Localized State-Change Consensus in Immense and Highly Dynamic Environments

Linir Zamir ● and Mehrdad Nojoumian *

Department of Electrical Engineering and Computer Science, Florida Atlantic University, 777 Glades Rd., Boca Raton, FL 33431, USA; lzamir2016@fau.edu
* Correspondence: mnojoumian@fau.edu

Abstract: Consensus algorithms are the building block of any decentralized network where the risk of malicious users is high. These algorithms are required to be robust, scalable, and secure in order to operate properly. Localized state-change consensus (LSC) is a consensus algorithm that is specifically designed to handle state-change consensus, where the state value of given data points can dynamically change and the new value needs to be reflected in the system. LSC utilizes a trust measurement mechanism to validate messages and also enforce cooperation among users. Consensus algorithms, and specifically LSC, can be a practical solution for the immutable and secured communication of autonomous systems with limited computational resources. Indeed, distributed autonomous systems are growing rapidly and the integrity of their communication protocols for

execute it automatically by running a simple server that can operate independently. This is the primary reason that new blockchain applications have been recently developed in emerging fields such as robotic swarms and autonomous driving [3].

1.1. Consensus Algorithms

The blockchain data structure was first introduced in 1990 by Stuart Habert and W. Scott Strornetta [4]. The primary goal was to timestamp digital documents, making them tamper proof. Over the years, blockchain data structure has expanded to many other fields such as economy, e-voting, assembly line, etc. Many of these applications do not rely on cryptocurrency exchange, rather, the users exchange information in a decentralized fashion. Different decentralized applications are classified as either public or private. Public systems do not have any restriction on peers and they do not require any authentication process for joining the network or initiating trades. The public decentralized systems are maintained only by the public community, which means a higher level of decentralization. Private or permissioned systems operate under the leadership of a group, often called *consortium*,

plicable to a dynamic environment, where information is time-bound and can be changed every so often. A dynamic state-change environment was presented in [11], where the shared information is a flying configuration between agents in the system. This imple-

consensus. As shown in Equation (2), a higher reputation value lowers the required number of votes.

2. **Local Consensus:** LSC is designed to handle quick changes in immense and highly dynamic environments. This means that changes can happen quickly and simultaneously. Instead of achieving at least 51% consensus, LSC uses a localized consensus algorithm to verify and validate changes within the environment. Once a local consensus has been achieved, it can be used to achieve a 51% consensus, very much like any other consensus protocol.

3. **Data Validation:** LSC helps agents validate the content of the shared data. It utilizes both localized consensus and reputation values to guarantee any changes in the environment are reflected on the chain. As stated earlier, validation is related to the content of the transaction, e.g., a road is blocked, not just on the transaction execution.

To the best of our knowledge, there is no other consensus algorithm with these properties. Table 1 provides a comparison between state of the art consensus algorithms and LSC.

Table 1. Summary comparison of blockchain consensus algorithms. Partially taken from [14].

Consensus Algorithm	Designing Goal	Type	Verification	Local Consensus	Data Validation *
PoW [1]	Sybil-Proof	Permission-less	Work or Hash	No	No
PoS [2]	Energy Efficient	Permission-less	Stake	No	No
PoA [15]	Combination of PoS and PoW	Permissioned	Vote and Work	No	No
PoR [6]	Reputation-Based Consensus	Permissioned	Vote	No	No
pBFT [16]	BFT-Based Consensus	Permissioned	Vote	No	No
Raft [17]	Accessible Paxos	Permissioned	Vote	No	No
ISRaft [13]	Raft for Malicious Environments	Permissioned	Vote	No	No
LSC [This Paper]	Localized State-Change Consensus	Permissioned	Vote or Localized Vote	Yes	Yes

* Data validation on the content of the transaction, not on the transaction execution.

The remainder of this paper is organized as follows. Section 2 presents a summary of the LSC consensus algorithm with its state-change method, Section 3 covers the consensus algorithm in detail, Section 4 provides a detailed analysis and finally, Section 5 summarizes the concluding remarks.

