Consensus under a Dynamic Synchronous Model

Kunal Kapoor and Jun Wan

January 3, 2022

1 Introduction

This paper deals with the problem of Byzantine Agreement. Byzantine Agreement is a core problem in the area of distributed computing as it deals with how multiple users in a network handle information. Specifically, the problem deals with how users can come to agreement about information being passed through a network given the potential of corrupted users that have the goal of disrupting consensus and tricking honest users. The basic setup of this problem goes as follows: There are **n** total nodes, or users, one of which is a designated **sender** node which outputs a bit b from set $\{0, 1\}$ to all the nodes. The goal of the protocol is to ensure that despite corrupt users acting in ways to disrupt consensus, there are two properties maintained:

- Consistency, which requires that all honest nodes must output the same bit (even when the sender is corrupt).
- Validity, which requires that all honest nodes output the sender's input bit if the sender is honest.

The variant of this problem that the paper focuses on is called "Sleepy Consensus". Traditional versions of this problem focus on the binary between honest users and corrupt users. However, the sleepy variant adds in a third kind of user, namely the "sleepy" user. A sleepy user is a user that is not active within the network and thus is unable to communicate with other users. This type of user is analogous to when in real world setups, a computer goes offline or a user stops participating in the network [1]. The main issue this presents to previous Byzantine Broadcast protocols is the violation of previous assumptions that were foundational to proofs of correctness for various sub-algorithms in past protocols.

Our results demonstrate that it is possible to achieve consensus under variations of the traditional problem. This paper explores the dynamic model which feature different assumptions about the abilities of the users and the information given to all users before the protocol.

2 Preliminaries

2.1 Problem Definition

There are n nodes in a network labeled from 1 to n. There are three possible states for these nodes: honest, corrupt, and sleepy. Going forward, the number of honest nodes will be represented by h, the number of corrupt nodes will be represented by f, and the number of sleepy nodes will be represented by s. Communication in this network is synchronous which means a message sent by an honest node in round r is received by the recipient node before/at the beginning of round r+1. There is a designated sender in the protocol that outputs a bit b to all nodes and by the end of the protocol the goal is for all honest nodes to have outputted the same bit, therefore achieving consensus. The protocol presented in this paper does not assume that nodes know the value of n or h beforehand but does assume there is a constant value of c which represents the minimum ratio of honest nodes to the total amount of nodes.

In this paper we deal with the dynamic model of the sleepy Byzantine Broadcast problem. This model features unique assumptions about starting information and properties of sleepy users that will be explained in a later section.

2.2 Technical Roadmap

The paper relies on key intuition established by Wan, Xiao Shi, and Devadas [2].

Section 3 features a change to the Trust Graph post processing algorithm and distrust messages. The lack of information about n, h, and sleepy nodes forces the creation of a different method to ensure the functionality of the Trust Graph.

Section 4 features a modification to the TrustCast protocol. The existence of sleepy nodes forces a change in the protocol to ensure it satisfies necessary correctness conditions.

Section 5 and 6 feature changes with respect to the model examined to the Byzantine Broadcast protocol which uses the building blocks established in Sections 3 and 4 to bootstrap consensus.

The largest contribution is the novel approach to adapting the post processing algorithm TrustCast protocol to the dynamic model. Afterwards, the high level intuition surrounding the actual consensus protocol is similar to the work in [2] and the relevant modifications are made rather than re-explaining the unchanged portions of the original protocol.

2.3 Dynamic Model

The Dynamic Model assumes that no user knows the value of n or h and only know a constant c such that $h * c > n$, meaning we have a consistent minimum ratio of honest nodes to total nodes. This model has sleepy users who have potential to wake up and rejoin the system but when sleepy, are completely disconnected and have no capacity to send messages. When woken up, users will receive all previous communication history (messages sent when they were not in the protocol).

2.4 Adversary

There exists an Adversary which controls the actions of all corrupt nodes and in the second variation, has the ability to adapt and turn certain honest nodes asleep as well as wake them up. In both variations of the problem, the adversary has the ability to turn honest nodes corrupt as long as the ratio k remains a minimum. When a node is asleep, the adversary can not send messages from it but can view the messages it would have sent before putting it asleep.

2.5 Modeling Setup

Important note: This section was taken directly from source 2 in the works cited. We believe that it provides useful information about the setup but was not in the bounds of our research.

