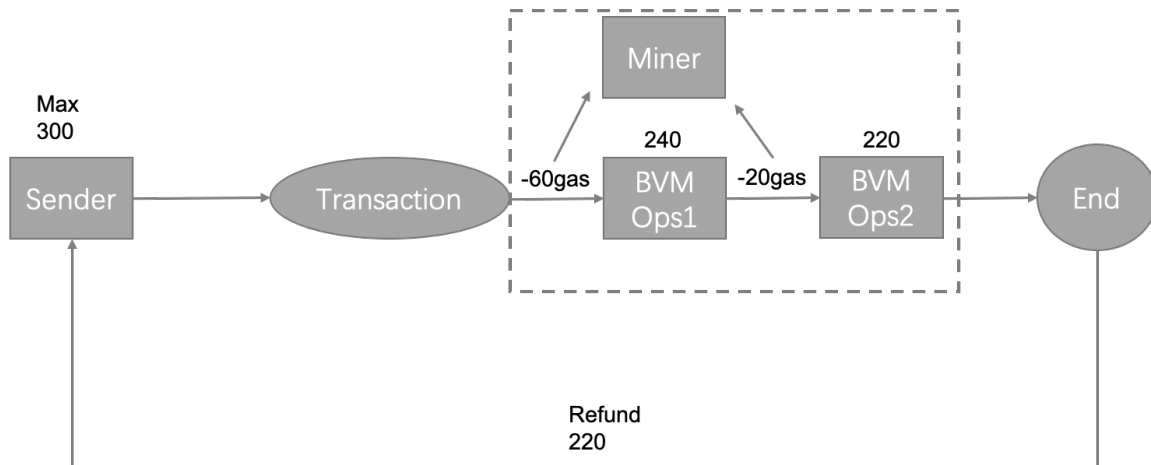


Figure 2.6: Successful transaction with enough gas

In case that the sender does not provide the necessary gas to execute the transaction, the BVM runs into Out of Gas (OOG) exception and the transaction is considered invalid. In this case, the transaction processing aborts and any state changes that occurred are reversed, such that we end up back at the state of BitRay network prior to the transaction, shown in Fig. 2.7. Additionally, a record of the transaction failing gets recorded, showing what transaction was attempted and where it failed, and since the BVM already spent computational power to run the calculations before running out of gas, none of the gas is refunded to the sender. Otherwise, the network could be filled with non-executable transactions fired by some malicious client.

Where does the gas go? Well, as shown in Fig. 2.6, all the money spent on gas by the sender is sent to the miner's address (beneficiary address shown in Fig. 2.5). Since miners are expending the effort to run computations and validate transactions, miners receive the gas fee as a reward. Typically, the higher the gas price the sender is willing to pay, the greater the reward miners derive from executing

<sup>2</sup>In this section we refer BTRC to be one BitRay coin.

<sup>3</sup>"rb" is the smallest unit of BTRC, where  $10^{18}$  rb represents 1 BTRC.  $1\text{grb} = 10^9$  rb.

<sup>4</sup>Gas price is the amount of BTRC you are willing to spend on every unit of gas, and is measured in "grb"

<sup>5</sup>Gas price  $\times$  Gas limit = Max transaction fee

the transaction. Thus encouraging miners to include it in the block. In this way, miners are free to choose which transactions they want to validate or ignore. In order to guide senders on what gas price to set, miners have the option of advertising the minimum gas price for which they will execute transactions.

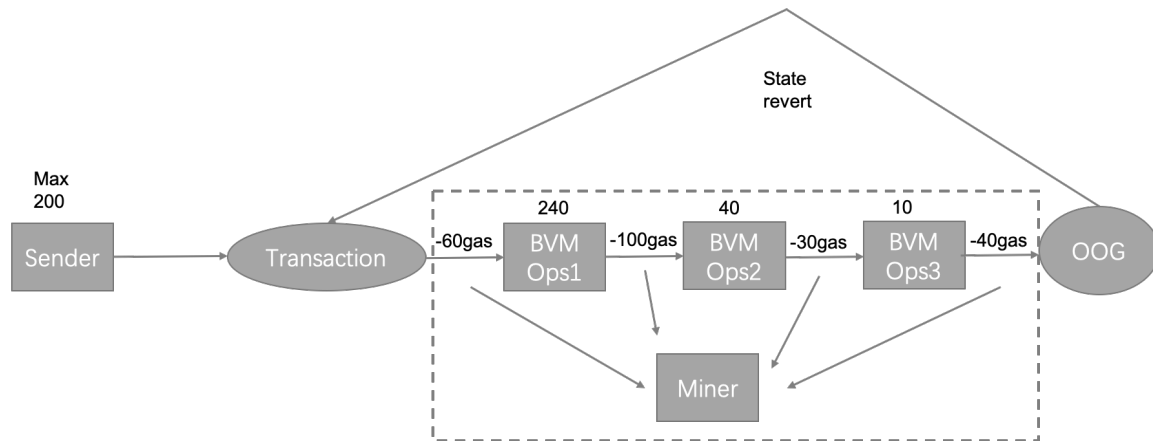


Figure 2.7: Failed transaction with insufficient gas

There are fees for BVM storage too. Not only is gas used to pay for computation steps, it is also used to pay for storage usage. Since the content uploaded to BVM will be stored forever, the cost of storing bytes on BVM will be significantly high. The total fee for storage is proportional to the multiple of every 32 bytes used. In general, there's an incentive to keep the amount of data stored small due to the high cost. We recommend DAPPs to shift their storage to the BitRay storage network, as we would introduce in the next subsection.

