A Simple Reduction from State Machine Replication to Binary Agreement in Partially Synchronous or Asynchronous Networks

Abhinav Aggarwal[*,†], Yue Guo[*]
[*]Cornell University, Ithaca, NY
[†]University of New Mexico, Albuquerque, NM

Abstract

The recent advent of blockchains has spurred a huge interest in the research and development of numerous cryptocurrencies as well as understanding the fundamental concepts that underly this technology. At the heart of this design is the classic state machine replication protocol in which a group of n machines (out of which f are Byzantine) want to agree on an ever-growing log of transactions.

In this paper, we present a simple blackbox reduction from state machine replication (SMR) to the classical binary agreement (BA) protocol on top of a fully decentralized network. We consider both synchronous and partially synchronous/asynchronous settings for our reduction. We also present an algorithm for a reduction from BA to SMR, thus establishing an equivalence between the two. In each of these settings, we analyze our algorithms with respect to the required security properties.

Although there is prior work that establishes these reductions, our solutions are simpler (at the cost of efficiency) and useful from a pedagogical point of view.

1 Introduction

The Internet today has provided a fast and reliable way of communication between any group of people independent of time and their location. With high guarantees of success, this communication is, on an average, a textbook example of what modern technology can aim to achieve with respect to bringing the world closer. However, there exist certain applications for which this feat of technology is just not enough until some demanding conditions are met. One such example, amongst many others, is the recently introduced and highly popular cryptocurrency technology where thousands of people have invested billions of dollars on a purely digital form of monetary exchange that at its very best demands strict assumptions on the underlying network it can securely and efficiently run on. At the heart of these assumptions is the knowledge of the exact (or an upper bound on the) timing delay for a message to propagate across the network. As shown independently by Garay et al. and Pass et al. [Gar+16; PSS17] in their analysis of the underlying blockchain protocol, this assumption is in fact a fundamental requirement for any cryptocurrency to function securely and avoid attacks by a clever network adversary. Such a model in which the knowledge of network delay forms a parameter for the design of algorithms (and is hence, assumed to be known to all participating nodes) is known as the *synchronous* model of communication in the distributed computing parlance.

Although at first it may seem like a reasonable assumption given the technological advancement and the success of Internet ever since it was first introduced, the assumption of synchrony is often far from reality. Unexpected disturbances in the network, natural phenomenon or even adversarial intervention can cause violations of any fixed bounds on the network delay and make synchronous protocols come to a halt. In situations like these, a weaker model that relaxes this requirement of the knowledge of network delay (yet assuming the existence of an unknown finite bound) comes to rescue. First introduced by Dwork, Lynch and Stockmeyer in their seminal paper [DLS88], *partial* *synchrony* has since been a popular choice of communication model that allows flexibility against assuming strict message delivery times by requiring the participating nodes to know that a finite bound on the message delay exists while being oblivious to the actual value of this bound. This lies somewhere in the spectrum between pure synchrony and asynchrony, where the latter only promises eventual message delivery without any guarantee on the delivery times of the messages sent.

In this partially synchronous world, it then becomes obvious to ask if certain fundamental problems like Byzantine agreement or State machine replication become inherently harder than their synchronous analogs. While the former forms the basis of any distributed decision making in the strongest possible fault model (Byzantine), the latter is a more practically motivated problem that achieves fault tolerance through appropriate replication of the working elements of a system that constitute its *state* in the given protocol execution. Ever since it was first introduced and studied in depth [Sch90], state machine replication has formed the basis of many large scale protocols including the modern blockchain technology. Keeping the financial aspect of the cryptocurrency aside, the blockchain underlying the ledger maintenance is essentially a replicated state machine, which is updated by thousands of nodes in parallel in a tamper evident undirectional manner. We, thus, seek an answer to the following basic question:

Is it fundamentally harder to achieve state machine replication than byzantine agreement in the absence of network synchrony?

This question was first answered in the negative by Liskov et al. [CL+99] in their celebrated paper on practical byzantine fault tolerance where they gave an algorithm to realize state machine replication under partial synchrony using Byzantine agreement as a primitive, thus establishing an equivalence in terms of computational hardness of the two problems (the reverse reduction is (comparatively) trivial). However, the protocol in the paper is highly complex to understand and unfortunately, falls victim to highly error-prone implementations as a result. Just short of two decades later, a cleaner and highly optimized reduction was provided by Miller et al. [Mil+16] for asynchronous networks. We take motivation from these two breakthrough papers and all the followup research to seek a simple, cleaner reduction without any optimization primitives to study the fundamental connection between state machine replication and byzantine agreement and ask ourselves the fundamental requirements and complexities of such an equivalence. Our algorithms on one hand may seem far less efficient and resource intensive than the ones provided in [Mil+16] or other papers, nevertheless, our reduction enjoys simplicity and easy adaptability with numerous scope of parameter tuning given any application at hand.

2 Preliminaries

2.1 Model

In this report, we focus mainly on *partial synchronous communication*, which refers to a network environment with message delay bounded by an unknown upper bound Δ in terms of number of rounds or real time. There are other versions of partially synchronous setting. As proven and first introduced in [CL+99], these versions are essentially equivalent. We also assume that this bound on the message delay holds no matter how long the messages are. Commensurate with the reasoning by Pass and Shi in [PS17], such as assumption is justified by assuming that the network is sufficiently connected and has sufficiently many honest nodes to ensure Δ delivery time for the messages. Furthermore, assuming that the lengths of all the inputs provided by the environment are bounded by some function of a security parameter, the assumption of ensured bound on the delivery time by the sufficiently connected network is justified.

