IoT Chain: A High-Security Lite IoT OS (Technical Paper)

draft content will change according to the implementation*

IoT Chain Team

February 11, 2018

1 Directed Acyclic Graph

A Directed Acyclic Graph, or DAG for short, is a graph with edges and vertices that is both directed (meaning all the edges have directionality from one vertex to another) and acyclic (meaning there is no graph cycle that loops around a vertex for every vertex). In the context of blockchain, it is a generalization of blockchain in the context of Bitcoin in that DAG allows a different structure altogether as follows.

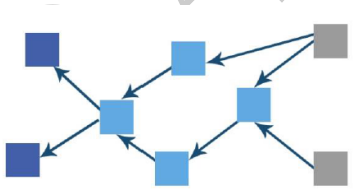

Figure 1: DAG structure

Note that in reality, traditional Proof-of-Work blockchain as in Bitcoin's blockchain admits numerous orphaned blocks. Hence, a more realistic representation of blockchain versus DAG can be shown by the following.

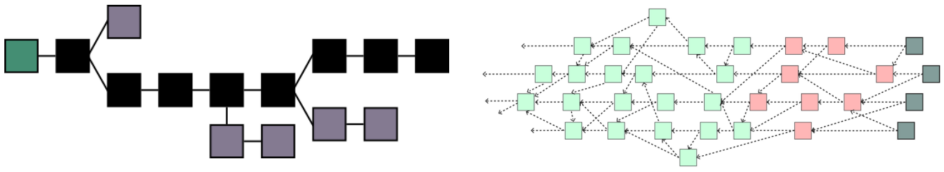

Figure 2: Realistic representation of blockchain versus DAG

Due to the rigid linked-list data structure of blockchain itself, Bitcoin's transaction performance has become worse while average transaction fees have become way too high as the network suffers scalability issues. On the other hand, DAG is a distributed architecture that has a better potential to overcome such scalability issues. Compared with Bitcoin's longest-chain consensus protocol, DAG itself as a data structure, could be tailored to different nonspecific consensus algorithms. As a result, DAG not only have the potential for solving the present problem of mining centralization, but also greatly improve the whole distributed network's capacity and scalability, lowering transaction cost while maintaining the timeliness of transactions.

DAG is also quantum-resistant. For instance, the known complexity of $\Theta(\sqrt{n})$ via quantum computing contrasts with that of $\Theta(n)$ via classical computing. Here, note that the big Θ notation refers to the following definition corresponding to the intersection between the big O notation and the big Ω notation:

$$f(n) = \Theta(g(n)) \iff \exists k_1, k_2, n_0 > 0 \text{ such that } \forall n > n_0 : \ k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n)$$

In other words, the complexity $\Theta(\sqrt{n})$ means that it is asymptotically bounded above and below by the function $\sqrt{n}$ as $n \to \infty$. The difference between quantum computing and classical is indeed drastic if one takes the case of Bitcoin and substitutes, at the time of writing, $n = 2^{73}$. Then $\sqrt{n} = \sqrt{2^{73}} = 2^{36.5} \approx 97$ billion, meaning that quantum computing would be 97 billion times more efficient than classical computers. Of course, the reality of quantum computing is not the most imminent at the moment, but it is at least arguable from this that blockchain itself will not be resistant to quantum attacks in the future. With DAG, however, n should not be a huge number, as each node must perform simple Proof of Work in order to partake in the transaction-making process. As a result, DAG's vulnerability to quantum attacks is much lower compared to traditional blockchain's.

That said, in order to add a new block in the DAG network, the node needs to reference several recently received but unconfirmed previous blocks as its parents by packing all the hashes of the previous blocks into its own block header. Like in blockchains, every new child block in the DAG confirms its parent blocks, hence the partial consensus will be established. Each previous block will be confirmed by more and more later blocks that reference its block hash indirectly.

ITC adopts such data structure of DAG to solve performance problems, exhibiting a great improvement in terms of transaction performance in the network with respect to speed, fees, and security.

Note that potential variations can include:

- Consensus algorithms, including PoW, PoS, DPoS, PBFT, etc.
- Parents selection algorithms.
- Crypto algorithms, such as lattice-based cryptography
- Additional network algorithms

2 Simplified Payment Verification

Downloading the entire data from a blockchain or a corresponding data structure is not the most efficient way. Especially in the scope of decentralization where users would be the ones who can serve as bookkeepers, there must be another way to store data. ITC uses SPV for that reason. As described in [2], SPV can help save disk space by a substantial portion.

