Blockchain and Consensus Algorithms in Internet of Things

Conference Paper · November 2018

DOI: 10.1109/ISETC.2018.8583923

4 authors:

Sorin Zoican
Polytechnic University of Bucharest
45 PUBLICATIONS 58 CITATIONS
SEE PROFILE

Marius Vochin
Polytechnic University of Bucharest
45 PUBLICATIONS 72 CITATIONS
SEE PROFILE

Roxana Zoican
Polytechnic University of Bucharest
40 PUBLICATIONS 51 CITATIONS
SEE PROFILE

Dan Galatchi
Polytechnic University of Bucharest
27 PUBLICATIONS 39 CITATIONS
SEE PROFILE

Blockchain and Consensus Algorithms in Internet of Things

Sorin Zoican, Marius Vochin, Roxana Zoican, Dan Galatchi
Politehnica University of Bucharest
Romania
sorin@elcom.pub.ro

***Abstract* — this paper focuses on the performance evaluation of consensus algorithms used in a blockchain system for Internet of Things (IoT). In such systems the time necessary to achieve consensus should be small. Three most used consensus algorithms (modified proof of work, practical byzantine fault tolerance and binary consensus) are evaluated in different situations – type of motes, number of nodes participating in consensus algorithm and radio propagation model. An integrated solution is proposed to adapt an IoT node to different consensus algorithm. The simulations in Contiki IoT operating system show good performance (time to achieve consensus less than seconds)**

Keywords— consensus algorithm, Internet of Things, blockchain

I. Introduction

The Internet of Things is nowadays in a continuous expansion [1]. Smart cities, intelligent houses and other applications require communication between a lot of nodes which usually consists of low or medium computational power and small memory amount microcontrollers. Generally, the IoT systems use a server-client communication model in which the devices are identified, authenticated and connected through servers with high computation and storage resources. The devices have a unique identity and they are connected through Internet. Nowadays this communication paradigm will not longer be used due the huge growing of the number of IoT devices. The centralized solutions became expensive – they use many servers and networking equipment to ensure the communication between billions of devices and they need maintenance. Decentralized methods represent solutions that answer to the high costs issue in huge IoT systems. The computation power and storage requirements will be shared among the devices in the network. However, the peer to peer communication arises some important issues to solve: privacy, security, data validation and consensus.

These nodes must maintain a coherent data base with all information regard to the operations of cooperative nodes. An efficient solution to do this is blockchain technology [2]. It represents a distributed database with a continuous growing of the public data records (or blocks) added to it. The distributed database can be only extended, and previous inserted blocks can not be removed. The whole chain can not be replicated without very high costs. Each participant maintains a copy of the blockchain and a new block will be inserted in the chain after consensus between participants is achieved. A centralized database (running over Internet) uses client-server network architecture. A client can change information on a centralized server (of course, if it has the permission to do this). Database control is maintained by a central authority. The main issue is the network security: if the central authority node is corrupt then all the nodes in the network will be at risk. In a blockchain database, each node updates the information into the database working together to ensure the consensus providing built-in security for the network. In a blockchain system the nodes are connected (using different topologies – peer to peer, star) in a distributed system and each node contributes to decide using a consensus algorithm. Two main issues arise here: (a) the complexity of the consensus algorithm should be chosen taking into consideration the limited resources of nodes and ensure that the consensus algorithm remains effective and (b) the network load should be kept as low as possible due the power consumption limitations. This paper is focused on comparison of practical methods to ensure the consensus in a blockchain system. The following consensus algorithms will be analyzed: the proof of work, the byzantine fault tolerance and the binary consensus.

Next sections will present consensus algorithms, their implementations in Contiki operating system, performance evaluation using Cooja simulator and conclusion.

II. Blockchain Technology in Internet of Things

The blockchain technology may represent a better solution for large number of nodes. In an IoT network, the blockchain keep an immutable record of the history of devices and enables its functioning without the need of a centralized node. Using blockchain in IoT will reduce the expenses and unpredictability of working devices [3]. A block contains the "transactions" created within a given interval of time. The data in each block is secured using a hash function and contains a reference to the block before it, using the previous block's hash (except for the first block) as it is shown in the Figure 1.