### Conclusion of fees

One important aspect of the way the BitRay works is that every single operation executed by the network is simultaneously effected by every full node. However, computational steps on the BVM are very expensive. Therefore, BVM smart contracts are best used for simple tasks, like running simple business logic or verifying signatures and other cryptographic objects, rather than more complex uses, like file storage, email, or AI & machine learning, which can put a strain on the network. Imposing fees prevents users from overtaxing the network.


BitRay is a Turing complete language. A Turing-machine is a machine that can simulate any computer algorithm. This allows for loops and makes BitRay susceptible to the halting problem, a problem in which you cannot determine whether or not a program will run infinitely. If there were no fees, a malicious actor could easily try to disrupt the network by executing an infinite loop within a transaction, without any repercussions. Thus, fees protect the network from deliberate attacks. And just like computing, BVM storage on the BitRay network is a cost that the entire network has to take the burden of. Actually if one computes in details, buying a car could cost less than adding the data for one photo onto the blockchain. Given this challenge of storing data, BitRay network has considered using IPFS to develop its storage network. IPFS helps solving a lot of problems on the current internet. It can also potentially solve a lot of problems for the new decentralized web applications (DAPPs). It turns out that by combining a blockchain with IPFS, we can serve much larger amount of data than we're able to serve by using just the blockchain.

### 2.1.4 Transactions and messages

We noted earlier that BitRay is a transaction-based state machine. That is to say, transactions occurring between different accounts are what move the global state of BitRay system from one state to the next. In the most basic sense, a transaction is a cryptographically signed piece of instruction that is generated by an externally owned account, serialized, and then submitted to the blockchain. There are two types of transactions: message calls and contract creations (i.e. transactions that create new BitRay contracts). All transactions contain the following components:

- **Nonce:** a count of the number of transactions already sent by the sender.
- **GasPrice:** the number of grb that the sender is willing to pay per unit of gas required to execute the transaction.
- **GasLimit:** the maximum amount of gas that the sender is willing to pay for executing this transaction. This amount is set and paid upfront, before any computation is done. And the sender is refunded once the transaction is processed and the remaining gas will be returned.
- **To:** the address of the recipient. In a contract-creating transaction, this field will be empty.
- **Value:** the amount of rb to be transferred from the sender to the recipient. In a contract-creating transaction, this value serves as the starting balance within the newly created contract account.
- **v, r, s:** used to generate the signature that identifies the sender of the transaction.
- **Init** (only exists for contract-creating transactions): An BVM code fragment that is used to initialize the new contract account. `init` is run only once, and then is discarded. When `init` is first run, it returns the body of the account code, which is the piece of code that is permanently associated with the contract account.
- **Data** (optional field for message calls): the input data of the message call. For instance, if a smart contract serves as a domain registration service, a call to that contract might expect input fields such as the domain and IP address pairs.
- **S-Data:** S-Data is *nil* for transactions of external account but will be filled when the transaction is for storage purposes. Then S-Data will be protocol files defined by BitRay network, which is shown in more detail in the next subsection.

We introduced in the "Account" subsection that transactions are always initiated by externally owned accounts and distributed across the blockchain. Another way to think about it is that transactions are communication channels that bridge the external world to the internal state of BitRay network. In the mean time, contracts can talk to other contracts through messages as well. Contracts that exist within the global scope of BitRay's state can talk to other contracts within that same scope. The way they do this is via "messages" or "internal transactions" to other contracts. We can think of messages or internal transactions as being similar to transactions, with the major difference that they are NOT generated by externally owned accounts. Instead, they are generated by contracts. They are virtual objects that, unlike transactions, are not serialized and only exist in the BitRay execution environment. When one contract sends an internal transaction to another contract, the associated code that exists on the recipient contract account is executed.

 One important thing to note is that internal transactions or messages don't contain a `gasLimit`. This is because the gas limit is determined by the external creator of the original transaction (i.e. some externally owned account). The gas limit that the externally owned account sets must be high enough to carry out the transaction, including any sub-executions that occur as a result of that transaction, such as contract-to-contract messages. If, in the chain of transactions and messages, a particular message execution runs out of gas, then that message's execution

will revert, along with any subsequent messages triggered by the execution. However, the parent execution does not need to revert.