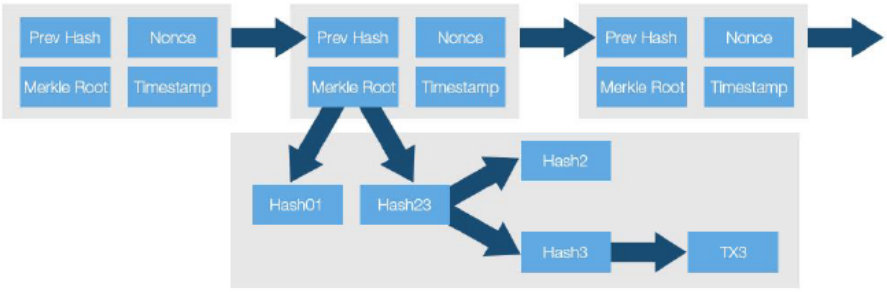

Figure 3: Simplified Payment Verification (SPV) principles

One must necessarily note that Merkle trees (or a variant) are employed in the process. By discarding the spent transactions in the structure as long as they are buried under enough blocks, the Merkle tree process significantly reduces the required storage space from the outset. As for the SPV process, each block header is what matters, as it becomes possible to verify transactions without fully downloading the whole chain by storing the relevant block headers only. This can be done by querying network nodes and obtaining the Merkle branch of a relevant transaction corresponding to the correct timestamp.

Proved accurate for several decades and used in the semiconductor industry to guide long-term planning of research and development, Moore's law states that the number of transistors in a dense integrated circuit has been observed to double roughly every two years. As a result, storage should be not an issue in years to come using SPV. Suppose a block header with no transactions would be about 80 bytes. Hence even in the case of Bitcoin whose blocks are generated every 10 minutes, the SPV process yields 80 bytes * 6 * 24 * 365 = 4.2 MB per year, which is definitely not a pressing amount of data.

Applied to IoT, a field that necessitates scalability, SPV would surely save the cost of blockchain payment verification and also reduce the burden on the users' end, aiding decentralization. Hence, ITC nodes would use such technology to solve the data expansion problem of major network and DAG. Improving payment verification efficiency is the key to ensuring the performance of the entire network.

3 Asymmetrical Encryption

Asymmetrical cryptography, or public key cryptography, offers a new paradigm when it comes to conjunction of security and decentralization of technology. As opposed to symmetrical, asymmetrical cryptography represents a system that uses a pair of keys: public (which may be disseminated to the public) and private (which is known only to its owner).

Hence, two functions can be accomplished. First is authentication, where one can use the public key to verify the fact that a sender of the message indeed has the paired private key. Second is encryption, where only the paired private key can decrypt the message encrypted with the public key. Note that the generation of a key pair should not be computationally hard in practice. Given this, what is required to guarantee effective security is then keeping the private key indeed private. Meanwhile, the public key can be distributed widely, as it will not compromise security.

As in the case of Bitcoin, asymmetrical encryption relies on mathematical problems that are currently difficult if not infeasible to solve. These include factoring $n \in \mathbb{N}$ into primes $p_1, p_2, ..., p_m$, discrete logarithm (i.e. solving $a^x \equiv b$ modulo p), and elliptic curve cryptography. Specifically, it is known that Bitcoin uses a specific elliptic curve called secp256k1, represented by an elliptic curve of the Weierstrass form

$$y^2 = x^3 + 7$$

over the finite field of $(2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1)$ number of elements, i.e. a Galoid field $GF(2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1)$.

Applying the Diffie-Hellman scheme to such elliptic curve cryptography, one can make use of ECDH (Elliptic Curve Diffie-Hellman) and ECDSA (Elliptic Curve Digital Signature Algorithm) by choosing an appropriate generator point on the elliptic curve, e.g. point

$$\begin{aligned} G &= (x, y) \\ &= (79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798, \\ &\quad 483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8) \end{aligned}$$

in hexadecimal. Such generator point combined with an essentially randomly generated private key (as well as hash functions such as SHA-256 and RIPEMD-160) then produces one's public key and eventually public address. Note that there does exist a collection of numerous other protocols and schemes utilizing elliptic curve cryptography, including those that use another famous curve called ecdsa25519, i.e. a Montgomery curve of the form

$$y^2 = x^3 + 486662x^2 + x$$

over the quadratic extension of the prime field defined by prime $p = 2^{255} - 19$.

Public key algorithms, unlike symmetric key algorithms, do not require a private or secure channel between the sender and the receiver. At the same time, they tend to be slower in computation than symmetric ones, as more intricate steps are necessary in the process to ensure security despite having potential attackers who can easily glean publicly available information during each transaction of data. Hence, those algorithms that are fast, secure, and contributing to decentralization should be the ones that will be adopted.

