UDC 004.056.5

Aspects of blockchain reliability considering its consensus algorithms

K.S. Gorniak[1,2], A.M. Kudin[1,2]

[1] *National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», Institute of Physics and Technology*

[2] *National Bank of Ukraine*

Abstract

The reliability of blockchain as an information system is discussed in this article. There were considered two types of models for blockchain systems. One assumes an almost ideal decentralized network with the probability of a software crush on an independent nodes, other adds to this approach a probability of the communication channels corruption. We study consensus algorithms used in blockchain and make assumptions about their reliability and functioning in practice.

Keywords: Blockchain, Consensus protocols, Blockchain reliability

Introduction

Building new information systems can lead to an actual problem of creating decentralized protocols of processing information, such that the participants of the protocol don't need a centralized service to work valid. One of the technologies that implement that kind of protocols is blockchain.

Blockchain is based on the distributed between participants of the peer-to-peer network ledger and specifications, so called blockchain consensus protocols, by which the ledger can be modified.

The ledger is used to record transactions across many computers so that any involved record cannot be altered retroactively, without the alteration of all subsequent blocks. This allows the participants to verify and audit transactions independently and relatively inexpensively.

Consensus protocols are at the core of distributed computing and also provide a foundational building protocol for multi-party cryptographic protocols. Such protocols must respect two important resiliency properties, consistency and liveness. Consistency ensures that all honest nodes have the same view of the log, whereas liveness requires that transactions will be incorporated into the log quickly.

All in all blockchain technology is one of the most promising modern hi-tech technologies for various processing of information. The efficiency of its functionalities is based on the resilience of cryptographic systems in cryptanalysis attacks.

This paper studies some of the consensus protocols of blockchain for their reliability.

1. Technical overview

Blockchain consensus protocols

Consensus algorithms allow a set of machines to work as a group that can survive the failures of some of its participants. Because of this, they play a key role in building reliable large-scale software systems.

In distributed systems, there is no perfect consensus protocol. The consensus protocol needs to make a trade-off among consistency, availability and partition fault tolerance (CAP)[1]. Besides, the consensus protocol also needs to address Byzantine Generals Problem that there will be some malicious nodes deliberately undermining the consensus process. Next in the section there are brief descriptions of three blockchain consensus protocols that can effectively solve Byzantine Generals Problem [2] and were taken for research in this paper.

PoW (Proof of Work)[3]: PoW selects one node to create a new block in each round of consensus by computational power competition. In the competition, the participating nodes need to solve a cryptographic problem. The node who first addresses the puzzle can have a right to create a new block. It is very difficult to solve a PoW puzzle. Nodes need to keep adjusting the value of nonce to get the correct answer, which requires much computational power. It is feasible for a malicious attacker to overthrow one block in a chain, but as the valid blocks in the chain increase, the workload is also accumulated, therefore overthrowing a long chain requires a huge amount of computational power. PoW belongs to the probabilistic-finality consensus protocols since it guarantees eventual consistency.

PBFT (Practical Byzantine Fault Tolerance)[4]: PBFT is a Byzantine Fault Tolerance protocol with low algorithm complexity and high practicality in distributed systems. PBFT contains five phases: request, pre-prepare, prepare, commit and reply. The primary node forwards the message sent by the client to the other three nodes. In the case that node 3 is crashed, one message goes through five phases to reach a consensus among these nodes. Finally, these nodes reply to the client to complete a round of consensus. PBFT

guarantees nodes maintain a common state and take a consistent action in each round of consensus. PBFT achieves the goal of strong consistency, thus it is an absolute-finality consensus protocol.

Raft[5]: Raft is a consensus algorithm for managing a replicated log. In order to enhance understandability, Raft separates the key elements of consensus, such as leader election, log replication, and safety, and it enforces a stronger degree of coherency to reduce the number of states that must be considered. Raft also includes a new mechanism for changing the cluster membership, which uses overlapping majorities to guarantee safety.

PoW is a probabilistic-finality protocol, and attackers need to accumulate a large amount of computational power or coins (stake) to create a long private chain to replace a valid chain. For instance, in Bitcoin, a fraction of the computational power is sufficient for an attacker to create a longer private chain to successfully complete a double-spend attack. Hence, if attacker's fraction of the computational power is more than or equal to, the blockchain network will be undermined. In PBFT, if there are a total of nodes in the network, the number of normal nodes must exceed, which means that the number of malicious or crashed nodes must be less than. Therefore, the fault tolerance of PBFT is 1/3. The fault tolerance of Raft is the same as PoW, 1/2, since Raft is in fact crash-tolerance protocol and doesn't actually deal with maliciously intended nodes.

