International Journal of Trend in Scientific Research and Development (IJTSRD)
International Open Access Journal
ISSN No: 2456 - 6470 | www.ijtsrd.com | Volume - 2 | Issue – 3

# Blockchain Beyond Cryptocurrencies

**[1]Pradeep Murugan, [1]Suraj Subramanian, [2]Mr. V Pandarinathan, [3]Dr. D. Rajinigirinath**
[1]Student, [2]Assistant Professor, [3]Head of the Department
Department of CSE, Sri Muthukumaran Institute of Technology, Chennai, Tamil Nadu, India

## ABSTRACT

A new model for building massively scalable and profitable applications is emerging. Bitcoin paved the way with its cryptographically stored ledger, scarce-asset model, and peer-to-peer technology. These features provide a starting point for building a new type of software called decentralized applications, or dapps. They are more flexible, transparent, distributed, resilient, and have a better incentivized structure than current software models. Centralized systems are currently the most widespread model for software applications. Centralized systems directly control the operation of the individual units and flow of information from a single center. Blockchain, a massively replicated database of transactions that's able to avoid Sybil attacks. For the first time, the blockchain lets us achieve decentralized consensus without the use of a centralized server.

***Keywords:*** *Blockchain, Decentralized applications, Proof of work, Cryptography, Smart contracts*

## 1. INTRODUCTION:

Most people are familiar with the term "application" as it pertains to software. A software application is software that defines a specific goal. There are millions of software applications currently in use, and the vast majority of web software applications follow a centralized server-client model. Some are distributed, and a select few novel ones are decentralized. Distributed means computation is spread across multiple nodes instead of just one. Decentralized means no node is instructing any other node as to what to do. A lot of Stacks such as Google have adopted a distributed architecture internally to speed up computing and data latency. This means that a system can be both centralized and distributed.

### *2. Decentralized Applications*

### *2.1 History*

In its early days, the Web was obviously not as useful as it is today with the array of apps and services that do everything under the sun, but it did have a more DIY distributed feel to it. The Web was pretty decentralized from the outset. The HTTP protocol connected everyone on the planet with a computing device and an Internet connection. In the HTTP protocol guidelines, there are a set of trusted servers that translate the web address you enter into a server address. Furthermore, HTTPS adds another layer of trusted servers and certificate authorities. People would host personal servers for others to connect to, and everyone owned their data. But soon, application servers began taking off and the centralized model of data ownership as we know it today was born. Why did it happen this way?

The simple answer is because it was easy, both conceptually and programmatically. It was the easiest thing to do and it worked. One individual or group pays for maintenance of a server and profits from the users that utilize the software on it. Apps like MySpace and Yahoo! were among the first popular centralized apps. More recent apps like Uber and Airbnb decentralize the "real-world" parts of a business by providing a central and trusted data store. They are among the first to allow for participation in one moneymaking endeavor from all sides of the economy. Their decentralized business model

foreshadows the development of even more decentralized apps.

As the HTTP web grew larger, a new protocol was introduced by a developer named Bram Cohen, called *BitTorrent*. BitTorrent was a protocol created as a solution to the lengthy time to download huge media files via HTTP and as an improvement on some of the P2P proposals before it, like Gnutella, Napster, and Grokster. The problem was that downloading huge files took a very long time and as the Web grew, so did the size of files that were available. Meanwhile, hard-drive space was increasing and more people were connected. BitTorrent solved this by making downloaders into uploaders, as well.

If there was a file you wanted, you would download it from not one, but multiple sources. The more popular the file, the more users who would be downloading it and subsequently uploading it, which meant you would be pulling from multiple sources. The more sources, the faster the download. Seeders were rewarded with faster download speeds, whereas leechers were punished with limited speeds. This tit-for-tat system of transferring data proved to be very useful for large media files like movies and TV shows.

BitTorrent grew and is for many the de facto way to download any sort of large media file like a game or movie. BitTorrent's speed, resilience, and reward mechanism proved to be better than HTTP for large data sets.