As a fundamental security ingredient in systems, applications, and protocols, asymmetrical encryptions underpin the Internet and various standards, such as TLS (Transport Layer Security), PGP, and GPG. In years to come, it is undoubtedly expected that such encryptions will be more commonly applied to fields such as information technology, security discipline, information security, and IoT.

Especially with respect to IoT, asymmetrical encryption will be powerfully used to assure confidentiality of data, authenticity, and robustness of communications, data storage, computation, and integration of physical machines into computer-based systems as a whole.

4 CPS

Cyber-physical system (CPS) refers to a mechanism in which computer-based algorithms integrate and intertwine software and physical components, allowing futuristic interactions between technologies, scales, and modalities pertaining to the Internet and its users in the system.

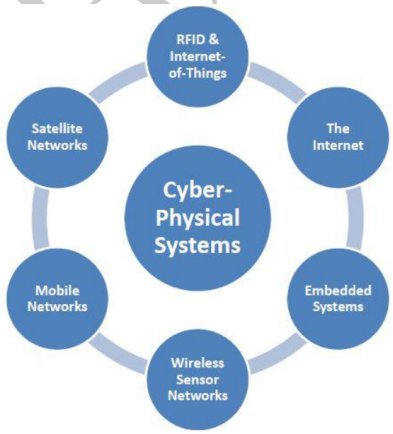

Figure 4: CPS [7]

As described in [7], it can be observed that recent developments in sensors, data acquisition systems, computer networks, and cloud computing have paved the road for designing CPS in various industry sectors, such as aerospace, automotive, civil infrastructure, energy, manufacturing, transportation, and consumer

appliances. What has been true is that the vast usage of sensors in industries has resulted in generating huge amount of data, which could be analyzed and utilized in systematic ways for the better. In the same vein is CPS, given the current big data environment in the modern day, and it is one of ITC's goals to achieve CPS in a meaningful way, incorporating autonomous control to industrial machines.

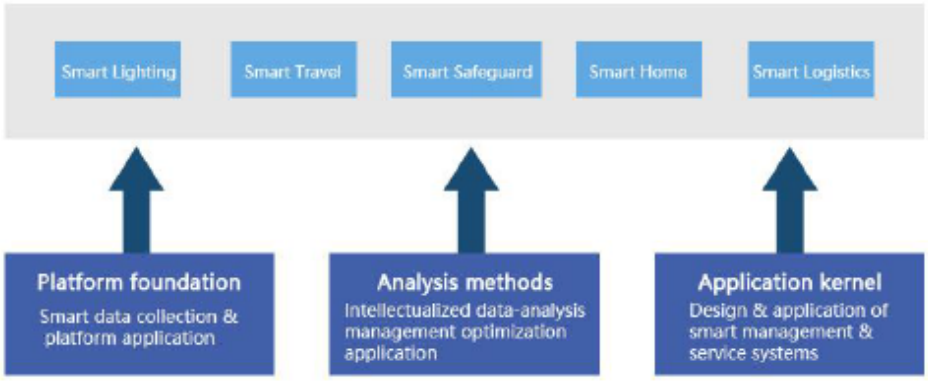

Figure 5: ITC's interactive network with respect to CPS

The architecture of ITC refers to CPS cluster and builds CPS technical system structure on networks of five levels, including connection, conversion, cyber, cognition, and configuration. On this system architecture, independent blocks of network communications, data analysis, and value transfer can improve the stability of IoT's ecosystem in ITC and make it more intelligent, contributing to the next phase of industrial evolution called Industry 4.0.