Reliability of software

Reliability of the software can be defined as a resistance or an ability to function smoothly and to restore the working state after failures have occurred. Stability of the system depends on the ability to respond to undetected errors so that it does not affect reliability[6].

All existing indicators of software reliability can be divided into two large groups:

1) quantitative indicators of software reliability;
2) qualitative indicators of software reliability.

We only consider quantitative characteristics of software reliability, which are:

- Infallibility. It characterizes ability of the software to perform its functions during exploitation. We assume that the failure of the program is the result of the manifestation of a hidden error. The input and data generated by the program are not software elements and their reliability is related to the operation of external devices and hardware. Only the constants entered by the programmer are considered as a part of the software.
- Resistance. Software stability determines the ability of a machine to perform a given function under the conditions of interference (errors, failures, crushes) occurring in non-program sources (technical support, initial data).
- Correctness. It characterizes the software's adaptability to finding and correcting errors and making changes to it during operation. It is used to characterize programs that are being restored during operation. Correction indicators are: T_k correction time, program correction probability for a given time $P_k(t)$, coefficient of readiness K_r, others.
- Protection. It is an indicator of protection against third-party software intrusions.

2. Reliability model

Speaking of validity and reliance - blockchain, as a distributed data base, highly depends on the boundaries that are set for its number of honest and corrupted players of the protocol. Blockchain functionates correctly due to the consensus algorithm which makes sure that all nodes in the network will come to a common solution. In other words it means the algorithm must solve the Byzantine generals problem, make decisions despite dealing with m corrupted participants.

"Honesty" is usually understood as an unmistakable work of a protocol participant that can be defined by the reliability of a software. Assuming that hardware part is working correctly, there was considered mathematical model about appearing mistakes in the program complex during its testing.

There were made next assumptions for building such model:

- time intervals between mistakes are statistically independent;
- in the testing process there can be accumulated big groups of mistakes, that create a queue for a program correction;
- intensity of finding mistakes remains constant, until they are not reformed.

We than consider different approaches to the blockchain model as a distributed system. First model is taking a conception that the only mistakes, that are appearing in the system, are faults and crushes in the members' software. Such approach helps build a mathematical model based on queuing management system. Another approach fills up the first one by making an assumption about mistakes in the transmission channels. Both of the methods consider that there are no delay when sending a message to another party.

2.1. Modeling blockchain as a queuing network

As a new reliability prediction model, we are considering a queuing network with blocking. In such system, a arrival that finds all channels busy receives a response, leaves the system, and no longer participates in the service. In the understanding of the model of detection of distortions in the PC we consider the actual errors, which during the operation of the system manifest themselves in some nodes (channels). That is, the trap that came to the channel is interpreted as detecting an error on the node.

The adequacy of such a model is based on assumptions about the indicative distribution of service time (allowed within the model) and the Poissonian nature of the application flow, which means the fulfillment of two properties: the process in the system must be

stationary and future changes should not be affected by changes in the past.

It is obvious that in the beginning, immediately after the system is put into operation, the process will not be stationary, and in the queuing system (as in any dynamic system) a so-called «transient», non-stationary process will arise. However, after a while, this transient process dies down, and the system switches to a steady-state, so-called «set» mode, the probabilistic characteristics of which will no longer depend on time.

In this paper we consider exactly the characteristics of the limit service mode. To be able to evaluate the reliability of the model being built, we simplify it (a complex model cannot be described) by neglecting the dependency of errors on the PC with each other. This assumption is possible because modern compilers detect and correct such errors more often and faster than those directly related to the logic of the program.

Suppose we have a n channel queued queuing system, that is, in terms of decentralized protocol, of n network members. Consider X as a physical system with a finite number of states:

x_0 – all channels are free (program error),

x_1 – Exactly one channel (single node error),

. . . . . .

x_k – equals k of channels (faulty nodes on the network),

. . . . . .

x_n – all n channels are busy, that is, distortions are detected on all nodes

The problem now transforms into finding state probabilities p_k $(k = 0,\ 1,\ \ldots,\ n)$ for any moment of time t. It can be solved if we take next assumptions as truth:

1) Flow of applications with density λ;
2) service time (exempting the channel from the application, i.e eliminating the error) T_{last} is exponentially distributed with a parameter $\mu = \frac{1}{m_t}$.