We will allow setup assumptions as well as standard cryptography. Our protocol makes use of a public-key infrastructure and digital signatures, and for simplicity in this paper we assume that the signature scheme is ideal. We adopt a standard idealized signature model, i.e., imagine that there is a trusted functionality that keeps track of all messages nodes have signed and answers verification queries by looking up this trusted table. Under such an idealized signature model, no signature forgery is possible. When we replace the ideal signature with a real-world instantiation that satisfies the standard notion of "unforgeability under chosen-message attack", all of our theorems and lemmas will follow accounting for an additive, negligibly small failure probability due to the failure of the signature scheme — this approach has been commonly adopted in prior works too and is well-known to be cryptographically sound (even against adaptive adversaries). For other cryptographic primitives we adopt, e.g., verifiable random functions, we do not assume idealized primitives since the computationally sound reasoning for these primitives is known to have subtleties.

3 Trust Graph

3.1 Overview

The Trust Graph is the crux of the protocol. The Trust Graph, as its name suggests, is a graph mapping out the relationships between all the users. Specifically, an edge between two users signifies mutual trust whereas a lack of an edge between two users signifies distrust. Throughout the protocol, each user's trust graph changes as it processes new information. In order to use the trust graph, the diameter of the trust graph is directly tied to the round complexity of the protocol. This is because the longer the diameter of the trust graph, the more rounds that need to occur to ensure that messages from all nodes are being received.

At the beginning of the protocol, each node's trust graph is a complete graph as there has been no reason to distrust another node. As the protocol goes on and nodes start to distrust each other, they communicate distrust via a distrust message. A distrust message is passed on to other neighbors in the trust graph and users remove the edge between the two nodes that distrust each other as they receive the message. It is important to note that a distrust message about the mutual trust between two nodes can only be issued by one of the two nodes in question or else corrupt nodes can manipulate the distrust message to disconnect two honest nodes.

To accommodate sleepy nodes, if node k receives no message from node j, it will not remove node j unless in the next round it receives a message from one of its neighbors that was originally sent to the neighbor by node j in the previous round. This has the effect of ensuring if a node is put to sleep, it will not be removed from the trust graph.

As the nodes start to distrust each other, the complete trust graph with an initial diameter of 1 starts to increase. This can quickly lead to graphs which have large diameters. In order to deal with this problem, we need to employ a post-processing algorithm. The purpose of the post-processing algorithm is to change the trust graph in a way such that the diameter decreases without removing edges between honest nodes.

It is also important to note we define S_k has the set of all nodes that are a distance of k away from the node who's trust graph is being examined. In this paper, any S_k is called a layer.

3.2 Post Processing Algorithm

In this section we explain the post-processing algorithm and show proof that it keeps the diameter within bounds of $O(\frac{n}{h})$ and never removes an edge between 2 honest nodes.

The post processing algorithm works as follows: Find the lowest sum of the number of nodes in any two consecutive layers and remove edges between those two layers. In other words, find k such that $|S_k| + |S_{k+1}|$ is minimal and remove all edges between S_k and S_{k+1}. This operation has the visual effect of completely cutting the graph at a layer. This process is then repeated a certain number of times until there the diameter is guaranteed to be within bounds of $\frac{n}{h}$. This algorithm will be called the "Minimum Layer Sum Removal Operation". Each time a node uses this algorithm it must show some sort of proof to its neighbors in order to convince them to remove the same edges. The proof it shows comes in the form of displaying its trust graph to other users and "hello" messages which will be explained more in depth later.

It is important to note that this algorithm offers an improvement over previous post processing algorithms. Previous algorithms tended to remove all edges between nodes who's shared neighbors were less in quantity than h which is an unviable strategy as the user doesn't know the exact value h beforehand.

The proof of correctness will be broken down into 2 parts:
(1) Prove that the Minimum Sum Layer Removal keeps diameter within the bounds of $\frac{n}{h}$.
(2) Prove that the Minimum Sum Layer Removal never removes edges between honest nodes.

Theorem 1 *Minimum Sum Layer Removal keeps diameter within bounds of* $\frac{n}{h}$.

We know the minimum sum layer removal operation always decreases the diameter of the trust graph as it removes all edges between two layers thus making the graph discard a fraction of its layers. The protocol uses this operation continuously until it reaches a value below the maximum possible value of $\frac{2n}{h}$ such that if the operation were to be applied one less time, the value would go above $\frac{2n}{h}$. The maximum possible value of $\frac{2n}{h}$ can be computed by nodes as they are given the fact the ratio of h to n is at least some constant c. This means the ratio from n to h, i.e. $\frac{n}{h}$, is at most $1/c$ and thus the max value of $\frac{2n}{h}$ is $\frac{2}{c}$. The protocol applies the operation such that the diameter becomes less than $\frac{2}{c}$ which ensures the diameter is bounded by the maximum value of $\frac{2n}{h}$.

