# COMPARISON OF CONSENSUS ALGORITHMS FOR IoT-BASED BLOCKCHAINS

CS-5320: Distributed Computing

Saim Khan
CS20MTECH14008
Sagar Agarwal
CS20MTECH11005
Department of Comp Sci & Engg, IITH

# 1 Problem Statement

We intend to compare the consensus algorithms that are/can be used in the IoT-based blockchains. Among the various options, we choose to compare the two most suitable candidates for IoT-based blockchains, namely Raft [1] and PoAu [2]. Our choices are mainly driven by the constraints that are usually associated with IoT networks like low computational power, low storage capacity, limited power, and low latency requirement.
We will compare their performance in terms of throughput, latency, number of messages exchanged and scalability.

# 2 Recent Related Works

There are numerous consensus algorithms that can be used in a conventional blockchain. But since these were not developed keeping in mind the resource constraint nature of IoT devices, very few of them are suitable for IoT-based blockchains. The low computational power of the IoT devices rules out the use of algorithms like PoW (Proof of Work), PoA (Proof of Activity), etc[3]. Similarly, low storage capacity rules out algorithms like PoC (Proof of Capacity) and so on[3]. Therefore a number of consensus algorithms have been proposed in recent years that were developed keeping in mind the constraints and requirements of IoT-based blockchains. For example, PoAu, Raft, PoAh, PoBT, and PoP. Among these, we are not going to consider PoAh because does not talk about the ordering of the block in the blockchain[4]. PoBT is promising but its usability is limited to a resource-rich industrial IoT environment[5]. PoP requires specialized hardware to operate[6].
**Raft:** It is light-weight, fast & scalable consensus algorithm. Due to its low computational requirements and low associated network overhead, it finds wide usage in the industry like in Hyperledger Fabric project [7]
**PoAu:** It is also a recently proposed concept, so it doesn't have wide implementations. Most of the implementations are based on the Ethereum network, as PoAu was developed by the Ethereum team itself. PoAu is currently being used by two clients for the permissioned setting of Ethereum namely Geth and Parity.

# 3 Algorithm Details

## 3.1 Raft

Raft is a voting-based non-byzantine fault tolerant consensus algorithm. It can only tolerate crash faults up to 50% of the nodes. It is fast, scalable algorithm, with low computational requirement and small network overhead.

The consensus mechanism involves the following steps:

1. **Leader Election**: Leader is responsible for ordering the transactions. Leader Election stage is executed using a randomized timeout for each server when existing leader fails.

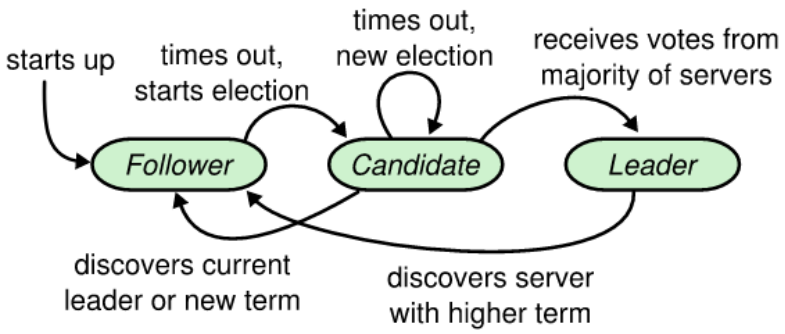


Figure 3.1: State transitions in the Raft

All the nodes start in the *follower* state. Since there is no leader, timeout is triggered in the nodes. This timeout happens after some random duration of time. Afterwards, the timed out node transitions into *candidate* state, increments the *term* and asks other nodes for the votes by sending *VoteRequest* message. If the receiving node itself is not a candidate and its *term* is less than the *term* of the received message, it votes for the *candidate* by sending back the *Vote* message. The *candidate* wins if he receives the majority votes, else re-election is triggered and timeout timer starts again.