2. Preliminaries, Notations and Definitions

LSC is a *scalable decision-making election-based* consensus algorithm. This means, unlike many other blockchain implementations that focus on achieving a consensus for the transactions on the ledger, LSC is designed to achieve consensus on the validity of the data that is written on the chain. In other words, LSC is designed to provide decision-based consensus by validating the information sent among users.

This paper nonetheless describes LSC under the context of a *permissioned* blockchain. Users must have prior knowledge of all other users and their signatures. When a new user joins the network, it goes through a series of enrollment processes where it is assigned a pair of cryptographic keys and a reputation value. Among the protocol's immutable communication, LSC also includes a state-change validation process where a group of users can change a state of some external information. It then goes through a validation process that includes the comparison of reputation values and state reading. If the validation data is accepted by the majority of users, the state changes and it is reflected in the blockchain. Two major properties that are required for achieving the decision-based consensus are

authentication and *reputation*. We assume a resilient reputation model [18] is utilized in our scheme for the trust management. For the sake of clarity, Table 2 defines our notations.

1.   **Authentication:**

Table 2. Notation Used.

Notation	Description
u_i	i th agent
T_i^j	Trust value assigned by *j* to *i*
T_i	Reputation value of *i*
ET_i	i th Election timeout
PK_{u_i}	Public-key of user i
Sig_i	Signature of i
d_j^i	State of data point j as seen by user i
$M[Sig_i, d_j^i, t_1]$	Message M signed by i with state s
L	Maximum number of nearest agents that can validate a message
h_i	Voting/signature threshold for approval of messages sent by agent i

3. Our Proposed Localized State-Change Consensus Mechanism

In this section, we cover the LSC consensus protocol as well as the five main communication algorithms. The main goal of LSC is to allow immutable and real-time decision making between agents on the LSC network. The second goal is to validate messages with adjustable reputation values that determine the validity of the agent. LSC achieves consensus over a set of messages through a Byzantine fault-tolerant (BFT) protocol. This protocol supports message authentication, partitioning and fast computation, which makes it ideal for resource-constrained devices.

3.1. Design Overview

LSC consensus protocol is a method where all agents hold an agreed upon ledger containing data. This ledger is constructed of blocks forming a chain. An agent can be in either of the three main roles: (1) follower, (2) candidate, or (3) leader. It is also possible to be an active validator, however, it is a temporary subrole. A leader is the only authority that can concatenate new blocks to the chain, acting as the *miner* of the newly concatenated block. Followers and candidates can add new information to a new block by sending the data to the leader.

LSC aims at achieving consensus among agents in a highly dynamic environment with a set of n agents where $U = \{u_1, u_2, \ldots, u_n\}$. Communications among agents are accomplished using five main algorithms. For the sake of simplicity, we assume that the communication is conducted on a secured channel without the possibility of having a man-in-the-middle attack. This, however, can be easily addressed in future works by using verifiable protocols. The five algorithms are:

- *RequestVote:* Initiated by a candidate agent and it is sent to all other agents. This message is sent as part of the *election process.*
- *StateChange:* Initiated by any agent and it is sent to L-nearest agents that can validate a given message. This message consists of the data point ID, new state, timestamp and signatures of agents approving the data. When a total of h_i agents sign the message sent by agent i, it is sent to the leader to be added to the next block.
- *AppendBlock:* Initiated by the leader at the end of every *leadership term.* This message contains the new block to be added to the chain. The agent receiving this message is required to respond with a signed approval message.
- *CommitBlock:* Initiated by the leader after receiving *AppendBlock* approval from the majority of agents.
- *Heartbeat:* Initiated by the leader and it is sent to all other agents. This message includes signed votes from the election, latest block number and leadership-term timer counting down to the end of the term. This message is sent periodically.

The number of required validators h_i for a *StateChange* message is defined as:

Definition 2. *Suppose n is the total number of agents in the network and $T_i \in [0, 1)$*

3. **Leader:** A leader is the highest authority who is responsible for sealing the next block on the LSC network. A leader is elected during the *election* process, and there can only be one leader for every newly generated block. Since the leader cannot reasonably be expected to maintain a fully synchronized communication with all other agents, it is expected that the followers are able to rebroadcast leader messages, excluding the heartbeat.