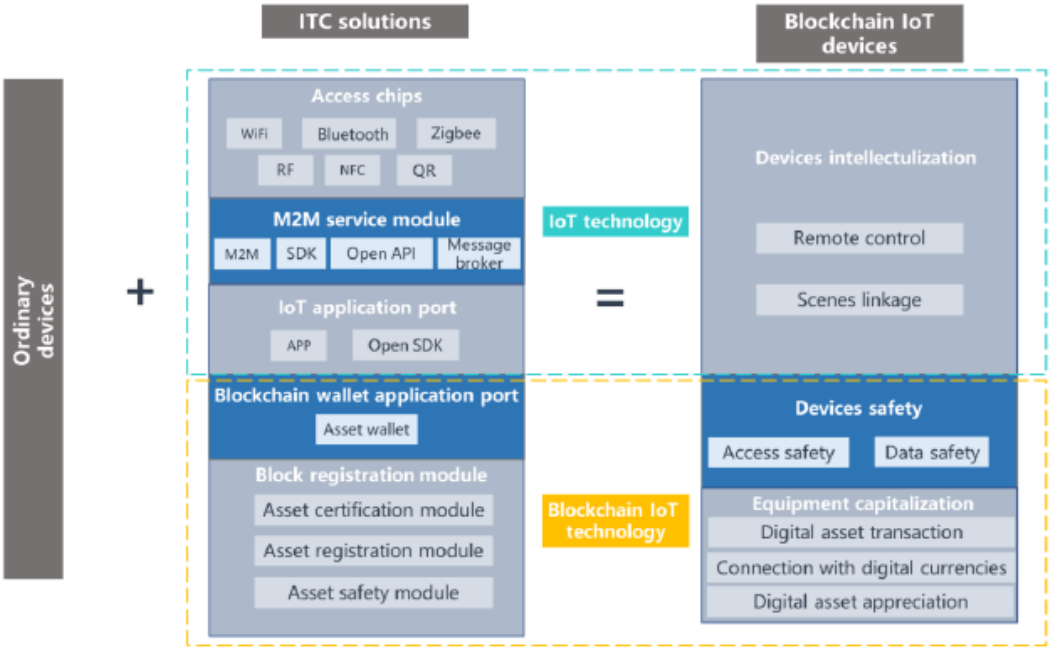

Figure 6: Architecture of ITC platform

5 System Architecture

In the system architecture of ITC, there are 3 defined types of devices, which are server, heavy node (known as Phone/Route), and light node (known as IoT chip). Under a certain set of circumstances, either light or heavy nodes can automatically be organized as a sub-network by themselves despite nodes' plugging in or out. These sub-networks are separated, and meanwhile servers are established to help them stay connected. The number of light nodes or heavy nodes can in fact be more than 1 in a sub-network, as only dependency is on the communication module's capability. All devices can connect with each other whereas the connection between light nodes and server is an exception.

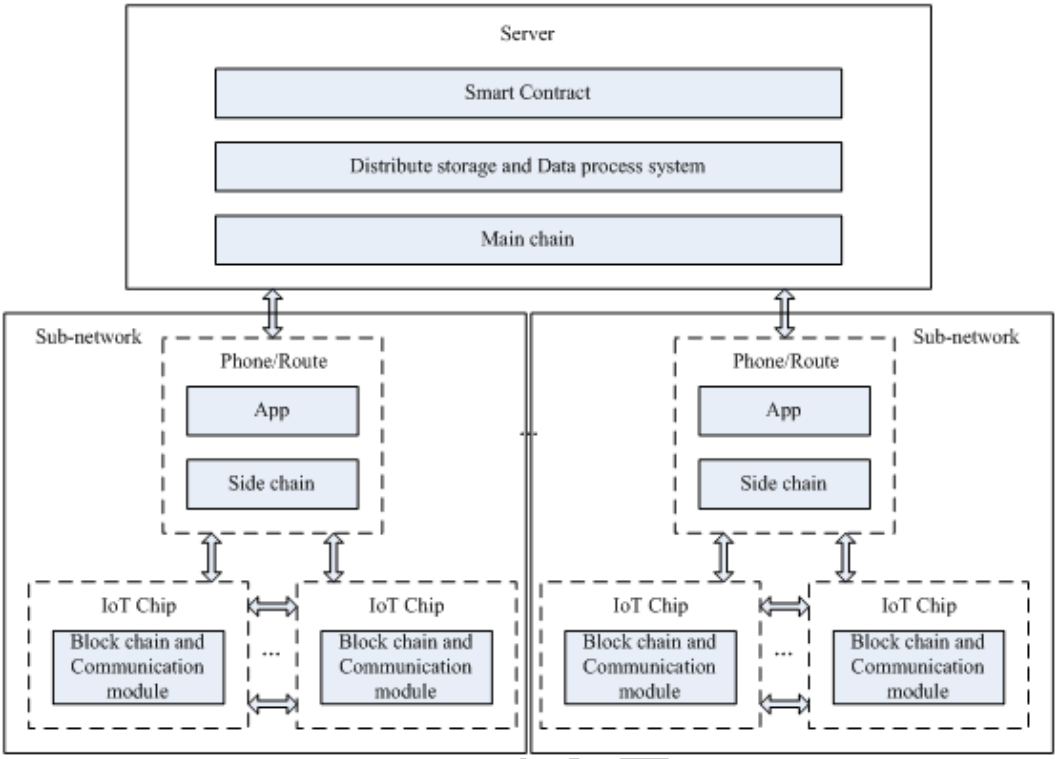

Figure 7: System architecture

There are some facts that distinguish the three devices. Normally, light node has a small storage memory and low computing speed, no side chain or consensus algorithm that can run on it, and no side chain ledger. Heavy node has a medium storage memory and medium computing speed, which is enough for side chain and the corresponding execution of consensus algorithm. It can also store the side chain ledger as well. Server has the largest storage memory and fastest computing speed so that it is natural to put the main chain on it. Distributed storage and data process script are also needed as main chain's components.