2. **Log Replication**: Leader accepts log entries from clients and broadcasts transactions to make its version of the transaction log.
   After leader is elected, it prepares a new block. The new block is then broadcasted to all other nodes. After reception of the new block, the receiver sends back the *acknowledgement* message. After receiving the *acknowledgement* message from all the nodes, the block making node sends the *commit* message to all the nodes. The block is subsequently checked and then if check is successful block is committed.
   Throughput and performance depend on the leader node

## 3.2 PoAu

This algorithm belongs to the family of Byzantine-Fault-Tolerant consensus algorithms and it gives better performance as compared to PBFT because of fewer message exchanges. PoAu can be applied to IoT-based on permissioned blockchains, where some amount of data integrity can be compromised whereas PBFT is good for the scenarios which require strong data consistency. Permissioned blockchains rely on message-based consensus schema rather than on hashing procedures. We are going to implement one of the PoAu implementations named Aura(Authority Round). To handle the faulty nodes scenario, PBFT relies on three rounds of message exchange before reaching some conclusion, whereas Aura requires only two rounds of message exchange. PBFT ensures that if there are 3f + 1 nodes and out of them if 'f' are faulty, the nodes can still reach a conclusion, which is around 33.33% whereas Aura claims to tolerate 50% of the faulty nodes.
PoAu has N trusted nodes(called authorities), each identified using a unique id and out of them, at least N/2 +1 are considered to be non-faulty. In order to order the transac-

tions issued by the clients, authorities run a consensus algorithm. This algorithm uses a mining rotations schema to distribute the workload of creating blocks. Here, time is divided into steps and in each step one of the authority is elected as a mining leader. Aura basically involves two rounds: block proposal round and block acceptance round.

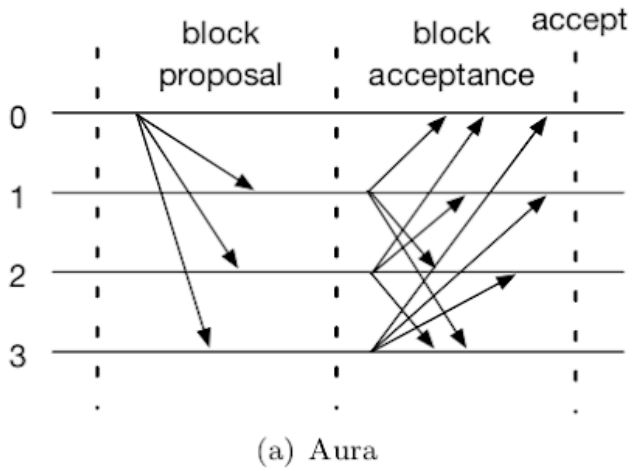


Figure 3.2: Messages sent in Aura

*Assumptions*

1. The network is considered to be synchronous. All authorities/nodes are also considered to be synchronized within the same UNIX time 't'.
2. Every authority maintains two queues, 'Qtxn' where each issued transaction is stored and 'Qb' where pending blocks are stored.
3. Variable 'step_duration' is the duration of each step that will be decided before starting the algorithm.
4. Let's call the index of each step 's', and each authority computes 's' by the formula s = t/step_duration.
5. The leader 'l' for each step is one of the authorities identified by the formula: l = s mod N.

The consensus algorithm mechanism for each 'step' involves the following steps:

1. Every authority will collect the issued transaction and store it in 'Qtxn'.
2. The leader puts all the transactions present in 'Qtxn' into a block 'b' and broadcasts to all the other authorities(called as block proposal round) as you can see in the above figure. If there are no transactions in 'Qtxn' , then an empty block will be sent.
3. Now each authority will send the received block to other authorities (called as block acceptance round).
4. If all the authorities received the same block b, all the authorities will put the block in their locally defined queue 'Qb'.

5. If the authorities disagree on the proposed block, voting is conducted to find that the leader is malicious or not. If the leader is decided to be malicious, then the leader will be turned down by its position if the majority of the votes are in favor of removing him.
6. The maliciousness of the leader is decided on these three factors:
   (a) Leader hasn't proposed any block to the authorities.
   (b) Leader has proposed more blocks than expected.
   (c) Leader proposed different blocks to different nodes/authorities.
7. When a leader is removed from its position, all the blocks in the 'Qb' are discarded.
8. Block b will remain in 'Qb' until N/2 + 1 authorities propose their block.
9. Then the block b is added to the block chain.