4. **Validator:** A validator's main purpose is to verify that a given data point's state has changed and that the *StateChange* message is true and valid. The process is fairly simple. Validators add their signature to the original *StateChange* message and then share it with nearby agents, who then also obtain the validator role.

   This is a dynamic role, and any agent can become a validator as long as it is close enough to the data point and to another validator agent.

3.3. Election Process

The first step of the protocol is to elect a leader in a *leader election* procedure. A leader is required to send periodic *heartbeat* messages to all other agents in the network. This message contains the signed signatures from the latest election, latest block number and leadership-term timer. This message acts as a leadership proof. Agents receiving this message validate the votes and check if the latest block number is at least equal to the block number on their database. When the timer reaches 0 or when no heartbeat messages have arrived over the expected period, a new election process begins.

At the beginning of an election, a follower agent changes its state to a candidate, and it begins sending out signed *RequestVote* messages, including the latest block number to all the agents it can contact. The latest block represents the term number of the chosen leader. For every new term, a new leader is elected. Agent u_m, who receives a *RequestVote* message, initially checks the following conditions:

- The agent did not receive any *heartbeat* messages from the current leader.
- The candidate agent is not the leader of the current term.
- The block number from the *RequestVote* message is at least equal to the agent's latest block plus 1.
- The message has a valid signature.

If all the conditions are met, the voting agent holds its vote for a fixed period of time equal to *election timeout* e. During this time, if any other *RequestVote* messages arrive, it once again checks if the conditions are met. If so, it then compares the reputation value of the candidate agents. At the end of e, the vote is sent only to the agent with the higher reputation value. Algorithm 1 illustrates the voting response process for an agent after receiving a *RequestVote* message.

The process ends when a candidate receives the majority of the votes, that is, at least $n/2 + 1$ votes for a network of n agents. At that point, the elected leader starts sending out *heartbeat* messages to the network, thus completing the election process.

Algorithm 1 RequestVote Response.

Require: *RequestVote* Message (RV_i) and Agent (u_i)
Ensure: Accept or Reject RV

1:   $t \leftarrow 0$
2:   $RV \leftarrow RV_i$
3:   *Decrease_reputation*(T_i^m)
4:   **while** $t \not\leqslant e$ **do**   ▷ Election Timeout
5:       **if** $RV.block\_num <= len(chain)$ **then**
6:           Reject RV
7:       **else if** $V(u_i.pk, RV, S(RV)) == reject$ **then**
8:           Reject RV
9:       **end if**
10:      **if** new RV_j from agent u_j **then**
11:         *Decrease_reputation*(T_j^m)
12:         **if** $T_i^m > T_j^m$ **then**   ▷ Compare reputation
13:            Reject RV_j
14:         **else**
15:            $RV \leftarrow RV_j$
16:         **end if**
17:      **end if**
18:      $t \leftarrow t + 1$
19:   **end while**
20:   Accept RV

3.4. Data Point's State Architecture—Local Consensus

LSC runs on an environment with a set of m data points $S = \{d_1, d_2, \ldots, d_m\}$. These data points are given per environment and can be changed when redeploying the system in different environments. A single data point can be in any number of states. For the sake of simplicity, we define these states as numerical values; however, it can be any data type. As in real life environments, a data point's state can change independently and randomly by conditions that the system is not necessarily aware of. It is up to the system to recognize the change and validate the data on the blockchain. The process goes as follows:

1. An agent u_k recognizes a change in the state of some data points d_j in the environment.
2. u_k then sends a *StateChange* message to the L-nearest agents, called **validators**. This message contains the data point ID, new state and a timestamp.
3. Any other agent receiving the *StateChange* message can decide whether to add its signature or not. When an agent adds his signature, it then sends the newly signed *StateChange* message back to the sender and to its nearest L agents.
4. Steps 2 and 3 are repeated until a total of h_i agents sign the original *StateChange* message. The value of h is calculated by the reputation values of the agents and by the total number of agents (Definition 2).
5. The h_i-th agent sends the message to the elected leader to add it to the next block.
6. Once a new block is mined, all other agents in the system update the state of data point d_j to the new state.