### Transaction execution

We've come to one of the most complex parts of the BitRay protocol: the execution of a transaction. Say you send a transaction off into the BitRay network to be processed. What happens to transition the state of BitRay to include your transaction?

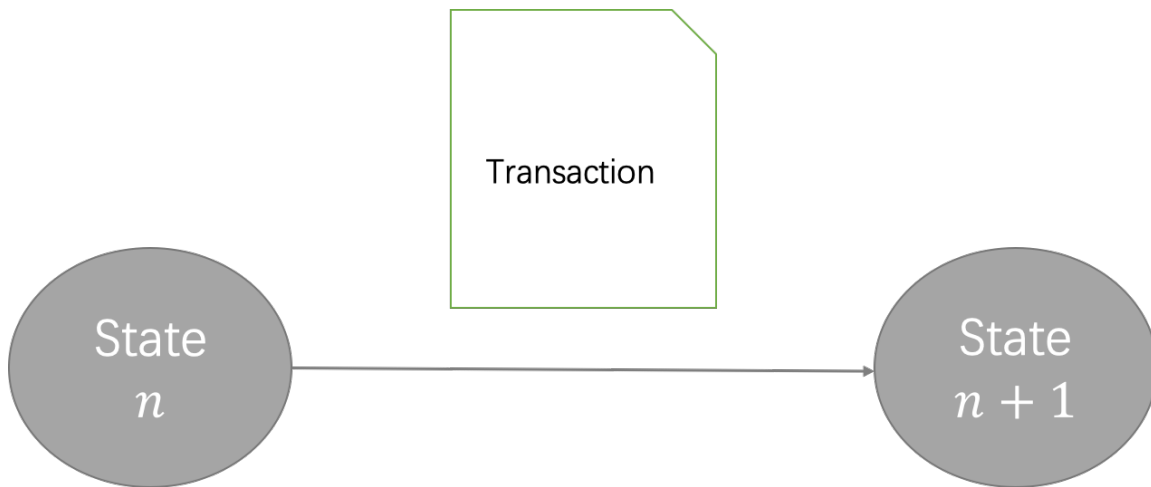


Figure 2.8: State transition to include a transaction

All transactions must meet an initial set of requirements in order to be executed. These include:

- The transaction must be a properly formatted Recursive Length Prefix (RLP). Basically, it is a data format used to encode nested arrays of binary data. RLP is the format Ethereum network (Vitalik 2014) uses to serialize objects.
- Valid transaction signature.
- Valid transaction nonce. To be valid, a transaction nonce must be equal to the sender account's nonce in the transaction pool.
- The transaction's gas limit must be equal to or greater than the intrinsic gas used by the transaction. The intrinsic gas includes:
  - a predefined cost of 21,000 gas for executing the transaction
  - a gas fee for data sent with the transaction (4 gas for every byte of data or code that equals zero, and 68 gas for every non-zero byte of data or code)
- if the transaction is a contract-creating transaction, an additional 32,000 gas will be added.
- The sender's account balance must have enough BTRC to cover the gas costs that the sender must pay. The calculation for the upfront gas cost is defined as the following process: First, the transaction's gas limit is multiplied by the transaction's gas price to determine the maximum gas cost. Then, this maximum cost is added to the total value being transferred from the sender to the recipient.

If the transaction meets all of the above requirements for validity, then we move onto the next steps: First, we deduct the upfront cost of execution from the sender's balance, and increase the nonce of the sender's account by one to account for the current transaction. At this point, we can calculate the gas remaining as the total gas limit for the transaction minus the intrinsic gas used. Next, the

transaction starts executing. Throughout the execution of a transaction, BitRay keeps track of the "substate". This substate is a way to record information accrued during the transaction that will be needed immediately after the transaction completes. Specifically, it contains:

- **Self-destruct set:** a set of accounts (if any) that will be discarded after the transaction completes.
- **Log series:** archived and indexable checkpoints of the virtual machine's code execution.
- **Refund balance:** the amount to be refunded to the sender account after the transaction. PS: a sender is refunded for clearing up storage. BitRay network keeps track of this using a refund counter. The refund counter starts at zero and increments every time the contract deletes something in storage.

Next, the various computations required by the transaction are processed. Once all the steps required by the transaction have been processed, and assuming there is no invalid state, the state is finalized by determining the amount of unused gas to be refunded to the sender. In addition to the unused gas, the sender is also refunded some allowance from the "refund balance" that we described above. Once the sender is refunded: the BTRC for the gas is given to the miner and the gas used by the transaction is added to the block gas counter (which keeps track of the total gas used by all transactions in the block, and is useful when validating a block). Then all accounts in the self-destruct set (if any) are deleted. Finally, we're left with the new state and a set of the logs created by the transaction.