This algorithm works fine until the number of byzantine nodes crosses N/2 - 1.

# 4 FUNCTIONAL VIEW

The setup consists of a beacon and n nodes. The nodes are continuously receiving readings (along with the time of reading) from a connected IoT-sensor which is keeping track of the electricity consumed by some machine. The beacon and the nodes are fully connected. After receiving the reading from the connected IoT-sensor, individual nodes broadcast this reading to other nodes.

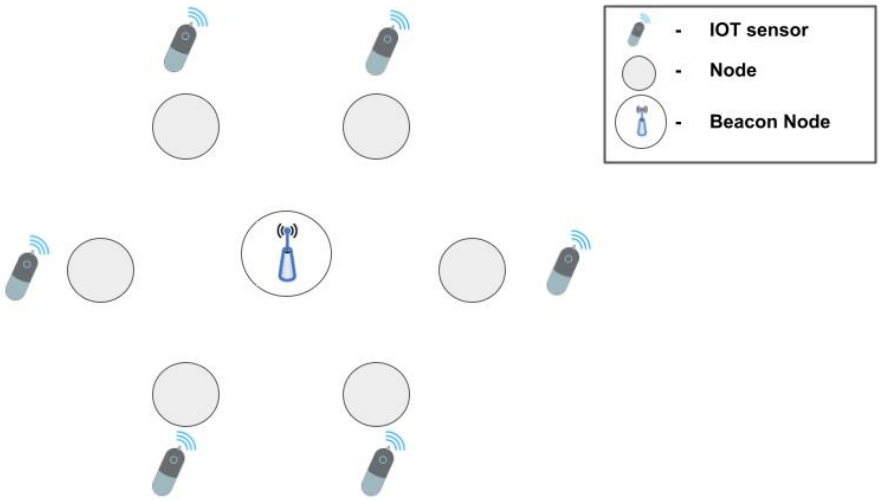


Figure 4.1: Functional View

# 5 Implementation Details

Nodes are either threads of same process running on the same processor or they are full processes running on geographically separated processors.
Initially, the nodes are not aware of the IP addresses of other nodes. The nodes only know the IP address of the beacon (given through command line).
The execution has two phases. Phase 1 is common for both the programs (Raft implementation and PoAu implementation).

## Phase I

The nodes send their IP address and the public-key to the beacon. The beacon collects the IP addresses and public-key of all the nodes, compiles them and sends them to all the nodes.

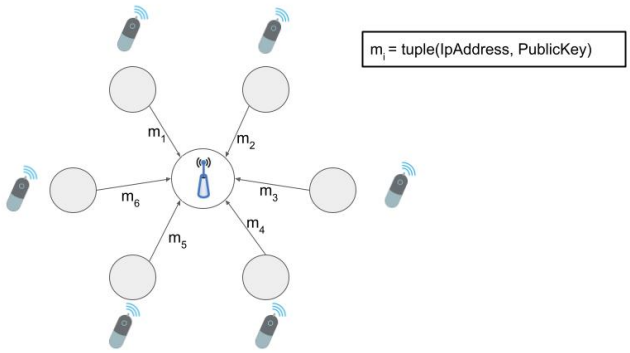


Figure 5.1: Nodes send their IP addresses and public-keys to the beacon

Afterwards, each node establishes a tcp connection with every other node and starts sending the transactions (readings), received from the attached IoT-sensor to every other node.

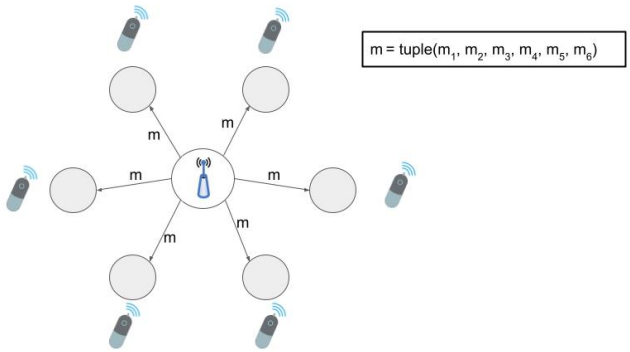


Figure 5.2: Beacon sends back the compiled IP addresses and public-keys to the node