This process happens every time an agent recognizes a data point's state that is different from the state written on the blockchain. Agents who validate the state change and add a signature to the original *StateChange* message are called *validators* and are rewarded with reputation value upon a successful state change. The number of validators for each state change is given at the design level, and can be changed based on the reputation value of the sender, whenever the system redeploys, or by forking. The reason for this localized consensus is to prevent overloading the leaders with any state change and to prevent a possible distributed denial of service (DDoS) attack. Algorithm 2 covers the process of *StateChange* message signature. This process can also be referred to as the localized consensus process.

Agents near any state change are also likely to sign and send many *StateChange* messages from different agents who repeatedly send the signed message. If the message has already been signed by the recipient, it neither signs, nor sends it again. Another possible scenario is to have multiple h_i-length signed messages of the same origin that are sent to the leader. In this case, the leader only adds the first message and rejects any message with the same origin. This process can be represented by an h_i tree structure in which each node is an agent who signed the message and sent it to L other nodes. Figure 2 illustrates an example related to this matter.
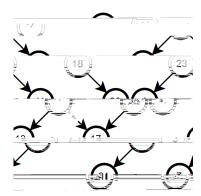

Figure 2. Representation of L tree structure.

Here, L = 2, meaning that at most 2 neighboring agents can validate a message, and a total of $h_{14} = b\frac{30 \ (1 \ 0.333)}{4} c = 5$ signatures are required to validate any messages sent by agent 14, where $n = 30$ and $T_{14} = 0.333$. The numbers within each node repre-

- **Previous Block Hash:** The hash value of the previous block.

The blocks are hashed and connected by the appropriate block number and the hashing value of the previous block. There can only be one leader for any single block. Followers can add data to the block by sending a signed *StateChange* message consisting of a list of the new data point's states. Each of these items holds the data point ID, new state value of the data point, original author of the message and the validators' signatures. The number of signatures per message is defined by h_i and can be different among agents based on their reputation values. Agents with higher reputation values have to provide more signatures, hence, more validators are required to verify the authenticity of the message. A leader adds the state-change message to the block if and only if:

- Enough validators have included their signatures.
- The originator of the message has not already been included in the block.

There can be a case where the leader does not obtain any *statechange* messages throughout its *leadership term*. In this case, the leader simply mines an *empty* block that does not contain any state-change data. Figure 3 shows an example for block properties in LSC.

Once a block is ready to be appended at the end of the *leadership term*, the leader sends the *AppendBlock* to all agents in the network. This message includes the next new block as well as the signed votes from the election in which it won. This prevents anyone from trying to disguise themselves as the leader. When an agent receives the *AppendBlock* message, it checks the following:

- Votes are legitimate.
- Hash value of the previous block and the block number fit the latest block in the chain. If not, the agent updates the chain from nearby agents.

agents from sending unauthorized messages and to prevent possible DDoS attacks on the network. Agents constantly change the reputation values of other agents over the network based on interactions in the network. Table 3 covers the reputation change for different messages in detail.

Table 3. LSC messages and their reputation costs.

Message	Reputation Cost	Details
RequestVote	7	Candidates send this message during the election process. This does not cost any reputation since we want all agents to have a chance of winning without the risk of constantly losing on reputation values.
StateChange	3	Initiated and sent to L-nearest agents, called **validators**. When a validator does not approve the state change, either by observing or by insufficient reputation values, it decreases the reputation value of the origin agent as well as agents who signed it. However, if the recipient approves the message and **signs** it, it increases the reputation value.
AppendBlock	3	This message is initiated by the leader and is part of the mining block. This message includes the signed votes from the latest election. Agents decrease the reputation value of the sender in the case if the signed votes are **not** valid.
CommitBlock	3	This message can only be initiated by the leader after the *AppendBlock* message was successful. Agents decrease the reputation value of the sender in the case the included approvals are **not** valid.
Hearbeat	7	This periodic message does not affect the reputation value.

Once a new block is committed on the blockchain, the reputation values of all agents who originated and signed a *StateChange* message on the new block increase as a reward in a process, called **block reputation increment**, where all agents update the state of the changed data point on the new block.