Theorem 2 *Minimum Sum Layer Removal never removes edges between honest nodes.*

We will prove by contradiction. Assume a corrupt node attempts to remove edges via the minimum sum layer removal process such that it removes edges between 2 honest nodes. This means the min layer sum would be at least h. This is because if there are honest nodes in both of the layers they must be connected to all of the other honest nodes because there is no point in time in which they have previously distrusted each other. We know that when we apply the operation, the diameter is greater than the maximum possible value of $\frac{2n}{h}$ otherwise we wouldn't need to apply the post-processing algorithm. The average sum of any two layers is less than h which means the minimum sum of two layers must also be less than h which proves we can never remove edges between honest nodes. Even with sleepy nodes, we have a guarantee on the ratio c and h as we never remove the sleepy nodes from the trust graph, ensuring they do not impact the calculations.

4 Trust Cast Protocol

4.1 Overview

The TrustCast protocol is a key building block for the eventual Byzantine Broadcast protocol.

The TrustCast protocol uses a designated sender s to send a message which gets passed along various trust graphs for d rounds. At the end of the d rounds, each user outputs the message they received if the sender is still viewed as online in the user's trust graph or mark the sender as dishonest or potentially sleepy and if applicable, remove the sender from their trust graph. It is important to note that the TrustCast protocol doesn't guarantee consistency as certain honest users may output different messages than others due to the ability for a corrupt sender to send varied messages. However, if the sender is honest, all honest nodes should output the same message m.

To better understand the protocol we consider 2 cases. First, if s is a direct neighbor of some node u. We know that since they are neighbors they trust each other and that u is expecting to receive a message in the first round. If u doesn't receive a message, it knows s is either sleepy or corrupt. Given u can't outright say s is corrupt, it would mark s as potentially sleepy and send out a corresponding message similar to a Distrust message indicating that. If in the next round u gets information s was not sleepy, it would then distrust s and send out the corresponding distrust message. The second case is if s is a distance r away from u in u's trust graph. In this case, we know that after r rounds, u expects to receive a message from s. If u doesn't receive a message from s after r rounds, it knows that all nodes a distance of $r - 1$ or less are sleepy or corrupt. This is because if one of those nodes was honest and online, u would have received a valid message by round r. Given this, our goal is to prove the following:

- At the end of the TrustCast protocol, any honest node either receives a message from s or removes s from its trust graph or marks s as potentially sleepy.
- In the TrustCast protocol, we never remove edges between two honest

nodes in any honest node's trust graph.

4.2 Proof of Correctness

Theorem 3 *At the end of the TrustCast protocol, any honest node either receives a message from s or removes s from its trust graph or marks s as potentially sleepy.*

Using the intuition above about action taken against nodes a distance of less than r from s in round r when u hasn't received a message, we can formulate a proof. Specifically, setting r to $d + 1$ tells us that if we haven't received a message by round $d+1$, s is a distance of at least $d+1$ from u or that s is sleepy or corrupt which the distrust and sleepy messages other nodes in the trust graph would have passed on. We know the former must be incorrect as the trust graph must have a diameter of at most d proven in section 3.2 which means s must be removed from the trust graph or the second case is true meaning that s is sleepy.

Theorem 4 *In the TrustCast protocol, we never remove edges between two honest nodes in any honest node's trust graph.*

We know that the only case in which distrust messages are sent are ones in which a node doesn't propagate a message but is proven to be online and therefore not sleepy. However, honest nodes that are online always propagate messages to each other which means that the conditions for a distrust message to be sent by another honest node will never occur therefore meaning edges between two honest nodes in the trust graph can never be removed.

5 Dynamic Model

5.1 Overview

Applying the post processing algorithm and TrustCast protocol to create a full consensus protocol relies on 3 separate phases. The three phases are described as follows [2]:

- Propose: the leader uses the TrustCast protocol to share the freshest commit evidence it has seen.
- Vote: each node uses the TrustCast protocol to relay the leader's proposal it receives in the propose phase. At the end of the vote phase, each node checks whether it can construct a commit evidence.
- Commit: nodes use the TrustCast protocol to share their commit evidence (if any exists).

To accomodate for previous changes, the phases need to be modified. Before that happens, we need to deal with the fact that users don't have a common value of n to use.