The earlier blocks cannot be modified, because changing a block changes its hash and this breaks references in a blockchain.

There are two types of nodes in a blockchain system: nodes that want to insert new blocks in the blockchain and nodes that receive these new blocks and implement a consensus mechanism to include a new block in the blockchain.

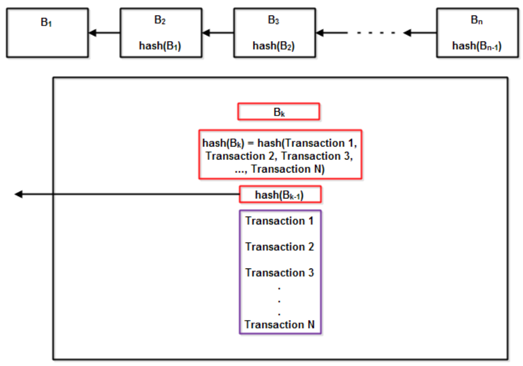


Figure 1. The blockchain

III. Consensus Algorithms

Three consensus algorithms will be analyzed: proof of work (PoW) and a new variant of PoW, the classical Byzantine Fault Tolerance (BFT) algorithm and a binary consensus (BC) algorithm. For all these consensus algorithms the following features will be evaluated: computation time (including all processing steps and the transmission time), number of messages exchanged in the network to implement the algorithm and the average time interval in which the consensus is reached considering two radio propagation models which increase the number of messages in the network due the needs of retransmissions.

Proof of Work

The proof of work consensus algorithm [3,4] is illustrated in Figure 2.

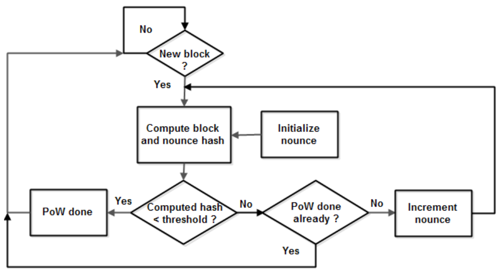


Figure 2. The PoW algorithm

When a node wants to generate a new block, the receiving nodes (called miners) will compute a hash function of the block data and a nounce. The computed hash value must be less than a predefined value (this value gives the complexity of the work). If this condition is not true, the nounce is incremented and the hash function is recomputed with the new nounce. When a miner has completed its work then the block has a proof of work and it will be verified by other nodes in the network. The hash function has the property that the verification can be performed fast, comparing with PoW calculations. Resolving the cryptographic problem with a target starting with k zeroes has the complexity $O(16^k)$ but checking a proposed block has only the complexity $O(1)$. Reducing the complexity of proof of work is needed in IoT systems to obtain a smaller time to achieve consensus. If most of these nodes validate the PoW, the block is inserted in the blockchain copies of each node within the system. The majority is given by $\alpha_1.N$ nodes where $0.66 < \alpha_1 < 1$ and N is the number of nodes participating to achieve consensus. The whole system is since the computation power necessary to insert a fake block is more than one third of total computational power in the system. The hash computing complexity may be adjusted by a threshold. In case that more than one miner completes the PoW at the same time a fork in blockchain appears. Futures validated block will be inserted in the longest chain and the remaining chain will be lost.

The PoW algorithm may be simplified as follows [4]:

- A miner will be forced to wait a small number of blocks after its last proposed block, before starting a new PoW job (except the first block)
- Each miner computes a hash function of the result of concatenation of its unique address identifier and its public key (denotes as *MIN* – Miner Identification Number)
- Compute a block score as $score = 2^{256} - |H - MIN|$ or $score = 2^{256} / |H - MIN|$ where H may be the hash value of the preceding block or the hash value of the concatenation of the hash values of the previous K blocks
- The first miner that complete the PoW will send the block for validation
- If more than one miner completes the PoW at the same time, the block proposed for validation will be the block with the highest score

For IoT systems the simplified PoW is more suitable due its smaller computational time. Using a simplified PoW will decrease the time to achieve the consensus between nodes in an IoT network and this is the most used business scenario.