We assume that honest nodes can send messages to all other honest nodes. The adversary is in charge of scheduling message delivery. However, it cannot modify the contents of messages broadcast by the honest nodes, but it can reorder and delay them subject to the constraints described above (bounded message delay). The adversary is allowed to send messages to a subset of honest nodes but not all of them. All the communication channels are authenticated and the identities of every node is known from the beginning of the protocol to all other nodes. We assume the existence of a public key infrastructure (PKI) to allow message authentication. This is akin to the permissioned-classical setting from [PS17], in which the environment is required to spawn all nodes upfront and inform all honest nodes the identities of all nodes spawned. Henceforth, without loss of generality, we assume that the spawned nodes have identities $0, 1, \dots, n-1$ respectively.

As the protocol is event driven rather than real time, we use the term *epoch* to denote the time unit of protocol execution. We purposely avoid the use of the term *round* here, since it usually refers to a certain fixed length of real time in the distributed computing parlance.

2.2 Building Blocks

In our reduction from state machine replication to binary agreement, we use reliable broadcast (RBC) and binary agreement (BA) as the building blocks for the state machine replication protocol. We define each of these primitives and give the abstractions and properties for these protocols in the partially synchronous setting. In the reduction from binary agreement to state machine replication, we use the same abstractions and definitions.

2.2.1 Reliable Broadcast

Reliable broadcast (RBC), first introduced as the *Byzantine Generals Problem* by Lamport [LSP82], provides a way for a designated node (called the sender) to send a message to all other nodes. We work with the following input and output abstraction for RBC.

- **Input:** The sender (known a priori) receives an input value v from environment.
- **Output:** If the sender is honest, then every honest node P_i outputs a value v'_i.

Note that RBC only guarantees an output when the sender is honest. A protocol for RBC requires that all honest parties deliver the same set of messages and that this set includes all messages broadcast by honest parties, without guaranteeing anything about the order in which messages are delivered [Cac+01]. Thus, except with negligible probability of the execution traces, the following security properties are guaranteed.

- **Consistency:** If any two honest nodes outputs v and v', then $v = v'$.
- T_{tot}**-Totality:** If some honest node outputs v in epoch t, then every honest node outputs v before epoch $t + T_{\text{tot}}$.
- T_{val}**-Validity:** If the sender is honest and inputs v in epoch t, then all honest nodes outputs v before epoch $t + T_{\text{val}}$.

We let $T_{\text{RBC}} = \max\{T_{\text{tot}}, T_{\text{val}}\}$ to denote the maximum time it takes for an instance of RBC to complete.

For our reduction in Section 4.1, we instantiate RBC with Bracha's reliable broadcast protocol from [Bra84] which works for asynchronous networks and tolerates $f \leq \lceil n-1 \rceil / 3$ Byzantine failures. The different steps of this protocol are presented in Fig. 1. The sender begins by sending the message to everyone. Each honest node, upon the receipt of this message, echoes it to everyone else and starts collecting the echoes from other nodes. Once sufficiently many echoes have been collected, the (honest) node believes that enough honest nodes have received the message from the sender and it sends a message to indicate that it is ready to produce an output. When a required number of ready messages have been received, the honest node produces the output and sends a notification to everyone about this. It terminates the protocol thereafter.

The notification of delivery (output) is sent in the final step purely for the knowledge of the other honest nodes that have not terminated yet. As will be seen in our reduction, such notifications will become important checkpoints for making progress in our algorithm since partial synchrony/asynchrony prohibits dependence on real time.

The sender P_s sends the message $(\texttt{send}, v)$ to everyone. Each honest player does the following:

- Upon receiving $(\texttt{send}, v)$ from P_s, send $(\texttt{echo}, v)$ to everyone.
- Upon receiving $2f+1$ messages $(\texttt{echo}, v)$ and not having sent $(\texttt{ready}, v)$, send $(\texttt{ready}, v)$ to everyone.
- Upon receiving $f+1$ messages $(\texttt{ready}, v)$ and not having sent $(\texttt{ready}, v)$, send $(\texttt{ready}, v)$ to everyone.
- Upon receiving $2f+1$ messages $(\texttt{ready}, v)$, send $(\texttt{deliver}, v)$ to everyone, output v and terminate.

Figure 1: Bracha's Reliable Broadcast Protocol [Bra84]

2.2.2 Binary Agreement

The standard notion of Byzantine agreement seeks a binary decision value derived off the inputs of its participating nodes. Such protocols, referred to as *Binary agreement protocols* (BA), often form fundamental building blocks for more involved distributed algorithms. The binary agreement protocols have the following input and output abstraction:

- **Input:** Every node P_i receives an input bit b_i from environment.
- **Output:** Every honest node P_i outputs a bit b'_i.

Except with negligible probability over the execution traces, a binary agreement protocol satisfies the following properties. As is usual with

- **Agreement:** If any honest node outputs the bit b. then every honest node outputs b.
- **T_{BA}-Termination:** If all honest nodes receive input in round T, then every honest node outputs a bit before round $T + T_{\text{BA}}$.
- **Validity:** If any honest node outputs b, then at least one honest node receives b as input.