Given the system architecture, note that there are following functions in the ITC system as follows.

Function	Description
Sub-network connect	1. Nodes automatically connect by each other. 2. Initialize nodes and server, and establish each side chain and main chain. Technical detail: Embedded programming using communication module such as BLE.
Transaction transfer	1. Transaction transfer between nodes in the same sub-network. 2. Transaction transfer between nodes in different sub-networks. Technical detail: PBFT (among nodes and server), DAG (storage data structure), and SPV (in obtaining brief info when transactions occur between different sub-networks). *Transaction type: Valuable or Unvaluable.
Device function	Device's own functions besides the ITC blockchain.
Data process	1. Respond to nodes' requirement. 2. Data analysis for nodes' requirement. 3. Ledger storage and synchronization. Technical detail: PBFT, DAG, SPV, and Chaincode (for data analysis).
Smart contract	Extend functions. Technical detail: CPS (for various application scenes).

Figure 8: System functions corresponding to system architecture

6 Chaincode Data Analysis

ITC will become the most abundant data ecosystem in the generation of IoT, with ample data from smart IoT devices. At the same time, data should not yield to monopoly. That is why in ITC, users' data belong to users. Any company that plans to perform big-data analysis or algorithm model training (e.g. for the purpose of advertisement recommendation) needs to submit Chaincode to the ITC platform.

Meanwhile, probability model algorithms such as HyperLogLog and Bloom filters will be employed to provide the necessary interface API for such Chaincode data analysis.

1. HyperLogLog: HLL is an algorithm that counts or approximates the number of distinct elements in a multiset, tackling the count-distinct problem. The basic idea of the algorithm comes from the fact that the cardinality (i.e. number of distinct elements in a multiset) of a multiset of uniformly distributed random numbers can be estimated by calculating the maximum number of leading zeros in the binary representation of each number in the set. For instance, the fact the maximum number of leading zeros observed is n would

imply an estimate that the cardinality is in fact 2^n. Another key feature of HLL is the usage of hash functions, applied to each element in the original multiset to obtain a multiset of uniformly distributed random numbers with the same cardinality as the original multiset. This new multiset's cardinality is then estimated. Consider the following.

Let $h : \mathcal{D} \to [0,1] \equiv \{0,1\}^\infty$ *hash data from domain* $\mathcal{D}$ *to the binary domain.*
Let $\rho(s)$*, for* $s \in \{0,1\}^\infty$*, be the position of the leftmost 1-bit* ($\rho(0001\cdots) = 4$).

Algorithm HYPERLOGLOG (**input** $\mathcal{M}$: multiset of items from domain $\mathcal{D}$).
assume $m = 2^b$ with $b \in \mathbb{Z}_{>0}$;
initialize a collection of m registers, $M[1], \ldots, M[m]$, to $-\infty$;

for $v \in \mathcal{M}$ **do**
  set $x := h(v)$;
  set $j = 1 + \langle x_1 x_2 \cdots x_b \rangle_2$; {*the binary address determined by the first* b *bits of* x}
  set $w := x_{b+1} x_{b+2} \cdots$; **set** $M[j] := \max(M[j], \rho(w))$;

compute $Z := \left(\sum_{j=1}^{m} 2^{-M[j]} \right)^{-1}$; {*the "indicator" function*}
return $E := \alpha_m m^2 Z$ with α_m as given by Equation (3).

Figure 9: The HyperLogLog algorithm [13]

2. Bloom filter: A Bloom filter is a probabilistic data structure that tests whether or not an element is a member of a given set. One of its properties includes the fact that false positives by the filter are possible whereas false negatives are not. Consider the following simple Bloom filter.