### Contract creation

Recall that in BitRay: there are two types of accounts in the network: contract accounts and externally owned accounts, as shown in Fig. 2.3. When the transaction code segment is non-*nil* we mean that the purpose of the transaction is to create a new contract account. In order to create a new contract account, we first declare the address of the new account and then we initialize the new account by:

- determine the address by some special formula
- set the nonce to zero
- sender send some amount of BTRC as value with the transaction, setting the account balance to that value
- deduct the value added to this new account's balance from the sender's balance
- set the storage as empty
- setting the contract's codeHash as the hash of an empty string

Once we initialized the contract account, we can proceed with creating the account, using the **init** code sent with the transaction. The purpose of the execution of *init* code is varied. Depending on the specific constructor of the contract, it might:

- update the account's storage
- create other contract accounts
- make other message calls
- etc..

As the code to initialize a contract is executed, it uses gas. The transaction is not allowed to use up more gas than the remaining gas. If it does, the execution will hit an OOG exception and exit, as shown in Fig. 2.7. If the transaction exits due to an OOG exception, then the state is reverted to the point immediately prior to transaction. The sender is not refunded the gas that was spent before running out and is given to the miner instead.

If the initialization code executes successfully, a final contract-creation cost is paid. This is a storage cost, and is proportional to the size of the created contract's code. If there's not enough gas remaining to pay this final cost, then the transaction again declares an OOG exception and aborts.

If all goes well and we make it this far without exceptions, then any remaining unused gas is refunded to the original sender of the transaction, and the altered state is now finalized!

### Message calls

The execution of a message call is similar to that of a contract creation, with a few differences. First of all, a message call execution does not include any init code, since no new accounts are being created. It contains input data provided by the transaction sender. Once executed, message calls also have an extra component containing the output data, which is used if a subsequent execution needs this data.

Similar as contract creation, if a message call execution exits because it runs out of gas or because the transaction is invalid (e.g. stack overflow, invalid jump destination, or invalid instruction), none of the gas used is refunded to the original caller. Instead, all of the remaining unused gas is consumed, and the state is reset to the point immediately prior to balance transfer.

In the BitRay network we adapt a new "revert" code that allows a contract to stop execution and revert state changes, without consuming the remaining gas, and with the ability to return a reason for the failed transaction. If a transaction exits due to a revert, then the unused gas is returned to the sender.

#### 2.1.5 BitRay virtual machine (BVM)

In this subsection, we'll look at how the transaction gets executed within the BitRay Virtual Machine (BVM). The part of the protocol that actually handles processing the transactions is BitRay's own virtual machine, known as BVM. The BVM is a Turing complete virtual machine, as defined earlier. The only limitation the BVM has that a typical Turing complete machine does not is that the BVM is intrinsically bounded by gas. Thus, the total amount of computation that can be done is intrinsically limited by the amount of gas provided.

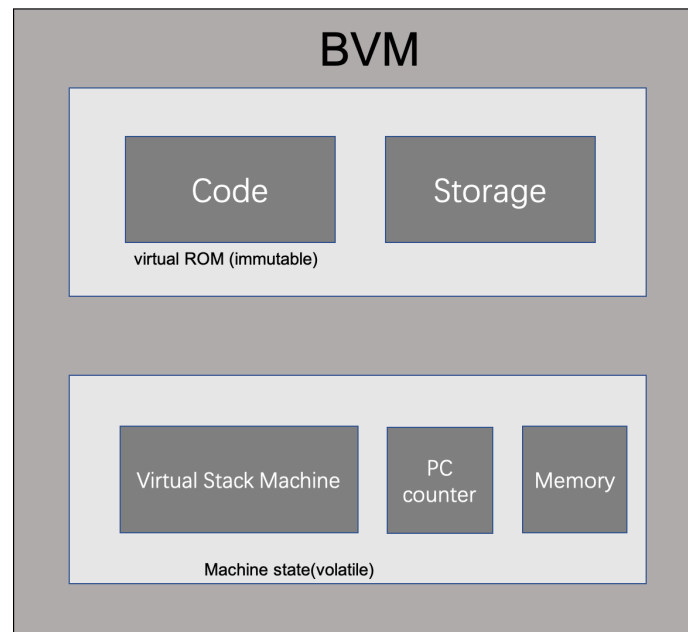


Figure 2.9: BVM diagram



As shown in Fig.2.9, the BVM has a stack-based architecture. A stack machine is a computer that uses a last-in, first-out stack to hold temporary values. The size of each stack item in the BVM is 32 Byte, and the stack has a maximum size of 1024. The BVM has memory, where items are stored as word-addressed byte arrays. Memory is volatile, meaning it is not permanent. The BVM also has storage. Unlike memory, storage is non-volatile and is maintained as part of the system state. The BVM stores program code separately, in a virtual ROM that can only be accessed via special instructions.