In other words, this assumptions leads our model to be a Markov process so the appropriate mathematical field can be used.

$$g(t) = \mu e^{-\mu t},\ t > 0$$

The parameter μ is similar to the parameter λ of the exponential distribution law T between adjacent simple flow events:

$$f(t) = \lambda e^{-\lambda t},\ t > 0$$

λ makes sense as a flow of applications (errors), and similarly μ can be considered as «release flow density». In fact, a queue that is constantly busy (program errors constantly appear); then obviously there will be a simpler stream with density μ in this channel.

We determine the probabilities $p_0(t), p_1(t), \ldots, p_n(t)$ to accept the possible states of $x_0,\ x_1, \ldots, x_n$. For any point in time is true

$$\sum_{k=0}^{n} p_k(t) = 1$$

We fix the time t and find the probability that at $t + \Delta t$ the system will be in state x_0 (all queues are free; all nodes are working correctly), this can happen in two ways:

A – at the moment t, the system was in state x_0, and during Δt did not change from x_1 (no application came in),

B – at t, the system was in the state x_1, and during Δt the channel was released (i.e. the error was corrected) and the system went into the state x_0.

Using the theorem of summing probabilities:

$$p_0(t + \Delta t) = P(A) + P(B)$$

We can find the probability of the event A using the multiplication theorem. The probability that at t the system was in the state x_0 is $p_0(t)$. The probability that no request will occur during δt is $e^{-\lambda \Delta t}$. Accurate to the magnitude of the larger order of smallness

$$e^{-\lambda \Delta t} = 1 - \lambda \Delta t$$

That means that $P(A) = p_0(t)(1 - \lambda \Delta t)$.

Using similar thoughts to the event B:

$$e^{-\mu \Delta t} = 1 - \mu \Delta t$$

That way we get $P(B) = p_1(t)(1 - \mu \Delta t)$

$$p_0(t + \Delta t) = p_0(t)(1 - \lambda \Delta t) + p_1(t)(1 - \mu \Delta t)$$

Than we transform the last equation into the differential one:

$$\frac{dp_0(t)}{dt} = -\lambda p_0(t) + \mu p_1(t)$$

Similar differential equations are constructed for other probabilities of states. Take any k $(0 < k < n)$ and find the probability $p_k(t + \Delta t)$ that at $t + \Delta t$) the system will have a state x_k.

Such a probability is calculated as the probability of the sum of two events rather than two:

A – at t, the system was in state x_k and did not switch from one to another during Δt

B – at t the system was in state x_{k-1} and switched to x_k during Δt (application came)

C – at t the system was in state x_{k+1} and switched to x_k during Δt (one queue became free)

By making computations on above mentioned equations next probabilities are acquired:

$$P(A) = p_k(t)(1 - \mu k + \lambda)\Delta t)$$

$$P(B) = p_{k-1}(t)(\lambda \Delta t)$$

$$P(C) = p_{k+1}(t)(\mu \Delta t)$$

We than can calculate all probabilities $p_n(t)$ and get a common system:

$$\begin{cases} \frac{dp_0(t)}{dt} = -\lambda p_0(t) + \mu p_1(t) \\ \ldots\ldots\ldots\ldots \\ \frac{dp_k(t)}{dt} = p_{k-1}(t)\lambda - (\lambda + k\mu)p_k + \mu(k+1)p_{k+1}(t) \\ \ldots\ldots\ldots\ldots \\ \frac{dp_n(t)}{dt} = \lambda p_{n-1}(t) - n\mu)p_n \end{cases}$$

To find the boundary probabilities (system states probabilities in the set mode), all the probabilities at

their boundaries are replaced by $p_0, p_1, \ldots, p_n$, and all derivatives become zero. The resulting system is no longer a system of differential but algebraic equations.

$$\begin{cases} -\lambda p_0 + \mu p_1 = 0 \\ \lambda p_0 - (\lambda + \mu) p_1 + 2\mu p_2 = 0 \\ \ldots\ldots\ldots\ldots \\ \lambda p_{k-1} - (\lambda + k\mu) p_k + (k+1)\mu p_{k+1} = 0, 0 < k < n \\ \ldots\ldots\ldots\ldots \\ \lambda p_{n-2} - (\lambda + (n-1)\mu) p_{n-1} + n\mu p_n = 0 \\ \lambda p_{n-1} - n\mu) p_n = 0 \end{cases}$$

Solving the second system by its unknown variables from the first equation $p_0, p_1, \ldots, p_n$, we obtain:

$$p_1 = \frac{\lambda}{\mu} p_0 \tag{1}$$

$$p_2 = \frac{1}{2\mu}(-\lambda p_0 + (\lambda + \mu) p_1) = \frac{\lambda^2}{2\mu^2} p_0 \tag{2}$$

The value of $\alpha = \frac{\lambda}{\mu}$ is the estimated density of requests flow, that is, the average number of applications that arrive at the average time of service of one application. In the model of error considered, this value indicates the average number of errors that occur over the average time required to correct a single distortion.

