Streamlining Classical Consensus

Kyle Butt[†], Derek Sorensen[‡]

Abstract. Consensus protocols based on rounds of voting naturally give rise to a directed acyclic graph (DAG), which we call the *message DAG*. We restructure two classical consensus protocols via the message DAG, consolidating message rounds with new messages, to achieve manifestly improved scalability. The second protocol lends itself to a scalable blockchain consensus protocol with strong safety and liveness guarantees. We also present a technique to generalize other message-based, classical consensus protocols.

KEY WORDS

1. Consensus protocol 2. Directed acyclic graph 3. Pre-Nakamoto consensus 4. Classical consensus 5. Byzantine fault tolerant

1. Introduction

The problem of achieving consensus over an asynchronous network between mutually distrustful parties has been a subject of research for several decades. Since Nakamoto's famous paper[5] that gave rise to Bitcoin, a whole class of consensus algorithms has emerged, often making use of a miner or a leader election of some kind. These include proof of work (PoW), proof of stake (PoS), and proof of elapsed time (PoET) protocols, among others.

Many of these new algorithms don't come with practical proofs of safety and liveness. Indeed, Bitcoin and Ethereum never guarantee that a block won't be later rewritten by introduction of a longer, conflicting chain because one can never know a block is final. Any guarantees are probabilistic at best, though the Bitcoin white paper gives no rigorous proofs[5]. Due to the high risk inherent in transacting with large amounts of money, if cryptocurrencies are to play an important role in financial and business settings it is vital that we develop scalable consensus protocols with mathematically rigorous safety and liveness proofs. While such proofs are not, generally, characteristic of post-Nakamoto consensus protocols, the literature on classical consensus is grounded in excellent, rigorous mathematics. Unfortunately, these classical algorithms tend to be unscalable due to sheer messages volume.

We present a technique to generalize classical consensus that tames this inherent scalability issue. Given such a protocol, it produces another that is manifestly more efficient, and equally safe and live. Lacking a coherent theory about leaderless message-passing protocols, it is difficult to give a characterization of algorithms for which this will work. En lieu of such a theory, we provide two examples for which this works and a general framework which one can apply to other consensus protocols based on message-passing rounds. We expect that readers will be able to adapt the techniques we present to classical algorithms relevant to them.

[†] K. Butt (kyle@pyrofex.net) is a Staff Software Engineer at Pyrofex Corporation.

[‡] D. Sorensen (derek@pyrofex.net) is a Research Mathematician at Pyrofex Corporation.

The outline of the paper is as follows: In §2 we introduce some preliminary terms and notation; in §3 we introduce a key concept, the *message DAG*; in §4 we show how the message DAG relates to classical consensus and give an overview of our generalization technique; in §5 we show a generalization of Byzantine Reliable Broadcast[2]; finally, in §6 we do the same but for Algorithm 2[4].

2. Preliminaries

2.1. Graph theoretic notions—A *directed acyclic graph* (DAG) is a directed graph with no cycles. A cycle is a path along graph edges whose source and target are the same vertex. A *leaf* is a graph vertex that is the source to some (possibly empty) set of edges but target to none.

Note that a finite nonempty DAG always has a nonempty set of leaves. To find a leaf, one can choose a vertex at random and follow any reverse path until termination. In the DAG consisting of one vertex (and no edges), that vertex is a leaf. In our case, we will only consider DAGs with a "genesis" vertex such that every node is connected to the genesis vertex via some path along vertices.

2.2. Induced subgraph—For a graph G with vertex set W and edge set F, the *induced subgraph* of the vertex set $W' \subseteq W$ is a graph G' with vertex set W' and edge set $F' \subseteq F$, where F' is the set of edges in F with both source and target in W'.

2.3. Partially ordered set—A *partially ordered set* (poset) is a set P and a binary relation $\leq$ such that (among other things) for any two elements p_1 and p_2, either $p_1 \leq p_2$, $p_2 \leq p_1$, or p_1 and p_2 are not related. In a *totally ordered set*, we eliminate the third possibility, requiring that every pair of elements be related. Such relations are called, respectively, a *partial order* and a *total order*.