5.2 Dealing with n being dynamic

The first problem to solve is to figure out the value of n used for the protocol as it is necessary for both the Trust Graph and TrustCast portions to function correctly. To get a good estimate of n we run the TrustCast protocol for the first d rounds. We know $\frac{h}{n} > c$ so $\frac{2n}{h}$ is upper bounded by $\frac{2}{c}$ which means we have an upper bound of d to use. We know after d rounds every potential online user has had the ability to send a message. If a corrupt user decides not to send a message to some honest users but does send a message to other honest users, the honest users that weren't sent a direct message by the corrupt user will still figure out about its existence through the honest users who did receive a direct message from the corrupt user in question. This proves any user known to one of the honest users will be known to all of them.

5.3 Propose Phase

The propose phase can remain the same given it uses the adapted TrustCast protocol described in section 4. Let's call the set of users online and honest in round r O_r. We know if the leader L_e is in O_r for r in [0, d] the proof in the paper is applicable. Thus we have to consider two cases to adapt to the sleepy model.
1) L_e is sleepy before it is selected as leader. If L_e is sleepy before it is selected as a leader, the TrustCast protocol ensures it will not be removed from the trust graph which is proven in section 8. It will not send a message so essentially nothing will be proposed and thus this is not a situation which could disprove correctness of the propose phase proof.
2) L_e becomes sleepy in some round after it is selected as leader. In this case, L_e was in O_r where r is the round when it was selected leader so we know it was able to TrustCast the propose message correctly. Based off this, the current TrustCast protocol ensures L_e will remain in the trust graph so all honest users will receive the message.

5.4 Vote Phase

The key modification to the vote phase is changing the condition for how many valid vote messages are needed to commit. With the addition of sleepy nodes, the idea that a user can receive a vote message from every node that remains in its trust graph is no longer true as there are users that are honest but sleepy meaning they can't send a vote message but shouldn't be removed from the trust graph.
It is important to note that we can't treat sleepy users as corrupt users as that would result in their removal from the trust graph which would become problematic as they could later come back on online and be completely isolated from other honest nodes. A strategic Adversary could selectively disconnect each individual honest node from the others which would make it impossible to attain consensus.

The solution is to use the ratio c to create a weaker condition for accepting a group of vote messages. We know that $(1-c) * n + 1$ is the amount of votes we need on a certain bit to accept it as $(1-c) * n + 1$ represents $f + 1$. However, the ratio c is between online honest nodes and all nodes which includes sleepy nodes which means that insofar as we don't remove sleepy nodes from the trust graph, it could be impossible to ever receive $f + 1$ votes because a large majority of n are sleepy and wouldn't be able to vote at all. Instead, we can note that if we disregard any node marked as "potentially sleepy" by the TrustCast protocol, we can still get a value of $f + 1$ that is attainable. We know if we disregard nodes marked as "potentially sleepy" the ratio between online honest nodes and total online nodes is also at least c. This is because every node marked as potentially sleepy is either a corrupt node or a sleepy node which doesn't change the number of online honest nodes but does decrease the value of total online nodes. We can then set $f + 1$ equal to $(1 - c) * k + 1$ where k represents the number of total online nodes. This can then be equated with the original vote phase as we know all k users have the potential to send vote messages and we have an achievable condition for committing on a certain bit b.

5.5 Commit Phase

The commit phase is essentially the same except we substitute the calculated value of $f + 1$ from the Vote Phase as the condition for committing on a bit.

6 Conclusion and Future Work

In this paper, we have been able to adapt traditional Byzantine Broadcast protocols to a dynamic model. So far, we have done this via modifying the important building blocks as the consensus protocol as well as the protocol itself. It will be interesting to see in future work if our results can hold when applied to models with different conditions, such as those which allow users to leave and rejoin the system arbitrarily or have weaker guarantees on starting information.

7 Acknowledgements

I would like to thank my research mentor, Jun Wan, who has guided me through this project as well as suggested the initial topic. I am also thankful for the MIT PRIMES program for giving me this research opportunity.

8 Works Cited

[1] Pass, Rafael Shi, Elaine. (2017). The Sleepy Model of Consensus. 380-409. 10.1007/978-3-319-70697-9_14.

[2] Wan J., Xiao H., Shi E., Devadas S. (2020) Expected Constant Round Byzantine Broadcast Under Dishonest Majority. In: Pass R., Pietrzak K. (eds) Theory of Cryptography. TCC 2020. Lecture Notes in Computer Science, vol 12550. Springer, Cham. https://doi.org/10.1007/978-3-030-64375-1_14