4. Technical Analysis or Our Proposed Solution

We now evaluate the effectiveness and scalability of LSC followed by security analyses.

4.1. Effectiveness

The effectiveness of the protocol can be measured by the total amount of messages that are sent among different agents on the network. LSC proposes a simple five-message protocol to limit the total required bandwidtwen1 16a can can can can cancap3(network)]TJTd [(12)

4.2. Scalability—A Storage Analysis

The generic assumption of the blockchain is that each agent can read a committed block and update the data point with the new state data. When a new block is committed, it is under the assumption of the LSC protocol that each agent stores the new block along with all previous blocks back to genesis. This assumption, however, can have some issues, especially when working with systems that can have limited memory capacity. This issue creates a scalability issue if the system is required to operate for a long period of time with many agents.

The memory requirement of each agent depends on the state-change content, the number of state changes per block and how frequently a new block is mined. More agents means more validators for any state change. As Definition 2 shows, an increase of n

Cryptography

Distributed Denial of Service Attack. Distributed denial of service attacks are a very common attack on systems that rely on communications. In general, this attack is an adversarial attempt to disrupt the normal activity of other agents by sending frequent messages to overwhelm a node and cause a traffic jam in the communication. In LSC, any message is priced with some reputation values. This means that a malicious agent's reputation value decreases as long as it continues sending messages. Once a reputation value reaches the lower threshold for an agent, any other messages sent by it is immediately dropped, i.e., stopping it from overwhelming the network.

5. Concluding Remarks

This paper provides a new information consensus algorithm for immense and highly dynamic environments using localized consensus with validations through reputation values. This algorithm can also be implemented in resource-constrained devices, such as autonomous systems or other smart devices that communicate over a network without the need for human intervention. Validation and authenticity of messages is achieved by first initiating a local consensus using cryptographic communication means and the usage of reputation values. The consensus can then be expanded globally when the state-change has been verified. LSC assures confidentiality, integrity and validity of messages on the blockchain among all agents.

In our future work, we will test the implementation of LSC in a real-world environment using communications that can be vulnerable to different side-channel attacks. We will validate the hardware-layer security of the system and measure its efficiency in an adversarial environment. We will also explore the possibility of adding more roles to the system to increase its efficiency and speed when implemented as a decentralized autonomous organizations (DAO).

References

1. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Bus. Rev.* **2008**, 21260.
2. King, S.; Nadal, S. Ppcoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. Self-Published Paper. 2012. Volume 19. Available online: https://bitcoin.peryaudo.org/vendor/peercoin-paper.pdf (accessed on 7 February 2022).
3. Yuan, Y.; Wang, F.Y. Towards blockchain-based intelligent transportation systems. In Proceedings of the 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, 1–4 November 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 2663–2668.
4. Haber, S.; Stornetta, W.S. How to time-stamp a digital document. In Proceedings of the Conference on the Theory and Application of Cryptography, Aarhus, Denmark, 21–24 May 1990; pp. 437–455.
5. Dwork, C.; Naor, M. Pricing via processing or combatting junk mail. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 16–20 August 1992; pp. 139–147.

6. Gai, F.; Wang, B.; Deng, W.; Peng, W. Proof of reputation: A reputation-based consensus protocol for peer-to-peer network. In Proceedings of the International Conference on Database Systems for Advanced Applications, Gold Coast, Australia, 21–24 May 2018; pp. 666–681.

7. Nojoumian, M.; Golchubian, A.; Njilla, L.; Kwiat, K.; Kamhoua, C. Incentivizing blockchain miners to avoid dishonest mining strategies by a reputation-based paradigm. In Proceedings of the Computing Conference (CC), Berkeley, CA, USA, 16–18 July 2018; pp. 1118–1134.

8. Jadbabaie, A.; Lin, J.; Morse, A.S. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Trans. Autom. Control.* **2003**, *48*, 988–1001. [CrossRef]

9. Beard, R.W.; Stepanyan, V. Information consensus in distributed multiple vehicle coordinated control. In Proceedings of the 42nd IEEE International Conference on Decision and Control, (IEEE Cat. No. 03CH37475), Maui, HI, USA, 9–12 December 2003; Volume 2, pp. 2029–2034.