Then $\alpha = \lambda m_t$, where m_t is the average service time of one application. In new notation, the formula for p_k takes the form

$$p_k = \frac{\alpha^k}{k!} p_0 \tag{3}$$

Expressing all probabilities of p_k through α:

$$p_0 = \frac{1}{\sum\limits_{k=0}^{n} \frac{\alpha^k}{k!}}$$

$$p_k = \frac{\frac{\alpha^k}{k!}}{\sum\limits_{i=0}^{n} \frac{\alpha^i}{i!}}$$

Thus, by obtaining probabilities $p_0(t), p_1(t), \ldots, p_n(t)$, we can find the average number of corrupted nodes among all participants of the decentralized protocol, that is, they had a detected error that led to a malfunction.

Let ξ be a random variable that takes $0, 1, 2, \ldots, n$. Each possible value of k means that x_k with probability p_k (working with k nodes) fails on the system. Then the mathematical expectation of a random variable ξ is the average number of unfair nodes.

$$M\xi = \sum_{k=0}^{n} k p_k$$

For each of the three blockchain consensus algorithms we evaluate probable meaning of the variables: m_t – the average error correction time and λ – the flux density of occurring errors: Hyperledger Fabric (IBM) – uses PBFT consensus, permissioned blockchain, closed software, $m_t = 2 - 3$ months, $\lambda = 7 - 8$ errors per month;

Bitcoin is a public blockchain with open (free) software, uses PoW, $m_t = 6$ months, $\lambda = 10 - 15$ errors per month;

Quorum (JP Morgan) is a private open source blockchain built on the Ethereum platform, uses Raft algorithm, $m_t = 4$ months, $\lambda = 7 - 8$ errors per month.

Knowing the restrictions on the number of dishonest nodes, in which the algorithm works correctly, and the average number of nodes with failures, it is possible to find the minimum number of nodes for the normal operation of the protocol for the three protocols, which were discussed (pic. 1, 2, 3).

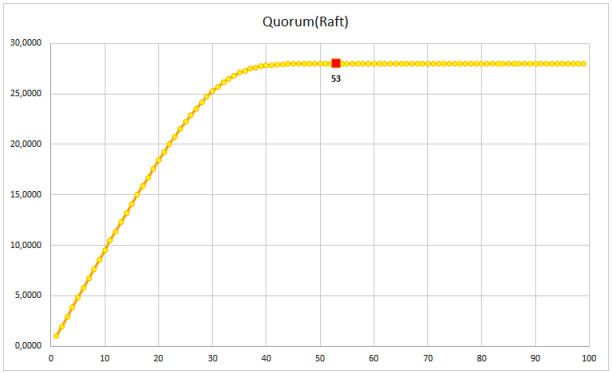

Fig. 1. Quorum. The average number of dishonest nodes, depending on the n number of all nodes

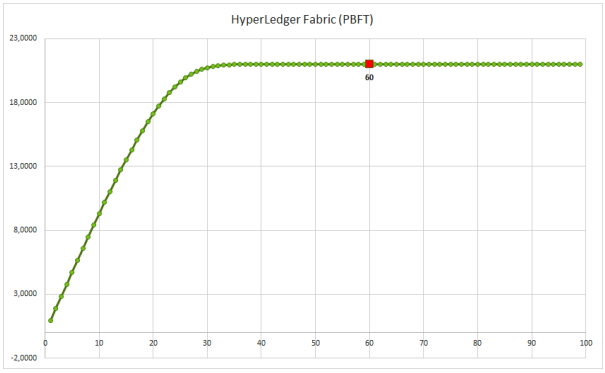

Fig. 2. Hyperledger Fabric. The average number of dishonest nodes, depending on the n number of all nodes

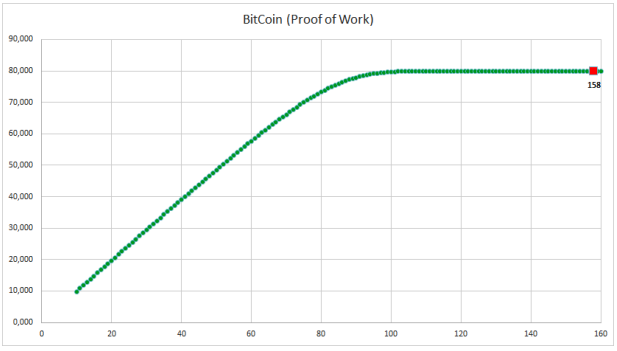

Fig. 3. Bitcoin. The average number of dishonest nodes, depending on the n number of all nodes