We instantiate this algorithm with the randomized binary byzantine agreement protocol given by Toueg in [Tou84]. Although they assume a trusted dealer who distributed a common random bit to all the players as required, such a functionality can be realized using private coins as well [CGR11]. Thus, we assume that each player P_i has a private coin coin_i which produces a random binary output each time it is called.

2.3 State Machine Replication

State machine replication (SMR) (for Byzantine faults), also referred to as *atomic broadcast* or *total order*, was first introduced by Lamport in [Lam84] and abstracted as a general protocol for fault tolerance and decentralized control by Schneider in [Sch90]. In a state machine replication protocol, a set of nodes seek to continuously agree on an ever-growing *linearly ordered log* over time. The input and output abstraction for such protocols is defined as following:

- **Input:** Every node keeps receiving transactions tx from the environment.
- **Output:** Every honest node P_i outputs a sequence of batchs of transactions as the log log_i^ℓ at the end of every epoch ℓ.

In every epoch, an honest and online node receives as input a set of transactions txs from the environment at the beginning of the epoch and outputs a log collected thus far to the environment at

Each honest player P_i with input b_i does the following:

- $r \leftarrow 0$
  while not decided, do:
  - Send the message $(\texttt{ready}, r, v_i)$ to everyone.
  - Upon, receiving $(\texttt{ready}, r, v_j)$ messages from $n - f$ players, let $\Pi \leftarrow$ set of received $(\texttt{ready}, r, v_j)$ messages.
  - Let $v \leftarrow$ majority value in Π.
  - Using reliable broadcast, send the message $(\texttt{commit}, r, v_i, \Pi)$ to everyone.
  - Upon receiving $(n - f)$ messages $(\texttt{deliver}, (\texttt{commit}, r, v_j, \Pi))$ from the reliable broadcast, set $m \leftarrow$ value v' that is contained most often among the these messages, $c \leftarrow$ the number of messages that contain value m and $r \leftarrow \mathsf{coin}_i$.
  - If $c = n - f$, then update $v \leftarrow m$, else $v \leftarrow r$.
  - If $c \geq t + 1$ and $m = r$, then send the message $(\texttt{decide}, v)$ to everyone. Output v and terminate. Else, $r \leftarrow r + 1$.
- Upon receiving $f + 1$ messages $(\texttt{decide}, v)$ and not having sent $(\texttt{decide}, v)$ so far, send the message $(\texttt{decide}, v)$ to everyone. Output v and terminate.

Figure 2: Toueg's Binary Agreement Protocol [Tou84]

the end of the round. Except with negligible probability over the execution traces, SMR ensures both *consistency* (all honest nodes' logs agree with each other although some nodes may progress faster than others) and *liveness* property (transactions received by honest nodes as input get confirmed in all honest nodes' logs quickly). We formally define these properties below. The notation $\preceq$ here denotes the "is a prefix of" relation.

- **Consistency:**
  - (*Common Prefix*) For all node_i which is honest by the end of epoch r, and for all node node_j which is honest by the end of epoch t, either $\mathsf{log}_i^r \preceq \mathsf{log}_j^t$ or $\mathsf{log}_j^t \preceq \mathsf{log}_i^r$.
  - (*Future Self-Consistency*) For each node_i which is honest by the end of epoch r and t where $r \leq t$, we have $\mathsf{log}_i^r \preceq \mathsf{log}_i^t$.
- **T_{conf}-Liveness:** If node_j that is honest in epoch r receives some set of transactions tx in epoch r, or if $\mathsf{tx} \subset \mathsf{log}_j^r$, then for all P_i which is honest in epoch $t \geq r + T_{\text{conf}}$, it must be the case that $\mathsf{tx} \subset \mathsf{log}_i^t$.

2.4 Reduction in Synchronous Network

We first present a simple reduction from SMR to BA in the synchronous setting. Here, the problem of realizing State Machine Replication based on one-time consensus protocol is easier. As we have the concept of round, we can use Byzantine Agreement only without using Reliable Broadcast since the latter does not guarantee termination when the sender is corrupt. There will, thus, be no need for a reliable broadcast protocol. The reduction we present here is presented in detail (in the context of blockchain) in the paper by Pass and Shi [PS17]. We assume a shared public key infrastructure among all players. We instantiate the byzantine agreement protocol by the multi-valued version of the classic Dolev-Strong binary agreement protocol [DS83] for this reduction. In this setting, we call the time for executing one instance of Dolev-Strong protocol an *epoch*. For each epoch ℓ, the protocol runs as following:

- The n players initiate n instances of the Dolev-Strong protocol, DS_j^ℓ for $j \in \{1, \dots, n\}$, simultaneously. P_i acts as the sender of the i-th instance DS_i^ℓ with the input received from the environment.
- Every player P_i collects its outputs of all DS_j^ℓ into batch_i^ℓ and updates $\mathsf{log}_i^\ell = \mathsf{log}_i^{\ell-1} \cup \mathsf{batch}_i^\ell$.