*So, why doesn't the Web work this way?*

Most likely because of HTTP's first mover advantage, its infrastructure, and all of the time and money already invested in it. There are currently active projects working on upgrading the HTTP web with BitTorrent-like technology, and they'll most likely be successful because of BitTorrent's huge value proposition. As soon as BitTorrent was introduced, developers began to use the technology to create nonprofit decentralized applications.

Centralized systems are currently the most widespread model for software applications. Centralized systems directly control the operation of the individual units and flow of information from a single center. All individuals are directly dependent on the central power to send and receive information and to be commanded. Facebook, Amazon, Google, and every other mainstream service we use on the Internet uses this model. The dapp space is currently an emerging field with a lot of smart people still experimenting with new models. Different developers have different opinions on what exactly a dapp is. Some developers think that having no central point of failure is all it takes and some think that there are other requirements. The reason for the profit focus is because profit is the cornerstone of a successful, robust, and sustainable dapp. Incentives keep developers building, users loyal, and miners maintaining a blockchain.

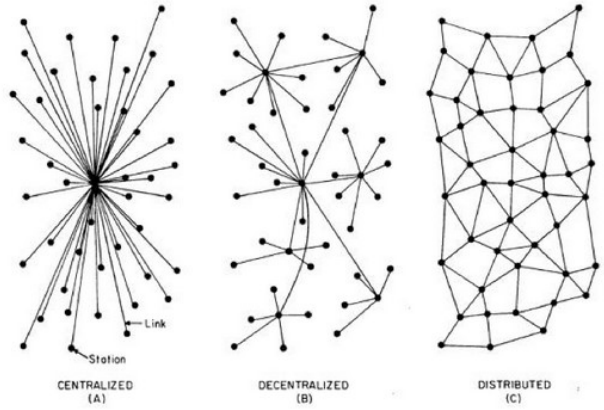


Figure 2.1 Centealized, Decntralized and Distributed

***3. Proof of Work***

Bitcoin and other cryptocurrencies will help define the fifth protocol layer of the Internet, letting machines transfer value as fast and efficiently as data. Bitcoin is a useful tool for online value transfer, but its most valuable innovation is its underlying technology, the *blockchain*, that for the first time in history made decentralized consensus possible.

The blockchain is a massively replicated database of all transactions in the Bitcoin network. It uses a consensus mechanism called proof-of-work which prevents double-spending in the network—a problem that had plagued cryptographic researchers for decades. Double-spending meant a bad actor could spend the same funds twice, denying the first transaction happened.

Proof-of-work solves this problem by having *miners* in the network solve cryptographic proofs using their hardware. Miners are Bitcoin nodes that verify a transaction and check it via its blockchain history, a timestamped record of all transactions ever made in the network. Someone could theoretically alter their

blockchain history, but with proof-of-work, they would also need to have the majority of computational power in the network to verify it. Because the Bitcoin network has much more computation power at this point than all of the world's supercomputers combined, an attacker would have an extremely difficult time trying to break the network.

Proof-of-work is expensive in terms of the cost of electricity and compute workload but it's the only known prevention mechanism against Sybil attacks, in which a bad actor claims to be multiple people in a network and gains resources that they shouldn't by doing so. A successful Sybil attack on the Bitcoin network would most likely result in a complete devaluation of the currency because people would no longer trust its stability. As expensive as proof-of-work is, it's the only thing that's proven to work so far on a massive scale.

So, we have this new tool called the blockchain, a massively replicated database of transactions that's able to avoid Sybil attacks. For the first time, the blockchain lets us achieve decentralized consensus without the use of a centralized server. You might be wondering what use cases this would have, and rightly so. I'm going to be devoting a good portion of the book to helping you think about all of the possibilities and ways with which you could implement them. The important bit for now is to understand that this data structure is one of many that will help you to create profitable decentralized applications.