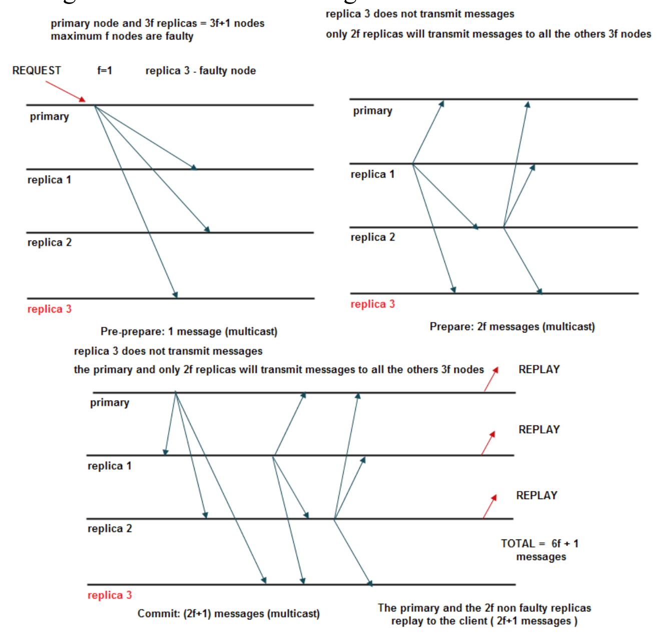
Practical Byzantine Fault Tolerance

The Byzantine Fault Tolerance algorithm [5] works in synchronous systems and requires several voting rounds, therefore may be impractical sometimes. There are practical implementations such Practical Byzantine Fault Tolerance (PBFT) where nodes share sequence nodes (called view or group). The group must have a leader (or a primary node) which is elected using an election algorithm. The rest of nodes are called replicas. If the primary node fails a new leader will be elected and a new group is created. The nodes do not change their states (decisions) and the requests are totally ordered, therefore the algorithm is safe. The client will receive a replay to every request, so the algorithm is liveness. The safety and liveness are guaranteed if less than $f = (N-1)/3$ replicas have failed, where N is the number of nodes in a group (view). The algorithm work as follows:

1. Client node sends a request message to the primary node. The primary node validates the message and provides an identification number to the request message.

2. The primary node sends “pre-prepare” messages to all replicas which validate the client request message and receive the identification number.
3. The replicas send “prepare” messages to all other replicas and agree on a total ordering of these messages.
4. The replicas multicast a “commit” message. They agreed on a total ordering and acknowledged the receipt of the client request.
5. Each non-faulty replica (including primary) send a “replay” message to the client.
6. The client waits for *2f* valid messages and makes the decision.

The algorithm is shown in the Figure 3.



primary node and 3f replicas = 3f+1 nodes
maximum f nodes are faulty
REQUEST
f=1
replica 3 - faulty node
primary
replica 1
replica 2
replica 3
Pre-prepare: 1 message (multicast)
replica 3 does not transmit messages
only 2f replicas will transmit messages to all the others 3f nodes
primary
replica 1
replica 2
replica 3
Prepare: 2f messages (multicast)
replica 3 does not transmit messages
the primary and only 2f replicas will transmit messages to all the others 3f nodes
REPLAY
primary
REPLAY
replica 1
REPLAY
replica 2
TOTAL = 6f + 1 messages
replica 3
Commit: (2f+1) messages (multicast)
The primary and the 2f non faulty replicas replay to the client (2f+1 messages)

Figure 3. PBFT Consensus algorithm

Binary consensus

In binary consensus [6] each node has one of four possible states. The nodes will learn the state of majority of nodes (that is, most nodes have observed an event). The binary consensus algorithms guarantee the correct conclusion – after an interval of time the consensus will be reached - and the time to convergence is upper bound. The possible node states are defined in the Table 1. In a fully connected topology the convergence time is bounded by $\log(N)/(2\alpha-1)$ where N represents the number of nodes participating to achieve consensus and $\alpha > 0.5$ is the fraction of nodes initially in the majority state [6].