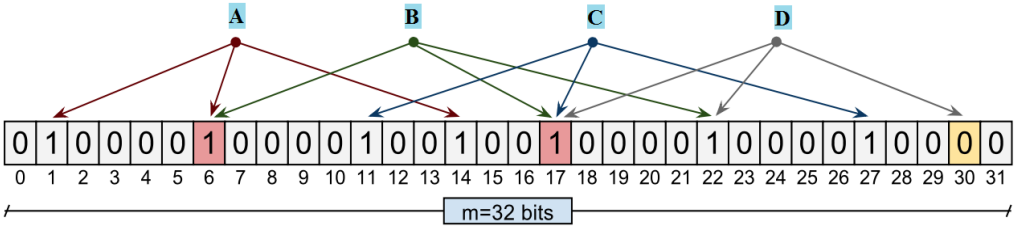

Figure 10: A simple Bloom filter

An empty Bloom filter is a bit array of m bits, initialized to 0. Also set from the outset are j number of distinct hash functions, each of which maps some element in the given set to one of the bits in the Bloom filter, generating a uniform random distribution. Note that it is typical to have $j < m$ whereas

the precise choice of j would depend on the intended false positive rate of the filter to begin with. Now, to add an element from the given set to the Bloom filter, one should feed it to each of the j hash functions to get j bit positions in the bit array and then set those bits to 1. The filter is set. To test whether or not an element is in the set, one should feed that element to the hash functions to get j array positions. If any of the bits at those positions is 0, the element is not in the set. It is in this way that false negatives are not possible. However, false positives can be possible, as those bits could have been set to 1 during the insertion of other elements by chance.

With the restriction and assignment of these interfaces combined with zero-knowledge proof, contracts in the ITC platform will not be able to steal users' initial data but can obtain aggregated data used for smart business decision. After execution of Chaincode, the companies would have to pay ITC tokens to users who provide the data. In this way, ITC provides a big-data analysis ecosystem where both users and companies can benefit in the platform in a way that incentives are aligned.

7 Practical Byzantine Fault Tolerance

There is no doubt that Byzantine fault tolerant algorithms will only become more important and relevant in years to come in the scope of technological advancement, as malicious attacks and network errors tend to be common and can cause faulty nodes to behave in undesirable ways. While algorithms whose assumptions include a synchronous system are not the most practical in the real world, PBFT (practical Byzantine fault tolerant) algorithms work in asynchronous environments like the Internet and are more flexible in the sense that there can be several major optimizations that can be done to improve the efficiency of such class of algorithms.

As in [1], it is known that PBFT offers both liveness and safety as long as at most $\lfloor \frac{n-1}{3} \rfloor$ number of nodes (out of n replicas) are faulty due to maliciousness or network error. Here, note that liveness assures the fact that clients will eventually receive a reply to every request sent, given that the network is functioning. On the other hand, safety assures the fact that the system will maintain state and look to the client like a non-replicated remote service. It means each replicated service would satisfy linearizability [3].

Previous to PBFT, most proposed algorithms resorted to assumptions on synchrony rather than asynchrony, making the algorithms less practical in a real network setting. Two notable algorithms were Rampart [4] and SecureRing [5] which were designed to be practical, but their assumptions were also on synchrony when it came to correctness. Such algorithms were more theoretical than pragmatic, as the possibility of malicious attacks in a network is not trivial especially in the modern day. For instance, an attacker may compromise the safety of a service by delaying

non-faulty nodes or the communication between them. The attack will work until the malicious nodes are tagged as malicious and hence faulty and excluded from the replica group. A denial-of-service attack is definitely easier than fully hacking a non-faulty node. By not relying on synchrony for safety, PBFT on the other hand proves to be impervious to such attack. Also, it is practical with regards to performance when compared to Rampart, SecureRing, and other algorithms, as it uses only one message round trip to execute read-only operations and two to execute read-write operations.

PBFT assumes an asynchronous distributed system. As the nodes are connected by a network, there is a nontrivial possibility where network errors (i.e. failures to deliver messages as intended) might occur. The model is as follows. Faulty nodes may behave arbitrarily but independently. In other words, one of the assumptions relies on independence of node failures. For this assumption to be true in the presence of malicious attacks, some steps need to be taken. For example, each node can run multiple versions of the service code as well as OS and should have a different root password as well as administrator. The same code base can indeed yield different implementations while for low degrees of replication, one can purchase OS from vendors for different results.

Another option is NVP (N-version programming), whose general approach can be described in the following way:

1. An initial specification is developed with regards to the intended software and its functionality. Such specification serves as the anchor point for further development. As a result, it should clearly define the basic structure, including but not limited to functions, data formats (e.g. c-vectors and cs-indicators), cross-check points, and comparison algorithm.

2. From the specifications, two or more versions of the program are independently developed. Due to this independence assumption, note that each group that works on the project should not interact with other groups. Hence, the implementations of these functionally equivalent programs might use different programming languages. At various points, special mechanisms are built into the software to allow the program to be governed by the N-version execution environment. These special mechanisms could be c-vectors (a data structure representing the program's state), cs-indicators, and synchronization mechanisms.