The BVM also has its own set of instructions: the BVM bytecode. It's compiled from highlevel languages such as solidity. Before executing a particular computation, the processor makes sure that the following information is available and valid:

- System state
- Remaining gas for computation
- Address of the account that owns the code that is executing
- Address of the sender of the transaction that originated this execution
- Address of the account that caused the code to execute (could be different from the original sender)
- Gas price of the transaction that originated this execution
- Input data for this execution
- Value (in ether) passed to this account as part of the current execution
- Machine code to be executed
- Block header of the current block
- Depth of the present message call or contract creation stack

The BVM then executes the transaction recursively, computing the system state and the machine state for each loop. The system state is simply BitRay's global state. The machine state is comprised of:

- gas available
- program counter
- memory contents
- active number of words in memory
- stack contents.

Stack items are added or removed from the leftmost portion of the series. On each cycle, the appropriate gas amount is reduced from the remaining gas, and the program counter increments. At the end of each loop, there are three possibilities:

- The machine reaches an exceptional state (e.g. insufficient gas, invalid instructions, insufficient stack items, stack items would overflow above 1024, invalid JUMP/JUMPI destination, etc.) and so must be halted, with any changes discarded
- The sequence continues to process into the next loop
- The machine reaches a controlled halt (the end of the execution process)

Assuming the execution doesn't hit an exceptional state and reaches a "controlled" or normal halt, the machine generates the resultant state, the remaining gas after this execution, the accrued substate, and the resultant output.

## 2.2 The storage Network

BitRay storage is a proposed decentralized file system designed to give everyone the ability to permanently store and host files accessible by any web browser. Unlike some other proposed alternatives such as filecoin, there would be no upfront fee or ongoing charge for storage bandwidth on BitRay network, aside from a completely refundable deposit. Users must hold tokens while they need bandwidth and may sell tokens when bandwidth is no longer required. Built on the InterPlanetary File System (IPFS) and the BitRay storage protocol, BitRay storage will be a service provided by block producers for those who hold tokens on a blockchain that adopts the BitRay protocol. The block producers would be incentivized to replicate and host these files, allowing anyone with an Internet browser to access them. With BitRay storage protocol the network can provide high bandwidth serving of videos, music, images and tons of decentralized applications. Promoting the development and ecosystem of blockchain 3.0.



This section will be a high level overview to introduce readers about BitRay storage network. A more technical guide on these technologies will be coming out in the near future.

### 2.2.1 The Decentralized Power of BitRay and IPFS

#### Limitations of blockchains

As we introduced in the first section, Blockchains are terrible at storing large amount of data. They're really bad at it. It might cost you thousands of dollars to just store a small picture. This reason is that, it takes a lot to duplicate data across thousands of machines. When a piece of data gets added to blockchain, the new data becomes part of a block. This means that thousands of computers around the world are working together to verify that data by running the consensus engine, preventing scalability. In the context of a blockchain, consensus means that these computers agree that the data for a given block is valid, and it's time to move onto the next block.

#### IPFS's potential

IPFS is a distributed system for storing and accessing files, websites, applications, and data. IPFS helps solve a lot of problems on the modern web. It also solves a lot of problems for the new decentralized web! It turns out that by combining a blockchain with IPFS, we can timestamp much larger amounts of data than we're able to by using just the blockchain. When data is added to IPFS, the protocol returns a hash of the data that was just added. This hash is cryptographically guaranteed to be unique to the content. For instance, if we are visiting <https://en.wikipedia.org/wiki/Aardvark>, the IPFS will search the following instead:

`/ipfs/QmXoybizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Aardvark.html`

In other words, IPFS allows us to store data and then later retrieve it, with the knowledge that our data wasn't tampered with. That's pretty powerful. Also, IPFS is **decentralized**, making it possible to download a file from many locations that aren't managed by one organization such as centralized storage providers(google drive, dropbox, iCloud), with the advantages of:

- **Supports a resilient internet.** If someone attacks Wikipedia's web servers or an engineer at Wikipedia makes a big mistake that causes their servers to catch fire, you can still get the same webpages from somewhere else.
- **Makes it harder to censor content.** Because files on IPFS can come from many places, it's harder for anyone (whether they're states, corporations, or someone else) to block things. We hope IPFS can help provide ways to circumvent actions like these when they happen.

- **Can speed up the web when you're far away or disconnected.** If you can retrieve a file from someone nearby instead of hundreds or thousands of miles away, you can often get it faster. This is especially valuable if your community is networked locally but doesn't have a good connection to the wider internet.

What's even more motivating is when we combine this technique with BitRay's current components. Recall how the blockchains are really bad at doing this with large amounts of data, we can get around that limitation using IPFS. technology. Hashes provided back from IPFS are currently 46 bytes in size, which is 1/43478 the size of our 2MB photo from earlier. This means that an IPFS hash is magnitudes cheaper to store on Ethereum than the data itself. Therefore, if we wanted to timestamp a large piece of data, all we would have to do is upload that data to IPFS and then put the corresponding hash onto BitRay smart contracts.