The graphs show that, although the limit on the number of dishonest nodes in the Proof of Work algorithm ($< 1/2n$) is weaker than for the PBFT algorithm ($< 1/3n$), the Hyperledger Fabric system requires fewer nodes to build consensus, that is, it comes to the correct working of the protocol rather than the Bitcoin network. This observation occurs because Hyperledger Fabric is a Linux Foundation project run by highly qualified professional teams and is client-specific, so bug fixes are faster than open source systems like Bitcoin.

Although Quorum has open source and similar bitcoin restrictions on unfair nodes (in the sense of failures) ($< 1/2n$), it is fairly quickly negotiated between the parties by using Ethereum, which has a high level of security, and a closed blockchain that guarantees control by a centralized organization, and therefore we have a fairly quick fix to the distortions in the software.

2.2. Modeling bitcoin with communication channels' faults

In the situation above, it is assumed unrealistically that all messages are transmitted without error. Although noisy channels with zero error capacity exist, there is more practical interest in the opposite.

Transmission errors are unavoidable, especially given the presence in any communication channel of noise, which is the sum total of random signals that interfere with the communication signal. In order to take the inevitable transmission errors into account, some adjustment in encoding schemes is necessary. We consider a simple model of transmission in the presence of noise, the binary symmetric channel. Binary indicates that this channel transmits only two distinct characters, generally interpreted as 0 and 1, while symmetric means that errors are equally probable regardless of which character is transmitted. The probability that a character is transmitted without error is labeled p, hence, the probability of error is 1 - p.

We also assume that information data is transporting without delays in time, so it can simplify describing the model. Excluding time delays rises the probability that the first winner of solving the puzzle will be the first to transmit their block successfully to the others.

Therefore, time will be considered as a main value for reliability of blockchain. We say that blockchain is not reliable when time for agreeing on a new block is higher than some known limit T, so it contradicts the ability of a system to perform given functions under the conditions of interference (errors, failures, crushes).

The situation when time limit is surpassed just by network not being able to generate a new block for some $t > T_{limit}$ is too unrealistic to take into account. This comes from the fact that bitcoin network is constantly changing the difficulty of the solving problem so the block can be found in an appropriate time interval (10 minutes).

For the practical interest is taken a situation where a block B can't be agreed upon by the majority of the nodes, i.e. more than the half of participants of the protocol don't have block B in their digital ledgers and every six validation blocks that goes after can't be agreed upon by majority too.

We assume that a block B generated by a particular miner will not be included into another node's blockchain if: 1) that node is malicious; 2) that node has crushed due to the software failure; 3) miner who solved the puzzle was not able to transmit found block to that node due to the corrupted communication channel. The summary number of such nodes must be higher than the half number of all protocol participants for all six blocks in a row, so that the first one will not be agreed upon for the time surpassing an established hour.

Conclusions

We discussed the reliability of the operation of blockchain systems in this paper. Reliability should be introduced as an opportunity to preserve the proper functioning of blockchain under the influence of random, natural phenomena. In particular, the solution of the problem of blockchain reliability considering errors in software, which appeared during development and were not detected during testing, was considered. The unreliability characteristics for Byzantine, Raft and POW blockchains consensus protocols were obtained.

References

[1] P. Kernfeld, "How bitcoin loses to the cap theorem," *https://paulkernfeld.com/2016/01/15/bitcoin-cap-theorem.htm.*

[2] M. P. Leslie Lamport and R. Shostak, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 3, pp. 382–401, 7 1982.

[3] "Consensuspedia: An encyclopedia of 30+ consensus algorithms. a complete list/-comparison of all consensus algorithms," *https://hackernoon.com/consensuspedia-an-encyclopedia-of-29-consensus-algorithms-e9c4b4b7d08f.*

[4] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, (Berkeley, CA, USA), pp. 173–186, USENIX Association, 1999.

[5] H. Howard, "ARC: Analysis of Raft Consensus," Tech. Rep. UCAM-CL-TR-857, University of Cambridge, Computer Laboratory, July 2014.

[6] V. Lipayev, *Reliability and functional safety of real-time software systems.* Moscow: in russian, 2013. 176 p.