10. Bellingham, J.S.; Tillerson, M.; Alighanbari, M.; How, J.P. Cooperative path planning for multiple UAVs in dynamic and uncertain environments. In Proceedings of the 41st IEEE Conference on Decision and Control, Las Vegas, NV, USA, 10–13 December 2002; Volume 3, pp. 2816–2822.

11. Ren, W.; Beard, R.W. Decentralized scheme for spacecraft formation flying via the virtual structure approach. *J. Guid. Control. Dyn.* **2004**, *27*, 73–82. [CrossRef]

12. McLain, T.W.; Beard, R.W. Coordination variables, coordination functions, and cooperative timing missions. *J. Guid. Control. Dyn.* **2005**, *28*, 150–161. [CrossRef]

13. Zamir, L.; Shaan, A.; Nojoumian, M. ISRaft Consensus Algorithm for Autonomous Units. In Proceedings of the 29th International Conference on Network Protocols (ICNP), Dallas, TX, USA, 1–5 November 2021; pp. 1–6.

14. Bamakan, S.M.H.; Motavali, A.; Bondarti, A.B. A survey of blockchain consensus algorithms performance evaluation criteria. *Expert Syst. Appl.* **2020**, *154*, 113385. [CrossRef]

15. of Authority | Governance | Diverse Ecosystem, V.F.W.P. Vechain Whitepaper: Vechain Builders. Available online: https://www.vechain.org/whitepaper (accessed on 7 February 2022).

16. Castro, M.; Liskov, B. *Practical Byzantine Fault Tolerance*; OsDI: New Orleans, LA, USA, 1999; Volume 99, pp. 173–186.

17. Ongaro, D.; Ousterhout, J. In search of an understandable consensus algorithm. In Proceedings of the 2014 fUSENIXg Annual Technical Conference (fUSENIXgfATCg 14), Philadelphia, PA, USA, 19–20 June 2014; pp. 305–319.

18. Nojoumian, M. Rational Trust Modeling. In Proceedings of the 9th Conference on Decision and Game Theory for Security (GameSec), Seattle, WA, USA, 29–31 October 2018; pp. 418–431.

19. Nojoumian, M. Trust, influence and reputation management based on human reasoning. In Proceedings of the 9th 4th AAAI Workshop on Incentives and Trust in E-Communities (WIT-EC), Austin, TX, USA, 25 January 2015; pp. 21–24.

20. Nojoumian, M.; Stinson, D.R.; Grainger, M. Unconditionally secure social secret sharing scheme. *Inf. Secur. Spec. Issue -Multi-Agent Distrib. Inf. Secur.* **2010**, *4*, 202–211. [CrossRef]

21. Nojoumian, M.; Golchubian, A.; Saputro, N.; Akkaya, K. Preventing collusion between SDN defenders and attackers using a game theoretical approach. In Proceedings of the Conference on Computer Communications Workshops (INFOCOM WKSHPS), Atlanta, GA, USA, 1–4 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 802–807.

22. Pourtahmasbi, P.; Nojoumian, M. Impacts of Trust Measurements on the Reputation-Based Mining Paradigm. In Proceedings of the 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), Paris, France, 27–30 September 2021; pp. 225–228.

23. Pourtahmasbi, P.; Nojoumian, M. Analysis of Reputation-Based Mining Paradigm Under Dishonest Mining Attacks. *Blockchain Res. Appl.* **2022**, *3*, 100065. [CrossRef]

24. Nojoumian, M.; Lethbridge, T.C. A new approach for the trust calculation in social networks. In Proceedings of the E-Business and Telecommunication Networks: 3rd International Conference on E-Business, Porto, Portugal, 26–28 July 2006; pp. 64–77.

25. Biswas, S.; Sharif, K.; Li, F.; Maharjan, S.; Mohanty, S.P.; Wang, Y. PoBT: A lightweight consensus algorithm for scalable IoT business blockchain. *IEEE Internet Things J.* **2019**, *7*, 2343–2355. [CrossRef]