The consistency and liveness property of this reduction trivially follows from the security properties of the Dolev-Strong protocol. We omit details of the proof here, however we note the the reduction above makes use of multiple instances of the multi-valued agreement protocol (DS), which requires each instance to be spawned with a unique session identifier for secure protocol composition. We do not include the identifiers in the protocol above for sake of clarity, but the detailed reduction is available in Section 2.4.2 of [PS17]. The paper also shows that assuming a secure signature scheme, this reduction preserves the properties of state machine replication.

3 Related Work

The concept of partial synchrony was introduced in 1988 by Dwork, Lynch and Stockmeyer in their seminar paper [DLS88] for consensus without any knowledge of the bound on the message delay over the network. The paper also presents tight lower bounds and optimal algorithms for consensus in this model as well as for the alternate definitions of partial synchrony, which were proved to be essentially equivalent. Unlike asynchrony, approximate notions of time can be achieved in this model and hence, it forms a strictly stronger assumption. Moreover, algorithms that parametrize themselves with the network delay and thus assume synchrony can fail to work under partial synchrony if the assumed bound on the message delay is conservative. This renders partial synchrony strictly weaker than the synchronous communication model.

Shifting gears from the network communication model, a core distributed protocol used in numerous practical and pedagogical applications is state machine replication, which was first studied in detail by Schneider in 1990 [Sch90]. Also known as *total order* or *atomic broadcast* [Cri+95], this protocol is used to make a set of participating nodes agree on an ever growing linearly-ordered log that satisfies consistency and liveness under appropriately defined settings. For more than a decade, companies like Google and Facebook have deployed Paxos-style protocols [Bur06; JRS11; Lam06] to replicate a significant part of their computing infrastructure. With the advent of cryptocurrencies, large scale deployment of state machine replication is in high demand, which causes a clear distinction in the *classical* approach to state machine replication with the modern *blockchain-style* approach [PS17]. While the former uses celebrated protocols like PBFT [CL+99] and Byzantine-Paxos [Lam06], it often suffers from poor implementation, maintenance and reconfiguration for large scale settings as a result of their notoriously complicated descriptions. As a contrast, the modern setting for blockchain-style state machine replication, represented by Nakamoto's original blockchain [Nak08], are relatively simple and tolerate minority corruptions as opposed to the super-majority of honest nodes required by the classical protocols.

Variants of atomic broadcast were subsequently proposed to realize the protocol under different practical requirements. In the early years of the millenium, two papers [KS05; RC05] studied atomic broadcast from the perspective of reducing communication cost by optimistically running a broadcast protocol under *normal* network conditions and switching to a slower mode when under attack. The novelty of the works lie in consistency and liveness preserving recovery from the slow mode while preserving the average or amortized cost of communication.

While only recently the paper by Pass and Shi [PS17] provides a clean reduction of state machine replication from byzantine agreement in the synchronous setting, from the very early years of this century, Cachin et al. provide an excellent survey of reductions of the different broadcast protocols in purely asynchronous environment using cryptographic primitives [Cac+01]. Their paper also provides the first randomized atomic broadcast protocol based on a novel efficient multi-valued asynchronous Byzantine agreement primitive with an external validity condition. In the spirit of such a reduction, follow up works like those of connecting Paxos to generalized consensus by Lamport [Lam05] and optimizing the latency of realizing state machine replication through the agreement primitive [SB12] were proposed. Further optimizations were made by Miller et al. in their paper [Mil+16] which provides a clean modular realization of atomic broadcast using the asynchronous common subset primitive realized off a simple binary agreement protocol. Being closest to our work, this paper tries to address an attack on partially synchronous protocols by exploiting their sensitivity on bounded network delay and provides a cryptographically-aided reduction in the purely asynchronous setting. The follow up work by Malkhi et al. [AM+17] discusses this reduction in the blockchain setting.

Although most of the reductions discussed above along with the numerous other attempts (that we omit mentioning for the lack of space) work in the strictly weaker communication model of asynchrony that will also extend to the partially synchronous setting, we seek to work with the minimal requirements for such a reduction or equivalence to hold in the stronger model and ask ourselves if liveness guarantees that are better than *eventual* deliveries (like in the asynchronous model) can be provided with a simple protocol. We note that our reduction is not entirely novel in its problem statement, however, its simplicity and lack of distracting optimizations are helpful from a pedagogical standpoint, which is quintessential in understanding the foundations of consensus in this setting.

4 Reduction in in Partially Synchronous Network

4.1 From State Machine Replication to Binary Agreement

As shown in Section 4.3, there exists a reduction from reliable broadcast to binary agreement. So in this section, we give a reduction from state machine replication to binary agreement by describing a SMR algorithm built with binary agreement protocol BA and reliable broadcast protocol RBC with n nodes, f of which are corrupted. The building blocks are used as black box.

Assumptions. Except with the assumptions made by the specific RBC and BA protocol used, the following reduction assumes that $n > 2f$. However, as proven in [DLS88], BA in partially synchronous setting with Byzantine faults requires $n > 3f$. So we are going to use the assumption $n > 3f$ rather than honest majority. The reduction works in permissioned model, in which the participants are determined at the beginning of execution, with both static and dynamic corruption with $n - f$ ever-honest nodes.

High level idea. Using BA, which is assured to terminate, to determine whether the output of the corresponding RBC instance should be (waited for and) collected into the final log or not.