2.4. Byzantine fault tolerant—A *Byzantine node* or *Byzantine validator* is a member of consensus that behaves arbitrarily. Consistent with the literature, we denote by f the number of Byzantine validators, and consider a network of size $N = 3f + 1$. A consensus protocol is *Byzantine fault tolerant* if it can achieve safety and liveness in the presence of $f < \frac{N}{3}$ Byzantine validators. A *Byzantine quorum* is a set of $2f + 1$ members of consensus.

3. The message DAG

Consider a message-passing protocol based on broadcasts to the whole network. We can define a total order on messages on the network, based on the time they were sent, as $m_1 \leq m_2$ iff m_1 was sent at exactly the same time or after m_2. In practice, on an asynchronous network, it is impossible to discover this total order. It is, however, possible to know that one message was sent after another if the sender of the first received the second before sending. Thus we can reasonably impose a partial order on messages, where some unknown monotonic map from the partial order to this total order completes the ordering. The *message DAG* graph-theoretically represents a partial ordering of messages on the network, based on the time they were sent, defined as $m_1 \leq m_2$ iff the sender of m_1 received m_2 before sending. By replacing $\leq$ with $\rightarrow$, a partial order yields a DAG (whose vertices are messages).

If each member of the network, which we call a *validator*, includes in any message they send the messages they saw before they sent it, validators can construct and keep a message DAG locally. If at any point all messages on the network get delivered, every nonfaulty validator will

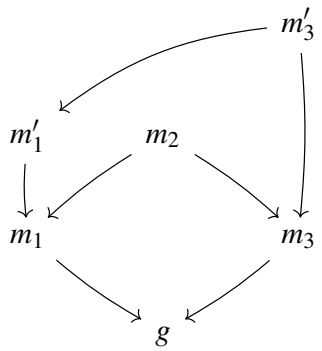

Fig. 1. A message DAG for a network of validators w_i, the senders of m_i or m'_i, with the genesis message g. When w_1 sent m'_1 it only saw its first message m_1 and g, and had received none from its peers; when w_2 sent m_2, it had seen m_1, m_3, and g; and when w_3 sent m'_3, it had seen m'_1 (thus also m_1) and its own previous message, m_3. Any validator sending a message with this message DAG stored locally would append their message to the DAG, pointing to m_2 and m'_3.

have exactly the same message DAG stored locally, up to leaves corresponding to messages sent by faulty validators to some subset of the network.

To build the DAG, each validator w begins with a genesis (or empty) vertex g which represents the time that the network started listening for messages; g is the maximal element in the partial order. When w sends a message m it updates its message DAG with a new vertex m and arrows that point to all the leaves of the DAG, indicating the partial order; it then includes this information about m's parents in m. When w receives a message m' from another participant w', it checks to see that its message DAG contains the parents of m', the messages w' received before sending m'. If it does not, w can either wait for more messages until it does have the parents, or ask w' for the parent messages.

In Figure 1, validators w_i send messages m_i, m'_i, etc. Because each validator points to the leaves of their message DAG when they send a new message, we can infer what w_1, w_2, and w_3 had already seen when they sent their messages. Note that m_2 and m'_3 are not related in the partial order; there is no way of knowing which was sent first, but we do know that both were sent after m_1 and m_3.

What we have described thus far is a slight modification to any message-based protocol that allows each validator to keep an accurate, local message DAG. We can, however, think of a message DAG from the view of an omniscient viewer without making any changes to the protocol. If such an omniscient viewer watches the validators pass messages, it can construct a message DAG (indeed it can do better and establish a total order). This DAG is a physical representation of how mathematicians reason about consensus protocols: we, omniscient viewers, see the actions and decisions of all validators and we reason about them based on the order in which the messages were sent and received. By keeping an up-to-date message DAG, a validator keeps a local copy of our abstract reasoning structure.