### A real example

There are a lot of potential applications by properly combine these two subsystems of BitRay blockchain and IPFS. We introduce one example we found on medium post:

Let's say that a company signed a rather large legal agreement. The signed PDF of that agreement was stored on IPFS and the corresponding hash was printed onto the blockchain. Now, let's say there was a dispute where one party claimed the agreement had never been signed. This could easily be disproven by referencing the block where the hash of the agreement was added to the blockchain. If all we had was the IPFS hash of the agreement, it could easily be claimed that the document was forged. After all, the document could have been added to the IPFS network at any time, not necessarily when the truthful party claimed it was. But, since the hash was added to the blockchain at the time of upload to the IPFS network, the signed document was timestamped in a tamperproof way. And, because IPFS provides tamper-proof content retrieval, it can be proven that the content the hash points to hasn't been altered in any way.

### Challenge

However, problem remains. IPFS does not guarantee the availability of files. A file may disappear if nodes deliberately decline to make it available. An inaccessible file may ultimately break the utility and purpose of a smart contract as parties are no longer able to verify the meaning of the file. For instance, let's consider a smart contract that reference a will by its IPFS name. That contract may fail if the file containing the will is unavailable. Smart contracts cannot simply store IPFS filenames and be confident that the file will always be accessible upon demand. We will introduce how we solve this issue in the following two subsections.

### The design of BitRay storage

In the following two subsections we introduce the implementation of BitRay Storage. We assume there is pre-built smart contract in the BVM, which allows the possibility to build our storage protocol such as allowing user to define directory structure and allowing the system to define a bandwidth model.

A user creates a link to an IPFS file by signing a transaction that is broadcasted to the blockchain, including all the information of the file. The user specifies the number of nodes they want the network to hold files. The more nodes they specify, the more likely that their files will be resilient (some nodes fails but they would still be able to retrieve the file from other nodes).

The user will then upload the file to one of the block producers via standardized APIs defined by the BitRay Network. Once the storage miner verifies the file, the miner will broadcast a transaction to the blockchain that files have been received and proper *proof-of-storage* will be generated periodically

to prove that the files are correctly stored.

### 2.2.2 Storage market

The BitRay storage market executes the *put* protocol<sup>6</sup>. In the BitRay storage network protocol, storage providers must convince their clients that they have stored the data they were paid to store. To ensure this, storage providers will generate Proofs-of-Storage (PoS) that the blockchain network (or the clients themselves) verifies. There are various types of PoS and in our protocol we adapt the implementations for the Proof-of-Replication (PoRep) and Proof-of-Spacetime (PoSt) schemes used in Filecoin (protocol labs, 2017).

In the storage network, any user can participate as a *client* or a *storage miner*.

- Clients pay to store data in the network, via Put requests.
- Storage Miners provide data storage service to the storage network. Storage Miners participate in BitRay by offering their disk space and serving Put requests<sup>7</sup>. Storage Miners respond to Put requests by committing to store the client's data for a specified time. Storage Miners generate Proofs-of-Spacetime and submit them to the blockchain to prove to the Network that they are storing the data through time. Storage miners are also eligible to mine new blocks via proof-of-work, and in doing so they hence receive the mining reward for creating a block and transaction fees for the transactions included in the block.

In order to allow clients to pay storage miners for their services, Demand and supply of storage meet at the storage markets. The market is decentralized exchanges and will be explained in more detail in our protocol paper. In brief, clients and miners set the prices for the services they request or provide by submitting bid/ask orders to the respective markets. The exchanges provide a way for clients and miners to see matching offers and initiate deals. By running the network protocol, the network can guarantee that miners are rewarded and clients are charged if the service requested has been correctly and successfully provided.

### 2.2.3 Retrieval market

A person who uploads and stores a file is likely very different from the individual who download the file. Consider the example of DTube where someone uploads a home movie which is then viewed by millions of people. The publisher of the video does not want or is unable to pay for the bandwidth consumption of a million viewers. Therefore, in this situation it would be ideal for each individual to pay for their own bandwidth. Once again this is a situation where micro-payments are not a convincing solution due to the cost of the transaction (physically and mentally), which becomes an obstacle that will hinder user's adoption. That being said, it should be entirely reasonable for all users to lock up (*proof-of-lock* (PoL)) enough BitRay tokens to permanently cover all of their bandwidth needs, without feeling like they are being charged per view. When they feel they no longer need the bandwidth, they can send messages to the network to claim their locked tokens back and realise their specific amount of bandwidth<sup>8</sup>. CRR means the bandwidth will never be completely

<sup>6</sup>Put(data) → key: Clients execute the *Put* protocol to store data under a unique identifier key, which is used for later retrieval of the file

<sup>7</sup>To prevent attack, we require in the protocol that as storage miners, they must pledge their storage by depositing collaterals proportional to it. In case of invalid or missing proofs, Storage miners are penalized and lose part of their collaterals.

<sup>8</sup>The amount of bandwidth available per BitRay coin is determined using the Bancor algorithm that maintains a constant reserve ratio (CRR) larger than 1.