The reduction is given in Figure 4. In this reduction, each node P_i participates in n RBC protocols and n BA protocols in each epoch ℓ, and maintains three local sets:

- $\mathsf{finished}_i^\ell$: contains all indices of RBC protocols that P_i has already output in.
- $\mathsf{collect}_i^\ell$: contains all indices of RBC protocols, the output of which need to be collected into the batch_i^ℓ.
- batch_i^ℓ: contains all transactions that will be finally included in the log.

Proof In the proofs in this section, we only consider the execution paths that all of the properties of RBC and BA protocols hold. The cases where some of the properties do not hold only happens with negligible probability, we can just take union bound and the probability is still negligible.

Here we give the proof for consistency and liveness properties of this protocol in static corruption setting.

Theorem 4.1 (Common Prefix)**.** *Except with negligible probability, for any two honest nodes P_i and P_j and any epoch ℓ, $\mathsf{log}_i^\ell = \mathsf{log}_j^\ell$.*

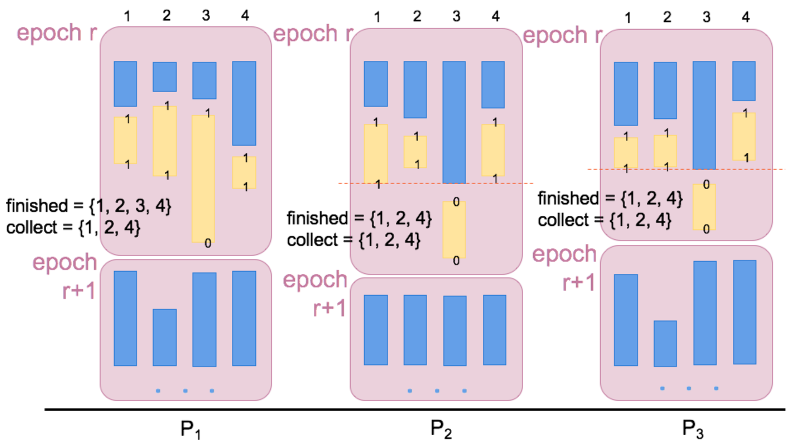

Figure 3: Example of the reduction for four nodes $P1 - P4$ of which node $P4$ is corrupt (and hence, its state is not shown). The blue bars indicate an ongoing instance of RBC, the yellow bars indicate theinstance of BA and the pink boxes indicate the ongoing epoch.

Proof. Here we show that, there is no transaction tx which is included in $\mathsf{log}_{i_0}^{\ell}$ and not included in $\mathsf{log}_{i_1}^{\ell}$ for and two honest nodes P_{i_0} and P_{i_1}.

For the sake of contradiction, assume that there is a transaction tx included in $\mathsf{log}_{i_0}^{\ell}$ and not included in $\mathsf{log}_{i_1}^{\ell}$. Then there must be a tuple $(\mathsf{tx}, j, \ell') \in \mathsf{batch}_{i_0}^{\ell''}$ for some $\ell'' \leq \ell$, which means this tuple must be included in $\mathsf{output}_{i_0,j'}^{\ell''}$ for some $j' \in \mathsf{collect}_{i_0}^{\ell''}$. And $(\mathsf{tx}, j, \ell') \notin \mathsf{batch}_{i_1}^{l''}$.

There are following two cases:

- **Case I:** $(j' \notin \mathsf{collect}_{i_1}^{\ell''})$**:** This implies that $b_{i_0,j'}^{\ell''} = 1$ and $b_{i_1,j'}^{\ell''} = 0$. However, according to the definition of binary agreement protocol, for the instance $\mathsf{BA}_{j'}^{\ell''}$, honest nodes P_{i_0} and P_{i_1} should output the same bit value. So there is contradiction.
- **Case II** $(j' \in \mathsf{collect}_{i_1}^{l''}$ **and** $(\mathsf{tx}, j, l') \notin \mathsf{output}_{i_1,j'}^{\ell''})$**:** According to the definiton of reliable broadcast protocol, for the instance $\mathsf{RBC}_{j'}^{\ell''}$, if P_{i_0} outputs $output_{i_0,j'}^{\ell''}$, and P_{i_1} outputs $output_{i_1,j'}^{\ell''}$, then $output_{i_0,j'}^{\ell''} = output_{i_1,j'}^{\ell''}$. So there is contradiction.

□

Theorem 4.2 (Future Self-Consistency)**.** *Except with negligible probability, for any honest node P_i and any two epoch $\ell_1 < \ell_2$, $\mathsf{log}_i^{\ell_1}$ is the prefix of $\mathsf{log}_j^{\ell_2}$.*

Proof. This theorem trivially follows the protocol description because there is no step in the protocol execution that modifies the existing log. □

Lemma 4.3 (Epoch Synchronization)**.** *If an honest node is in epoch ℓ, then no honest node is in epoch $\ell + 2$ or higher.*

Proof. For the sake of contradiction, assume there is an honest node P_{i_0} in epoch $\ell + 2$ when there is another honest node P_{i_1} in epoch ℓ for some ℓ. This assumption implies that P_{i_0} has output in all $\mathsf{BA}_{\ell+1,j}, j \in \{1, \ldots, n\}$ instances, otherwise it won't enter epoch $\ell + 2$. At the same time, P_{i_1} has not input in all $\mathsf{BA}_{\ell+1,j}, j \in \{1, \ldots, n\}$ instances because it is still in epoch ℓ. However, according to the definition of binary agreement protocol, P_{i_0} in $\mathsf{BA}_{\ell+1,j}, j \in \{1, \ldots, n\}$ won't output anything if P_{i_1} has not participated in yet. So there is contradiction. □