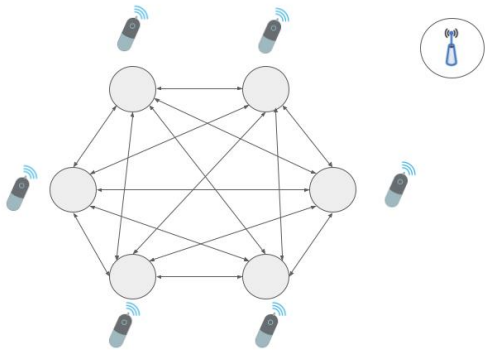

Figure 5.3: Nodes fully connected now

## Phase II

***Raft:***
All the nodes start in the *follower* state. Since there is no leader, timeout is triggered in the nodes. This timeout happens after some random duration of time. Afterwards, the timed out node transitions into *candidate* state, increments the *term* and asks other nodes for the votes by sending *VoteRequest* message. If the receiving node itself is not a candidate and its *term* is less than the *term* of the received message, it votes for the *candidate* by sending back the *Vote* message.
The *candidate* wins if he receives the majority votes, else re-election is triggered and timeout timer starts again.
After leader is chosen, it picks the oldest transactions from the pool of unconfirmed transaction, prepares the block as shown below. The term is set to the its current term and block number & message type is set appropriately.

| Previous Block Pointer (8 bytes) |
|---|
| Previous Block Hash (33 bytes) |
| Transactions' Hash (33 bytes) |
| Block Number (4 bytes) |
| Message Type (2 bytes) |
| Term (2 bytes) |
| Transactions (10 x 86 bytes) |
| Next Block Pointer (8 bytes) |

Figure 5.4: Block Structure of Raft implementation

The new block is then broadcasted to all other nodes. After reception of the new block,

the receiver sends back the *acknowledgement* message. After receiving the *acknowledgement* message from all the nodes, the block-making node sends the *commit* message to all the nodes. Afterward, *Previous Block Hash* of the block is matched to the hash of the last block in the blockchain. If it does not match, it is put in the *orphaned block pool*. Else, the *orphaned block pool* is searched for which could be its successors. Subsequently, the transactions in the block are checked against their signature using public-key. If the transactions match, the block is committed. Else it is discarded.

***PoAu:***
The leader is chosen in round-robin fashion and changes after every round. The leader picks the oldest transactions from the pool of unconfirmed transaction, prepares the block as shown below.

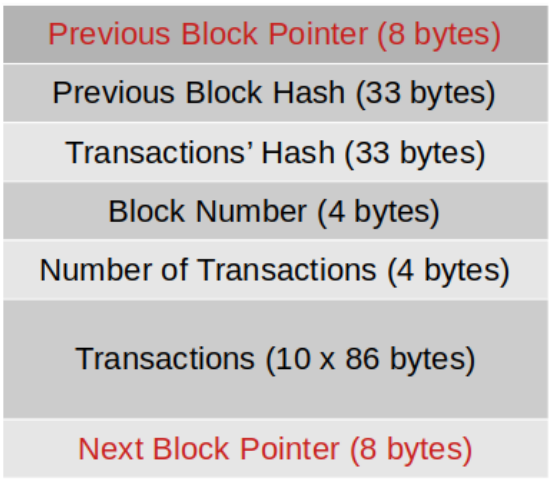


Figure 5.5: Block Structure of PoAu implementation

The leader then broadcasts the new block to all other nodes. After reception of the new block, the receiver checks its signature and if the check is successful, forwards this block to every other node. After receiving the blocks from all the nodes, their signature are checked and they are compared to each other. If majority of the block are not identical, then no block is committed in that round. Else afterward, the *Previous Block Hash* of the block is matched to the hash of the last block in the blockchain. If it does not match, it is put in the *orphaned block pool*. Else the *orphaned block pool* is searched for which could be its successors. Subsequently, the transactions in the block are checked against their signature using public-key. If the transactions match, the block is committed. Else it is discarded.