3. The N-version execution environment itself is developed, running the N-version software and making final decisions regarding the N-version programs as a whole given the output of each individual N-version program. Note that the decision algorithms can vary. They can be as simple as accepting the most frequently occurring output given the entire output set by the N-version programs.

All in all, cryptographic techniques are vital in preventing attacks, replays, and obliviousness to corrupted messages. Each message in PBFT contains asymmetrical signatures, message authentication codes, and message digests produced by mathematical hash functions. A message m signed by node i can be denoted by the pair (m, σ_i) whereas the hash (digest) of message m can be denoted by $h(m)$.

Note that a common practice is in fact to sign a digest $h(m)$ of message m and appending it to the plaintext of m rather than signing the full message itself. All replicas know the others' public keys to verify signatures. Assume a very strong adversary that can coordinate faulty nodes and cause intrusive delays in order to cause the most damage to the replicated service in the network. Nonetheless, assume that the adversary cannot cause delays indefinitely and that it is computationally bound so that it is infeasible for the adversary to subvert the modern day cryptography altogether. The adversary, for instance, should not be able to produce a valid signature of a node or create collisions of a hash function at will.

Given this, it is important to note that $3f+1$ is the minimum number of replicas needed to allow consensus in an asynchronous system and to provide liveness and safety, where f is the number of faulty replicas. A proof can be found in [6]. This makes sense, because consensus should be achieved after communicating with $n-f$ replicas, given f replicas that might be faulty and unresponsive. This yields $n-f > f$ so $n > 2f$. However, it is also possible that the f unresponsive replicas are not faulty while f of those that responded might be faulty. This yields $n-2f > f$ so $n > 3f$.

PBFT can be considered as a form of state machine replication, a term that refers to the general method for implementing a fault-tolerant network and service by replicating servers and orchestrating client interactions with server replicas. Here, the set of replicas can be denoted by R where $R = \{0, 1, ..., |R|-1\}$ with each integer representing a replica, indexed for simplicity. The replicas proceed through a series of configurations called views. In a view, one replica is called the leader while the others are called backups. Views are numbered consecutively. The leader of a view is replica l such that $l \equiv v \pmod{|R|}$ where v is the view number. Note that view changes occur when the leader has evidently failed.

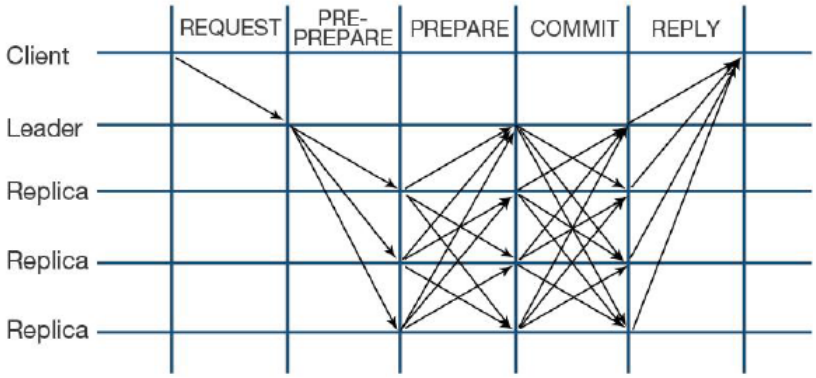

Figure 11: Achieving consensus in a PBFT algorithm

The following are the germane theoretical steps.

1. Request: The client sends a request to the leader. The leader validates the message and proposes a sequence number corresponding to it.
2. Pre-prepare: The leader sends a pre-prepare message to the backups, i.e. rest of the replicas, allowing them to validate the message and receive the sequence number.
3. Prepare: All functioning backups send a prepare message to every other, allowing replicas to acknowledge and agree on a total ordering.
4. Commit: All replicas multicast a commit. An ordering has been agreed upon.
5. Reply: Once $2f + 1$ commits have been received, a client places the request in the queue. Each functioning replica sends a reply directly to the client.

Note that in order to account for the case where the leader is shown to be faulty, the client can use a timeout. When this timeout expires, the request is automatically sent to all replicas. If a replica already knows about the request, the rebroadcast can be ignored. If a replica does not know about the request, another timer will be started. If this timer reaches a timeout, the replica can start the view change process.