Lemma 4.4 (Epoch Progress)**.** *Except with negligible probability, if an honest node P_i changes its current epoch to ℓ in round T, then P_i will output log_i^{ℓ} and change its epoch number to $\ell + 1$ before round $T + \max\{T_{\mathrm{RBC}}, T_{\mathrm{BA}}\} + T_{\mathrm{RBC}} + T_{\mathrm{BA}}$.*

The protocol starts with $\ell = 0$ and all $v_{0,i} = \varnothing$. For epoch ℓ, the node P_i in SMR runs as following.

- At the beginning of epoch ℓ, $\mathsf{batch}_i^\ell \leftarrow \varnothing, \mathsf{finished}_i^\ell \leftarrow \varnothing, \mathsf{collect}_i^\ell \leftarrow \varnothing, v_{\ell+1,i} \leftarrow \varnothing$. and participate in $\mathsf{RBC}_{\ell,i}$ as sender with input value $v_{\ell,i} \setminus \mathsf{log}_i^{\ell-1}$.
- On receiving input transaction tx from the environment, set $v_{\ell+1,i} \leftarrow v_{\ell+1,i} \cup \{\mathsf{tx}\}$ and gossip it to other nodes.
- On receiving tx from other nodes, set $v_{\ell+1,i} \leftarrow v_{\ell+1,i} \cup \{\mathsf{tx}\}$.
- On outputting value $\mathsf{output}_{i,j}^\ell$ in $\mathsf{RBC}_{\ell,j}$ for $j \in \{1, \dots, n\}$, participate in $\mathsf{BA}_{\ell,j}$ with input bit $b_{i,j}^\ell = 1$, and set $\mathsf{finished}_i^\ell \leftarrow \mathsf{finished}_i^\ell \cup \{j\}$.
- On outputting 1 in $\mathsf{BA}_{\ell,j}$ for $j \in \{1, \dots, n\}$, update $\mathsf{collect}_i^\ell \leftarrow \mathsf{collect}_i^\ell \cup \{j\}$.
- When $|\mathsf{collect}_i^\ell| \geq n - f$, participate in all instances $\mathsf{BA}_{\ell,j}$ for $j \in \{1, \dots, n\} \setminus \mathsf{finished}_i^\ell$ with input bit $b_{i,j}^\ell = 0$.
- After outputting in all $\mathsf{BA}_{\ell,j}$ for $j \in \{1, \dots, n\}$:
  1. Wait for all $\mathsf{RBC}_{\ell,j}$ for $j \in \mathsf{collect}_i^\ell$ to terminate.
  2. $\mathsf{batch}_\ell \leftarrow \cup_{j \in \mathsf{collect}_i^\ell} \mathsf{output}_{i,j}^\ell$
  3. $\mathsf{log}_i^\ell \leftarrow \mathsf{log}_i^{\ell-1} || \ \mathsf{batch}_i^\ell$
  4. Enter epoch $\ell + 1$.

Figure 4: Reduction from SMR to BA and RBC

Proof. As described in the protocol, P_i will change its epoch number to $\ell + 1$ when all of the $\mathsf{BA}_{\ell,j}$ instances terminates and all $\mathsf{RBC}_{\ell,j}$ for $j \in \mathsf{collect}_i^\ell$ terminates. In the worst case, the corrupt nodes does not send anything, and the waiting time is maximized.

As P_i is already in epoch ℓ, P_i should have collected outputs of all $\mathsf{BA}_{\ell-1,j}, j \in \{1, \dots, n\}$, which implies that every nodes has already input to all $\mathsf{BA}_{\ell-1,j}$ instances. According to the termination property of binary agreement protocol, all nodes will output in all $\mathsf{BA}_{\ell-1,j}$ instances before round $T + T_{\text{BA}}$.

Let's consider the "slowest" node $P_{i'}$in epoch $\ell - 1$, which is the last one enters epoch ℓ. For the worst case, it wait for T_{BA} rounds to collect outputs of all $\mathsf{BA}_{\ell-1,j}$ instances, and it may still need to wait for some $\mathsf{RBC}_{\ell-1,j'}$ after $\mathsf{BA}_{\ell-1,j'}$ outputs 1. So the maximum time it takes for $P_{i'}$ to enter epoch ℓ is $\max\{T_{\text{BA}}, T_{\text{RBC}}\}$. Then $P_{i'}$ will input to $\mathsf{RBC}_{\ell,i'}$. It takes at most $T_{\text{RBC}} + T_{\text{BA}}$ rounds for P_i to finish instance $\mathsf{RBC}_{\ell,i'}$ and $\mathsf{BA}_{\ell,i'}$. P_i will output in all other instances before this time point. So P_i will enter epoch $\ell + 1$ before round $T + \max\{T_{\text{RBC}}, T_{\text{BA}}\} + T_{\text{RBC}} + T_{\text{BA}}$.