Bandwidth Price =  $\frac{\text{balance}}{\text{network supply} \times \text{CRR}}$

consumed, the miners may adjust CRR or total network supply but may never decrease the storage supply below what has already been claimed.

In a brief review, we claim that existing decentralized solutions all rely on micro-payments, but this is likely not sustainable as it requires clients to trust a 3rd party storage providers and it prevents usage in the following two major aspects:

- Clients always have to trust the storage providers to honest serve the file, if they declined to serve some pieces, clients lose their payment in the process.
- Micro-payments create transaction friction which discourages adoption. In practice we typically see strong consumer resistance to micro-payments in favor of flat fee or one-time payments.

Since the cost for using BitRay network bandwidth will always be 0, since clients can freely lock or unlock their share of BitRay coins to gain the service, this solved the two major concerns mentioned above. However, there is no such thing as a free lunch, so who is actually paying for the storage and bandwidth provided by the block producers? All BitRay holders will be paying for this via a portion of the 10% annual inflation. More specifically, those who will be storing files are exposed to this inflation as they are unable to sell their coins until they realise their bandwidth.

## 2.3 Network unification and mining economics

The security of BitRay network is guaranteed by *proof-of-work* (PoW). Miners compete with each other to make sure enough computational power has been dedicated to the network. The storage services are provided by proof-of-storage (PoS) and proof-of-lock (PoL). Storage miners are paid when the pledge to store clients' data and they are also paid by serving clients' retrieval request. In the BitRay network, (45% of block rewards go to PoW workers while 55% of block rewards go to storage workers (35% for retrieval and 20% for storage)).

How do we encourage PoW workers to include transactions in the storage network into the block, so that clients and miners can proceed without delay? A preferable solution would be to include gas in the storage transaction, which will go to storage miner in the block finalization step. PoW miners will try to include as many storage transactions as possible in their block finalization.

In the BitRay storage network, we discuss separately the following two aspects:

- Storage economics
- Bandwidth economics

### Storage economics

Storage miners earn their rewards mainly through the BTRC clients who pay for their storage. In the mean time, they receive rewards from the network for pledging their storage (in case there is very small storage demand they can still earn rewards). Generally they receive less from the network compared to bandwidth miners, who receive rewards solely from the BitRay network.

### Bandwidth economics

As mentioned previously, all BitRay holders will be paying for this via a portion of the 10% annual inflation, which is actually the case in most other cryptocurrencies. We make a step forward by using certain portion of the mining power in useful things—providing storage. More specifically, those who will be downloading files are exposed to this inflation as they are unable to trade their coins unless they realise their bandwidth. As long as the rate of the new retrieval requests is locking up faster than the BitRay inflation rate, the BitRay effectively undergo monetary deflation thus

increasing the price of BTRC. This will in turn increase the value of the BitRay coin paid to the miners and encourage them to expand the supply of storage.

In an unusual event when there is a significant reduction in retrieval demand, unlocked BTRC will enter the market causing effective price inflation. Alternatively, they could lower the CRR to increase the amount of BTRC that must be locked up to enjoy the bandwidth capacity.

The bottom line is that, those who require storage pay for it through the time-value of money. This shouldn't be implemented by micro-payments, which might result in transaction friction and surprise fees.

## 2.4 Conclusion

The BitRay network is designed from experience with proven concepts, best practices and fundamental advancements in blockchain technology. The chapter is a blueprint for a globally scalable blockchain society in which decentralized applications can be easily deployed and governed. Moreover, the BitRay storage network has the potential to fundamentally change the decentralized storage market by revolutionizing the economic model, eliminating the compromised design of micro-payments and the perception of cost will enable many more innovative applications such as decentralized video hosting and sharing. To the best of our knowledge, these types of DApps are commercially impossible in currently blockchain solutions. BitRay network is there to be one of the first few platforms to offer these services competitively, pushing the technology developments in the blockchain industry to version 3.0.

## 3. Economics Model

### 3.0.1 Initial BitRay token distribution

The initial supply of BitRay token is set to be one billion units, which is to be issued in the ethereum network. After the go implementation of BitRay is ready, initial tokens would be snapshot and mapped to the BitRay main network.

Of the initially created tokens, as shown in Fig. 3.1, 50% will be sold to the public in 2021 after the release of the initial token. Revenue from this public sale funds the operation of the BitRay Foundation, including system development, marketing, financial and legal consulting. 10% BitRay coins will be allocated to the founding team, 30% will be reserved for BitRay foundation for network management and stability guarantee. 10% of BitRay coins will be for research on existing and future blockchain technology.

### 3.0.2 Initial BitRay token distribution usage

#### team

10% of the token will be allocated to the 'development fund' which consisted of early contributors to the project and developers. The token release is of quarterly basis. It will be linearly unlocked each quarter during 2021 and 2022 as rewards for contributions team members made to the project. That being said, approximately 12.5 million tokens will be released each quarter to be shared among team members.