### *4. Cryptography*

Blockchain systems make heavy use of cryptographic techniques to ensure integrity of the ledgers. Integrity here refers to the ability to detect tampering of the blockchain data. This property is vital in public settings where there is no pre established trust. For example, public confidence in crypto-currencies like Bitcoin, which determines values of the currencies, is predicated upon the integrity of the ledger; that is the ledger must be able to detect double spending. Even in private blockchains, integrity is equally essential because the authenticated nodes can still act maliciously. There are at least two levels of integrity protection. First, the global states are protected by a hash (Merkle) tree whose root hash is stored in a block. Any state change results in a new root hash. The tree's leaves contain the states, the internal nodes contain the hashes of their children. For instance, Hyperledger v0.6 uses a bucket hash tree, in which states are grouped (by hashing) into a pre defined number of buckets. Ethereum, on the other hand, employs a Patricia- Merkle tree which resembles a trie and whose leaves are key-value states. Second, the block history is protected, that is the blocks are immutable once they are appended to the blockchain. The key technique is to link the blocks through a chain of cryptographic hash pointers: the content of block number n + 1 contains the hash of block number n. This way, any modification in block n immediately invalidates all the subsequent blocks. By combining Merkle tree and hash pointers, blockchain offers a secure and efficient data model that tracks all historical changes made to the global states.

Blockchain's security model assumes the availability of public key cryptography. Identities, including user and transaction identities, are derived from public key certificates. Secure key management, therefore, is essential to any blockchains. As in other security systems, losing private keys means losing access. But in blockchain applications such as crypto-currencies, losing the keys has direct and irrevocable financial impact. We discuss in Section 4.2 different schemes for key and identity management. There exist many research systems that extend the original blockchain design with novel and complex cryptographic protocols. They aim to improve security and performance with esoteric techniques such as zero-knowledge proofs, group signatures and trusted hardware.

### *5. Smart Contracts*

A smart contract refers to the computation executed when a transaction is performed. It can be regarded as a stored procedure invoked upon a transaction. The inputs, outputs and states affected by the smart contract execution are agreed on by every node.

```
contract Doubler {
struct Partitipant {
address etherAddress;
uint amount;
}
Partitipant[] public participants;
uint public balance = 0;
...
function enter() {
...
balance+= msg.value;
...
```

```
if (balance >
2*participants[payoutIdx].amount) {
transactionAmount = ...
participants[payoutIdx].
etherAddress.send(transactionAmount);
...
}
}
...
}
```

An example of Ethereum smart contract, written in Solidity, which implements a pyramid scheme. All blockchains have built-in smart contracts that implement their transaction logics. In crypto currencies, for example, the built-in smart contract first verifies transaction inputs by checking their signatures. Next, it verifies that the balance of the output addresses matches that of the inputs. Finally, it applies changes to the states. In the rest of the paper we do not refer to such built-in logics as smart contracts. Instead, we only consider smart contracts that can be defined by users. One way to characterize a smart contract system is by its language. At one extreme, Bitcoin provides fewer than 200 *opcodes* from which users can write stack-based scripts. For example, the following script verifies if 2 out of 3 valid signatures are available.

At the other extreme, Ethereum smart contracts can specify arbitrary computations, i.e. they are Turing complete code. Figure 2 shows a snippet of a real smart contract running on Ethereum. It implements a pyramid scheme: users send money to this contract which then pays interests to early participants. The contract has its own states, namely the list of participants, and exports a function called enter. A user invokes the contract by sending his money through a transaction. When executed, the contract can access the input address (user account) via msg.sender and the transaction value via msg.amount. It updates the accumulated balance, computes the interest for each participants. Finally, payment is made by invoking etherAddress.send. In between the two extremes are smart contract systems. that offer more expressiveness than Bitcoin's opcodes, but they reject Turing-completeness. Kadena and BigchainDB support contracts with complex, but constrained semantics so that they can be formally checked for safety. Another way to categorize smart contract systems is by their runtime environments. Most systems execute smart contracts in the same runtime as the rest of the blockchain stack. We refer to them as employing native runtimes. For example, Kadena parses contracts written in its Haskell-like language and executes them directly as Haskell programs. Ethereum, on the other hand, comes with its own virtual machine for executing Ethereum bytecodes.