□

As described in the protocol, if a transaction tx is input to some honest node P_i in round T, P_i will broadcast tx to all other nodes by gossiping network. So at round $T + \Delta$, tx will be known by all honest nodes, and is ensured to be included in the $\mathsf{batch}_j^?$ for the next epoch. Therefore, Lemma 4.4 implies the liveness property of this state machine replication protocol.

Theorem 4.5 (Liveness). *Except with negligible probability, if P_i that is honest in round r receives some transaction* tx *in round r, then for all P_j which is honest in round $t \geq r + \Delta + 2(\max\{T_{\text{RBC}}, T_{\text{BA}}\} + T_{\text{RBC}} + T_{\text{BA}})$ and its current epoch number ℓ after round,* $\mathsf{tx} \in \mathsf{log}_i^\ell$.

4.2 From Binary Agreement to State Machine Replication

Here we introduce a trivial way to implement binary agreement with SMR in hand. This reduction works in permissioned model with PKI in the presence of dynamic corruption, assuming $n > 3f$, and $n - f$ nodes are ever-honest.

All nodes in BA starts an instance of SMR at beginning. Denote the output of P_i in SMR in (local) round ℓ with log_i^ℓ. Let $\mathsf{batch}_i^\ell := \mathsf{log}_i^\ell - \mathsf{log}_i^{\ell-1}$. The node P_i in BA runs as following.

- On receiving input bit b_i from environment, participates in the SMR with input set $\{\mathsf{vote}(i, b_i)\}$.
- For index ℓ from 1, on outputting the batch_i^ℓ, collects all *fresh* votes $\mathsf{vote}(j, b_j)$ into pool_i. For convenience, denote pool_i after collecting fresh votes from batch_i^ℓ with pool_i^ℓ.

  A vote $\mathsf{vote}(j, b_j)$ is *fresh* if pool_i does not contain any vote $\mathsf{vote}(j, \cdot)$.
- If for some l, $|\mathsf{pool}_i^{\ell-1}| < n - f$ and $|\mathsf{pool}_i^\ell| \geq n - f$, then outputs the bit with more votes in pool_i^ℓ.

  If there is a tie, output a pre-determined bit, like 0.

Figure 5: Reduction from BA to SMR

Proof In the proofs in this section, we only consider the execution paths that all of the properties of SMR protocols hold. The cases where some of the properties do not hold only happens with negligible probability, we can just take union bound and the probability is still negligible.

Here we give the proof for the three properties of this protocol in static corruption setting.

Theorem 4.6 (Agreement)**.** *Except with negligible probability, if any honest node outputs the bit b, then every honest node outputs b.*

Proof. For the sake of contradiction, assume that there are two honest nodes, P_{i_0} outputting 0 and P_{i_1} outputting 1. Then for the epoch l_0 in which P_{i_0} makes decision, the majority of votes in $\mathsf{pool}_{i_0}^{\ell_0}$ are for bit 0, and for the epoch l_1 in which P_{i_1} makes decision, the majority of votes in $\mathsf{pool}_{i_1}^{\ell_1}$ are for bit 1.

There are two cases:

- $\ell_0 \neq \ell_1$. This implies that $|\mathsf{pool}_{i_0}^{\ell_0}| \neq |\mathsf{pool}_{i_1}^{\ell_1}|$ or $|\mathsf{pool}_{i_0}^{\ell_0-1}| \neq |\mathsf{pool}_{i_1}^{\ell_1-1}|$. However, according to the common prefix property of SMR protocol, $|\mathsf{pool}_{i_0}^{\ell_0}| = |\mathsf{pool}_{i_1}^{\ell_1}|$ and $|\mathsf{pool}_{i_0}^{\ell_0-1}| = |\mathsf{pool}_{i_1}^{\ell_1-1}|$. Here is the contradiction.
- $\ell_0 = \ell_1 = \ell$, but numbers of votes for 0 are for 1 are different in $\mathsf{pool}_{i_0}^\ell$ and $\mathsf{pool}_{i_1}^\ell$. However, according to the common prefix property of SMR protocol, For the two honest nodes P_{i_0} and P_{i_1}, $\mathsf{pool}_{i_0}^\ell = \mathsf{pool}_{i_1}^\ell$ for any ℓ. Then there is a contradiction.

□

Theorem 4.7 (Termination)**.** *Except with negligible probability, If all honest nodes receive input in round T, then every honest node outputs a bit before round $T + T_{\text{conf}}$.*

Proof. This is trivially guarenteed by the liveness property of SMR protocol. As each honest node will only input one vote to the SMR protocol, and at most one vote from each node will be accepted by honest node, according to the liveness property of SMR, all of the inputs of honest nodes will be included in the log before round $T + T_{\text{conf}}$. Therefore, all of honest nodes will have output before round $T + T_{\text{conf}}$. □

Theorem 4.8 (Validity)**.** *Except with negligible probability, If any honest node outputs b, then at least one honest node receives b as input.*

Proof. Assume for the sake of contradiction that some honest node P_i outputs b which is not the input of any honest node. Then we know that in P_i's final pool_b, the majority of votes vote for the bit b. As required in the protocol description, $|\mathsf{pool}_b^\ell| \geq n - f$, which means the number of votes for bit b is larger than or equal to $(n-f)/2$, which is greater than f. Then the pool_b^ℓ contains at least one vote for b from an honest node. Then there is contradiction. ☐

4.3 From Reliable Broadcast to Binary Agreement

In this reduction, we assume the existance of authenticated channel, and that the length of input value v is known a priori. If $|v|$ is not pre-determined, the protocol can first have all node agree on $|v|$, then agree on the input value v. The reduction is presented in Figure 6. It also works in both static and dynamic corruption settings with $n - f$ ever-honest nodes.