Table 1

State	State description (what the node believes)
0	The opinion of most of its neighbors is that the event is *most likely false*
1	The opinion of most of its neighbors is that the event is *might be false*
2	The opinion of most of its neighbors is that the event is *might be true*
3	The opinion of most of its neighbors is that the event is *most likely true*

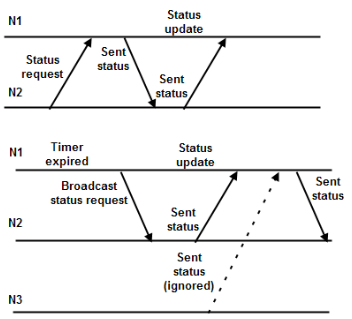


Figure 4. Binary consensus algorithm

The nodes update their state as is indicated in Table 2 (assume that node **N1** communicate with node **N2**). The nodes communicate as in Figure 4. The convergence is achieved when all nodes are in states $\{0,1\}$ or in $\{2,3\}$.

Table 2

Node N1 initial state	Node N2 state transmitted to node N1	Node N1 updated state	Node N2 updated state
0	1	1	0
0	2	1	0
0	3	2	1
1	0	0	1
1	2	2	1
1	3	3	2
2	0	0	1
2	1	1	2
2	3	3	2
3	0	1	2
3	1	2	3
3	2	2	3
x	x	x	x

IV. Implementation and Performance Evaluation

The implementation of these algorithms was done using the Contiki operating systems for Internet of Things. Each node run a control process which integrate the above presented types of consensus algorithms as is shown in Figure 5. It is a good assumption that the number of nodes participating in the consensus must not be greater than 25.

The computational effort was evaluated for the three above presented consensus algorithms for two radio media (UDGM - Unit Disk Graph Medium and MRM - Multi-path Ray tracer Medium [7]). To implement the graph of the control process, a multicast transmitter thread (*MC_TX_Process*) and a multicast receiving thread (*MC_RX_Process*) must be created. These threads are realized as protothreads in Contiki IoT operating system and use Real-time Transport Protocol (RTP) protocol. The protothreads are lightweight threads that operate stackless using minimum memory and provide fastest context switching [8]. The execution time for these threads is about *0.2* μs per byte. The proposed solution adapts automatically each node in the IoT network to each considered consensus algorithm. A control variable, *Consensus_Type,* was defined and it is set by the network accordingly with the type of consensus algorithm as *PoW* – blockchain consensus, *BFT* – practical Byzantine fault tolerance consensus, and *BC* – binary consensus.

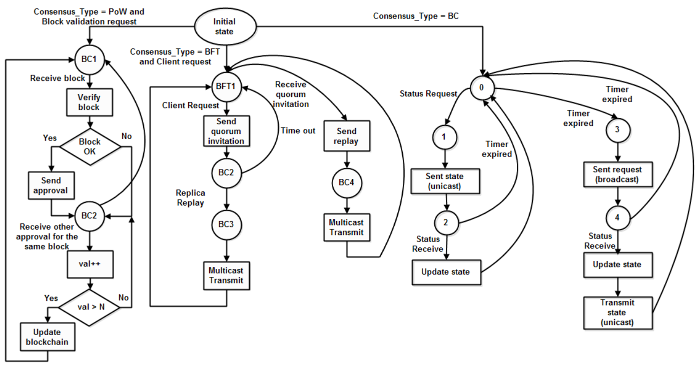


Figure 5. The graph of the control process

The graph from Figure 5 was implemented in C language and was simulated using Cooja simulator for Contiki operating system [7]. The computation time was estimated for various situations such as – mote used (Tmote Sky [9] and Z1 mote [10] at maximum 16MHz frequency), the number of nodes, *N*, that participate to set a quorum to achieve consensus, the consensus algorithm used and the radio channel propagation. The average execution time (which consists of computation time and transmission time for all the messages necessary to implement the algorithm) is in the order of milliseconds per transmitted packet of *320* bytes as it can be observed in Table 3 and Figure 6. All the three consensus algorithms have good computing times. Also, the time to achieve agreement is small (less than second) for a reasonable number of nodes participating to consensus algorithm as one can observe in Figure 7. The time to achieve consensus is about milliseconds for UDGM radio propagation model and about hundreds of milliseconds for MRM radio propagation model. From Figure 7 we can note that the average time to agreement is greater for BC algorithm in both radio propagation medium and the modified PoW and PBFT has an opposite behavior in UDGM and MRM medium.