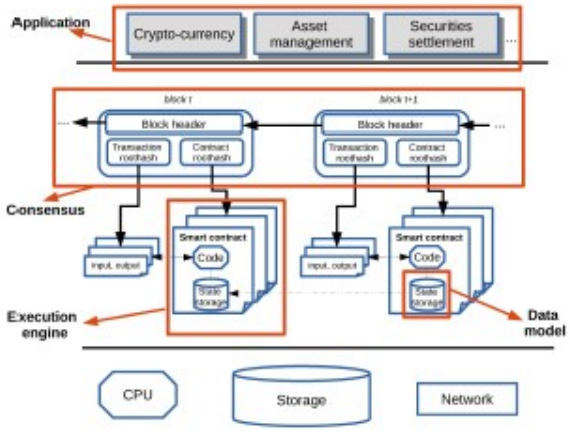


Figure 5.1 Smart contracts

### *6. Network*

The steps to run the network are as follows:

1 New transactions are broadcast to all nodes.
2 Each node collects new transactions into a block.
3 Each node works on finding a difficult proof-of work for its block.
4 When a node finds a proof-of-work, it broadcasts the block to all nodes.
5 Nodes accept the block only if all transactions in it are valid and not already spent.
6 Nodes express their acceptance of the block by working on creating the next block in the
7 chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one. New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block

broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

### *7. Incentive*

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them.

The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended. The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free. The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

### *8. Reclaiming Disk Space*

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree, with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.

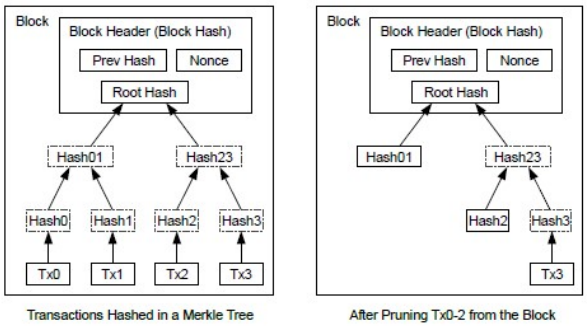


Figure 8.1 Reclaiming Disk Space

A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 8GB of RAM as of 2018, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

### 9. Conclusion

In this paper, we have conducted a comprehensive survey on blockchain technologies. We laid out four underpinning concepts behind blockchains and analyzed the state of the art using these concepts. We presented the possibility of using blockchain for decentralized applications which has many use cases. We hope that the survey would serve to guide the design and implementation of future blockchain systems that are not only secure, but scalable and usable in the real world.

### *References*

1) S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

2) Q. Lin, P. Chang, G. Chen, B. C. Ooi, K. Tan, and Z. Wang, “Towards a non-2pc transaction management in distributed database systems,” in *Proceedings of ACM International Conference on Manage- ment of Data (SIGMOD), San Francisco, CA, USA*, 2016, pp. 1659– 1674.

3) A. Thomson, T. Diamond, S. Weng, K. Ren, P. Shao, and D. J. Abadi, “Calvin: fast distributed transactions for partitioned database systems,” in *Proceedings of ACM international Conference on Management of Data (SIGMOD), Scottsdale, AZ, USA*, 2012, pp. 1–12.

4) P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica, "Coordination avoidance in database systems," *PVLDB*, vol. 8, no. 3, pp. 185–196, 2014.

5) "Ethereum blockchain app platform," https://www.ethereum.org/.

6) Ripple, "Ripple," https://ripple.com.

7) Melonport, "Blockchain software for asset management," http://melonport.com.

8) J. Morgan and O. Wyman, "Unlocking economic advantage with blockchain. a guide for asset managers." 2016.

9) G. S. Group, "Blockchain: putting theory into practice," 2016.

10) T. T. A. Dinh, J. Wang, G. Chen, L. Rui, K.-L. Tan, and B. C. Ooi, "Blockbench: a benchmarking framework for analyzing private blockchains," in *SIGMOD*, 2017.

11) Ethcore, "Parity: next generation ethereum browser," https:// ethcore.io/parity.html.

12) Hyperledger, "Blockchain technologies for business," https://www.hyperledger.org.