Denote the input value from environment to the sender with v.

- For $k = 1$ to $|v|$, the sender diffuse v_k, the k-th bit of v, to every node, and all nodes run a binary agreement instance BA_k tagged with the index k.
- For index k from 1 to $|v|$, on outputting the bit b_i^k, P_i concatenate b_i^k to the output value, i.e. $\mathsf{output}_i = \mathsf{output}_i || b_i^k$.

Figure 6: Reduction RBC to BA

In this reduction, consistency is guaranteed by the agreement property of binary agreement.

As defined in the property of RBC protocols, this reduction does not guarantee termination when the sender is corrupt. For example, if the corrupt sender skips some index k, then the execution of the protocol can be suspended. If the sender is honest, then the termination is guaranteed by the termination property, and validity is guaranteed by the validity property of binary agreement.

5 Conclusion and Future Work

In this report, we gave two-way reduction algorithms between state machine replication and binary agreement in the partially synchronous setting. The existence of reduction implies the equivalence between these two consensus protocols when we assume the existence of PKI.

In the next step, we want to figure out if there exists reduction from BA to SMR with assumption of authenticated channel rather than PKI. We also plan to find reduction algorithms that work in permissionless model.

References

[AM+17] Ittai Abraham, Dahlia Malkhi, et al. "The Blockchain Consensus Layer and BFT". In: *Bulletin of EATCS* 3.123 (2017).

[Bra84] Gabriel Bracha. "An asynchronous [(n-1)/3]-resilient consensus protocol". In: *Proceedings of the third annual ACM symposium on Principles of distributed computing.* ACM. 1984, pp. 154–162.

[Bur06] Mike Burrows. "The Chubby lock service for loosely-coupled distributed systems". In: *Proceedings of the 7th symposium on Operating systems design and implementation.* USENIX Association. 2006, pp. 335–350.

[Cac+01] Christian Cachin et al. “Secure and efficient asynchronous broadcast protocols”. In: *Annual International Cryptology Conference*. Springer. 2001, pp. 524–541.

[CGR11] Christian Cachin, Rachid Guerraoui, and Luis Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[CL+99] Miguel Castro, Barbara Liskov, et al. “Practical Byzantine fault tolerance”. In: *OSDI*. Vol. 99. 1999, pp. 173–186.

[Cri+95] Flaviu Cristian et al. “Atomic broadcast: From simple message diffusion to Byzantine agreement”. In: *Information and Computation* 118.1 (1995), pp. 158–179.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. “Consensus in the presence of partial synchrony”. In: *Journal of the ACM (JACM)* 35.2 (1988), pp. 288–323.

[DS83] Danny Dolev and H. Raymond Strong. “Authenticated algorithms for Byzantine agreement”. In: *SIAM Journal on Computing* 12.4 (1983), pp. 656–666.

[Gar+16] Juan A Garay et al. “Bootstrapping the Blockchain-Directly.” In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 991.

[JRS11] Flavio P Junqueira, Benjamin C Reed, and Marco Serafini. “Zab: High-performance broadcast for primary-backup systems”. In: *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*. IEEE. 2011, pp. 245–256.

[KS05] Klaus Kursawe and Victor Shoup. “Optimistic Asynchronous Atomic Broadcast.” In: *ICALP*. Springer. 2005, pp. 204–215.

[Lam05] Leslie Lamport. *Generalized consensus and paxos*. Tech. rep. Technical Report MSR-TR-2005-33, Microsoft Research, 2005.

[Lam06] Leslie Lamport. “Fast paxos”. In: *Distributed Computing* 19.2 (2006), pp. 79–103.

[Lam84] Leslie Lamport. “Using time instead of timeout for fault-tolerant distributed systems.” In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 6.2 (1984), pp. 254–280.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine generals problem”. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (1982), pp. 382–401.

[Mil+16] Andrew Miller et al. “The honey badger of BFT protocols”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 31–42.

[Nak08] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008.

[PS17] Rafeal Pass and Elaine Shi. *Thunderella: blockchains with optimistic instant confirmation*. 2017.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. “Analysis of the blockchain protocol in asynchronous networks”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2017, pp. 643–673.

[RC05] HariGovind V Ramasamy and Christian Cachin. “Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast”. In: *OPODIS*. Springer. 2005, pp. 88–102.

[SB12] João Sousa and Alysson Bessani. “From Byzantine consensus to BFT state machine replication: A latency-optimal transformation”. In: *Dependable Computing Conference (EDCC), 2012 Ninth European*. IEEE. 2012, pp. 37–48.

[Sch90] Fred B Schneider. “The state machine approach: A tutorial”. In: *Fault-Tolerant Distributed Computing*. Springer, 1990, pp. 18–41.

[Tou84] Sam Toueg. “Randomized byzantine agreements”. In: *Proceedings of the third annual ACM symposium on Principles of distributed computing*. ACM. 1984, pp. 163–178.