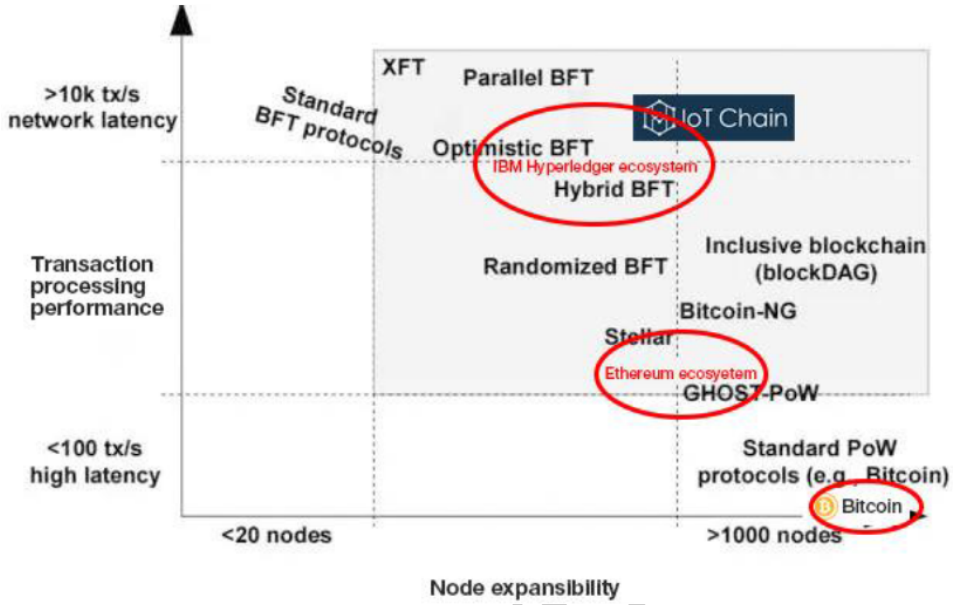

Figure 12: Comparison to other systems

Though using PBFT algorithm may cause some losses in the extensibility of nodes, both extensibility and performance needs can be balanced by adjusting the weight. As of yet, the blockchain technology based on PBFT consensus algorithm has been applied in digital currency of Central Bank of China, Bumeng Blockchain, and IBM's Hyperledger. Recently, the HoneyBadgerBFT consensus protocol has been put forward as another viable asynchronous BFT protocol.

By adopting the PBFT consensus protocol, ITC has greatly improved the main chain's processing performance and at the same time sought to maintain and promote the idea of decentralization. Compared to Bitcoin's Proof-of-Work algorithm and also Ethereum's GHOST-PoW, ITC's algorithm boasts an exceptional balance between node expansibility and performance with respect to transaction rate.

References

[1] Castro, Miguel, and Barbara Liskov. *Practical Byzantine fault tolerance*. OSDI. Vol. 99. 1999.

[2] Nakamoto, Satoshi. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf

[3] M. Herlihy and J. Wing. *Axioms for Concurrent Objects*. ACM Symposium on Principles of Programming Languages, 1987.

[4] M. Reiter. *The Rampart Toolkit for Building High-Integrity Services*. Theory and Practice in Distributed Systems (LNCS 938), 1995.

[5] K. Kihlstrom, L. Moser, and P. Melliar-Smith. *The SecureRing Protocols for Securing Group Communication*. Hawaii International Conference on System Sciences, 1998.

[6] G. Bracha and S. Toueg. *Asynchronous Consensus and Broadcast Protocols*. Journal of the ACM, 32(4), 1995.

[7] Bagheri, Behrad, et al. *Cyber-physical systems architecture for self-aware machines in industry 4.0 environment*. IFAC-PapersOnLine 48.3 (2015): 1622-1627.

[8] Gilks, Walter R., Sylvia Richardson, and David Spiegelhalter, eds. *Markov chain Monte Carlo in practice*. CRC press, 1995.

[9] Geyer, Charles J. *Practical markov chain monte carlo*. Statistical science (1992): 473-483.

[10] Lewenberg, Yoad, Yonatan Sompolinsky, and Aviv Zohar. *Inclusive block chain protocols*. International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2015: 528-547.

[11] Cachin, Christian, and Marko Vukolić. *Blockchains Consensus Protocols in the Wild*. arXiv preprint arXiv:1707.01873 (2017).

[12] Cachin, Christian. *Architecture of the Hyperledger blockchain fabric*. Workshop on Distributed Cryptocurrencies and Consensus Ledgers. 2016.

[13] Flajolet, Philippe, et al. *Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm*. AofA: Analysis of Algorithms. Discrete Mathematics and Theoretical Computer Science, 2007.

[14] Flajolet, Philippe, and G. Nigel Martin. *Probabilistic counting algorithms for data base applications*. Journal of computer and system sciences 31.2 (1985): 182-209.