#### network management and stability guarantee

In case of future updates of the network, 30% of the initial tokens will be reserved for the BitRay foundation. This will be kept for potential voting mechanisms and proof-of-stake, which require a fair amount of tokens to participate.

#### Research

The BitRay team always keep an open mind and are willing to learn new and useful techniques and incorporate into our network. This amount of tokens will be used to motivate the BitRay community to reasearch better frameworks of the ecosystem as a whole.



**presale**

The public pre-sale will be used for commercial and community development, education, and market expansion. These revenue will be used for business development, including expansion of industry-related applications, supporting DApp (Distributed Application) development, business expenses (legal, compliance, accounting, consulting), marketing and public relations, and token swaps. It will also be used to support developer education, promotion of BitRay chain technology (team members joining summit), and cooperation with the open-source community.

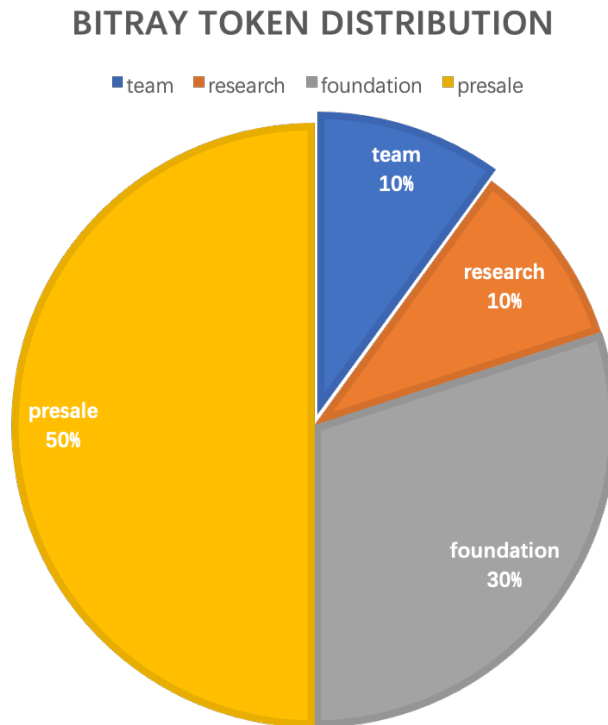


Figure 3.1: Distribution diagram

## 4. Implementation and Iteration

### 4.0.1 MileStones

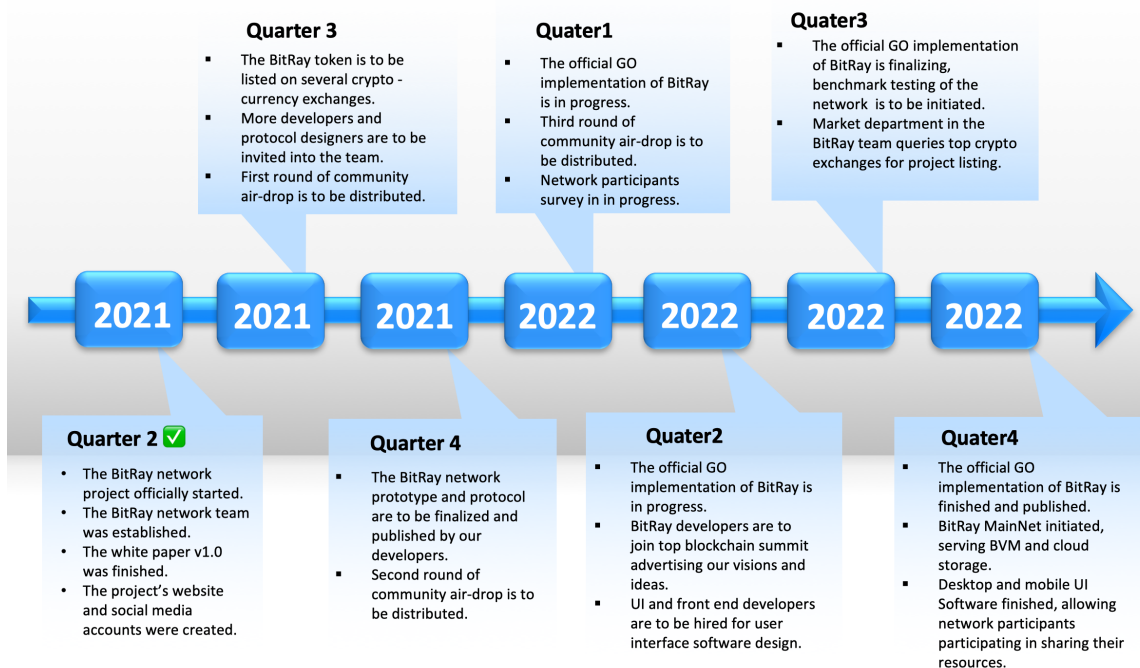


Figure 4.1: BitRay network milestones