Table 3

	PoW	**PBFT**	**BC**
Computing	*30 μs*/byte	*0.2 μs*/byte	*0.2 μs*/byte
Messages	$\alpha_1 N$	$(N-1)/2+3$	$\log(N)/(2\alpha-1)$

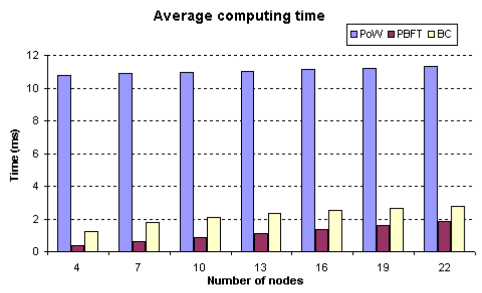


Figure 6. Average computing time

The explanation is that the PBTF is more sensitive to the number of messages necessary to achieve agreement than PoW and the number of retransmissions is greater in MRM than in DRGM. This is observed also from the Figure 6 – the average computing time is almost constant for PoW.

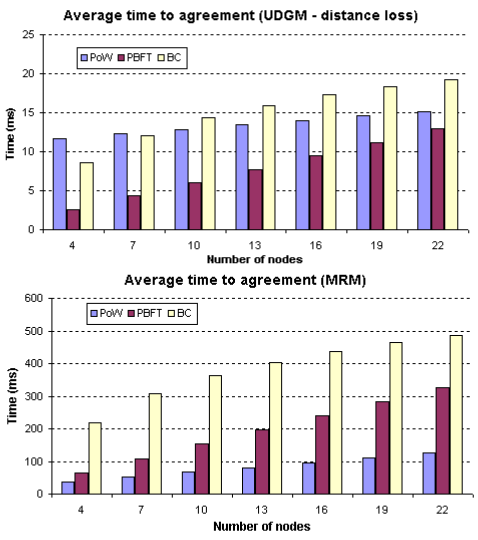


Figure 7. Average time to agreement

V. Conclusion

The paper evaluates the computational effort to implement three consensus algorithms used in blockchain systems for IoT in which the time for achieving consensus should be reasonable small. An integrated solution is proposed that adapts automatically a node to the consensus algorithm used in the network. The simulation results show that the proposed implementation has good performances (the computational time is less than seconds) and it is robust to various conditions (such as type of mote, number of nodes used in consensus algorithm and radio propagation). The performance is due both to the optimized primitives of Contiki and to the definition and organization of the proposed processes.

VI. References

[1] Wortmann, Felix & Flüchter, Kristina. (2015). Internet of Things. Business & Information Systems Engineering. 57. 221-224. 10.1007/s12599-015-0383-3.

[2] Zibin Zheng ; Shaoan Xie ; Hongning Dai ; Xiangping Chen ; Huaimin Wang. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends, Big Data (BigData Congress), 2017, Honolulu, HI, USA, DOI: 10.1109/BigDataCongress.2017.85

[3] Zhang, Y. & Wen, The IoT electric business model: Using blockchain technology for the internet of things J. Peer-to-Peer Netw. Appl. (2017) 10: 983. https://doi.org/10.1007/s12083-016-0456-1

[4] http://www.chainfrog.com/wp-content/uploads/2017/08/consensus.pdf

[5] Castro, M.; Liskov, B. (2002). "Practical Byzantine Fault Tolerance and Proactive Recovery". ACM Transactions on Computer Systems. Association for Computing Machinery. 20 (4): 398–461, doi:10.1145/571637.571640

[6] Abdaoui, Abderrazak & El-Fouly, Tarek. Distributed binary consensus algorithm in wireless sensor networks with faulty nodes. 2013 7th IEEE GCC Conference and Exhibition, GCC 2013. 495-500. 10.1109/IEEEGCC.2013.6705829

[7] Thomson, Craig & Romdhani, Imed & Al-Dubai, Ahmed & Qasem, Mamoun & Ghaleb, Baraq & Wadhaj, Isam. (2016). Cooja Simulator Manual. 10.13140/RG.2.1.4274.8408.

[8] http://dunkels.com/adam/dunkels05using.pdf

[9] http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf

[10] http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.