# 6 Results

As evident from the figure 6.1, we are getting higher throughput with the Raft compared to the PoAu (Aura). This can be attributed to the fact that to commit a block in the PoAu, we do extra operation of comparing the block received from the leader to the blocks forwarded by the other non-leader nodes, which are absent in the Raft. Another

reason is that in the PoAu, the non-leader node has to forward and receive the block (received from the leader) to all other non-leader nodes, whereas in case of the Raft, non-leader node has to only send the acknowledgement messages to the leader. This disparity in number of messages exchanged can also be implied from the figure 6.2.

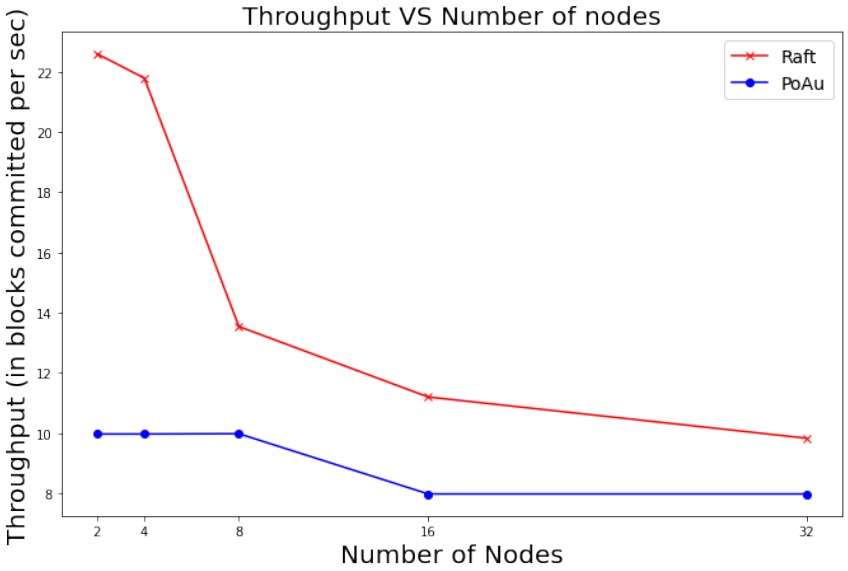


Figure 6.1: Throughput VS Number of nodes

The dip in the throughput of the Raft can be attributed to the fact that as the number of nodes increase, the number of nodes from whom votes are requested and number of votes required to be received increase. Hence the delay in the leader election results in decreased throughput.

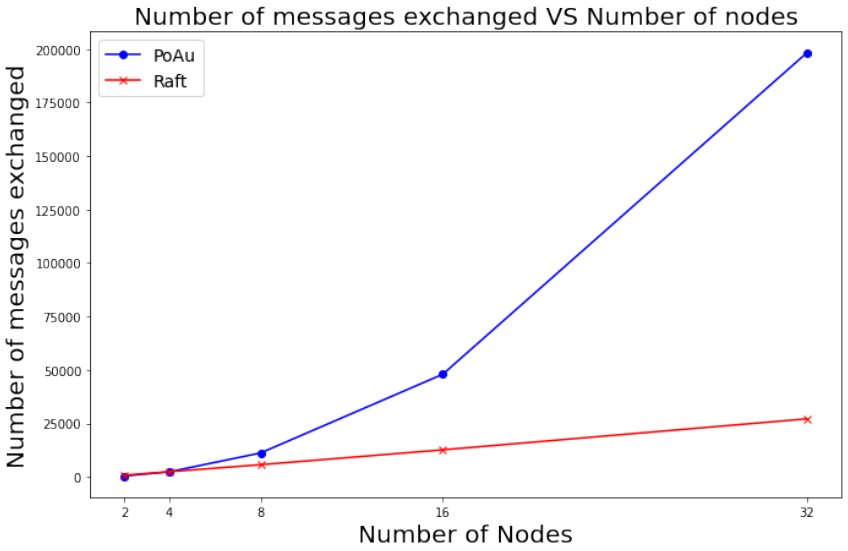


Figure 6.2: Number of messages exchanged VS Number of nodes

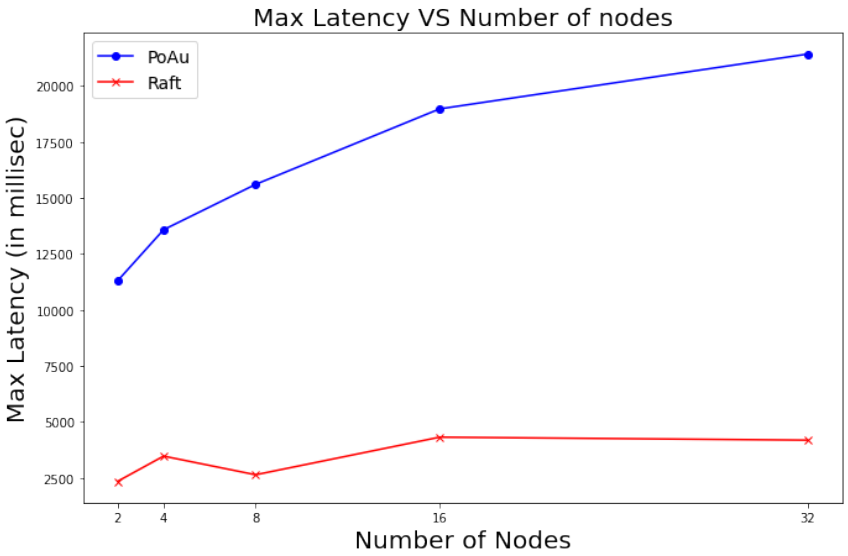


Figure 6.3: Max Latency VS Number of nodes

In the case of Raft, message complexity for leader and a non-leader node are O(n) and O(1) respectively. Whereas in the case of PoAU, its O(n) and O(n). The overall message complexity for Raft and PoAu stands at O(n) and O($n^2$). This is in-line with what we observe in figure 6.2.
Raft also performs better in terms of latency, as can be observed from figure 6.3.

# 7 Conculsion

From our quantitative analysis in the preceding section, Raft appears to be a clear winner in terms of throughput, network overhead and latency. But there are other factors that must be taken into account before choosing a consensus algorithm for a particular application like security. The PoAu is more secure against the malicious nodes (33%) than the Raft which is just crash-tolerant (50%).
Therefore, if the concerned IoT-blockchain is permissioned or private then only Raft should be preferred, else PoAu must be preferred over Raft. However since most of the IoT networks are private, it is expected that Raft would find more acceptance in the arena of IoT-based blockchains.

# References

[1] Ongaro, Diego, and John Ousterhout. "In search of an understandable consensus algorithm." In 2014 USENIX Annual Technical Conference (USENIXATC 14), pp. 305-319. 2014. i

[2] De Angelis, Stefano, Aniello, Leonardo, Baldoni, Roberto, Lombardi, Federico, Margheri, Andrea and Sassone, Vladimiro (2018) PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain. Italian Conference on Cyber Security, Milan, Italy. 11 pp. i

[3] Mehrdad Salimitari, Mainak Chatterjee, A Survey on Consensus Protocols in Blockchain for IoT Networks arXiv:1809.05613 [cs.NI] i

[4] D. Puthal and S. P. Mohanty, "Proof of Authentication: IoT-Friendly Blockchains," in IEEE Potentials, vol. 38, no. 1, pp. 26-29, Jan.-Feb. 2019, doi: 10.1109/MPOT.2018.2850541. i

[5] S. Biswas, K. Sharif, F. Li, S. Maharjan, S. P. Mohanty and Y. Wang, "PoBT: A Lightweight Consensus Algorithm for Scalable IoT Business Blockchain," in IEEE Internet of Things Journal, vol. 7, no. 3, pp. 2343-2355, March 2020, doi: 10.1109/JIOT.2019.2958077. i

[6] S.P. Mohanty, V.P. Yanambaka, E. Kougianos and D. Puthal, "PUFchain: A Hardware-Assisted Blockchain for Sustainable Simultaneous Device and Data Security in the Internet of Everything (IoE)," in IEEE Consumer Electronics Magazine, vol. 9, no. 2, pp. 8-16, 1 March 2020, doi: 10.1109/MCE.2019.2953758. i

[7] https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html i