4. Streamlining consensus

With the message DAG in hand, we have an essential tool to rethink classical consensus protocols that rely on rounds of message-passing. Our thesis is that if validators keep and update a message DAG locally, we can vastly improve these protocol's throughput and scalability by reducing the prohibitively large message volume.

We will show this streamlining method on two classical consensus protocols. The first is Byzantine Reliable Broadcast[2], a Byzantine fault tolerant message-passing protocol; the second is Algorithm 2 due to Dwork, Lynch, and Stockmeyer[4], a classical protocol that achieves consensus over a value set V where each validator (member of consensus) begins with an initial value from V. As we will show, the latter is particularly well-suited for use on the blockchain. In principle the technique we present could be used on any consensus protocol based on message passing and rounds. It is not immediately clear to us that we can improve post-Nakamoto consensus protocols in the same way, due to randomized leader election and other characteristic features.

The general method for a message-passing protocol P, roughly speaking, goes as follows. The consensus protocol P comes to consensus on elements from some collection of objects $\mathbb{O}$. If it is running on a blockchain, $\mathbb{O}$ is the collection of possible blocks. If the protocol is purely for consensus on messages, $\mathbb{O}$ is the collection of possible messages. The network only communicates via some broadcast primitive. Any message required by P is either sent in the header of a broadcast, marked with its intended recipient (if any), or in the body of the broadcast. We call elements $O \in \mathbb{O}$ *body messages* and messages in P related to consensus *header messages*. As the terminology suggests, header messages go in the broadcast header and body messages go in the broadcast body.

Either the header or body could be empty, for example if the validator doesn't have any elements of $\mathbb{O}$ at hand to propose, or if a validator has no header messages queued. The header could also include multiple messages with distinct recipients. If each validator has an unending supply of elements of $\mathbb{O}$ (i.e., of transactions to put into blocks or of messages to send), we streamline P so that every message related to consensus is first and foremost a proposal of some $O \in \mathbb{O}$, with information in the header related to consensus on previously proposed elements of $\mathbb{O}$ in the network.

Each validator keeps a message DAG locally. Validators include the parents of a particular message in the broadcast. We omit this from the notation, thought it is essential so that validators can keep an accurate message DAG. When any validator receives a message, it (1) processes information from the header and body related to consensus, (2) follows P for each message received in both the header and body, and (3) queues any header messages produced by P. When it is ready to broadcast, it bundles its queue into the header of a new message, with some fresh $O \in \mathbb{O}$ in the body (if there is one) and broadcasts. The protocol always initiates over an empty *genesis element* $g \in \mathbb{O}$, which we assume to always exist in $\mathbb{O}$. This element g is the initial vertex in every validator's message DAG and we begin consensus by following P to come to consensus on g.

Once P terminates on any given $O \in \mathbb{O}$, validators no longer need to include header messages related to O in their broadcasts. Thus if P can be proved to terminate for any O, data related to O will be in only a finite number of messages.

Finally, the proofs of safety and liveness all have a similar structure: We look at a particular

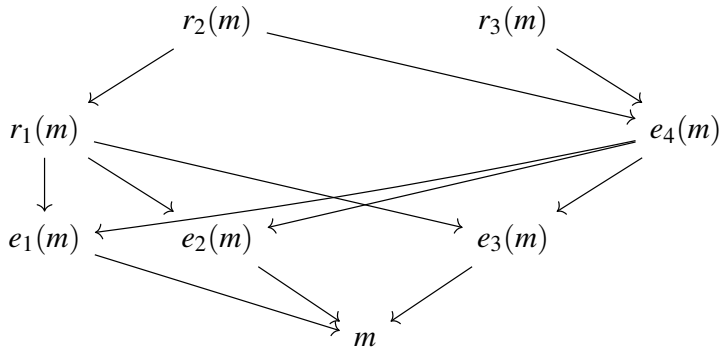

Fig. 2. The message DAG for consensus on the message m, where $e_i(m)$ and $r_i(m)$ are, respectively, echo and ready messages for m from validator w_i. This message DAG constitutes sufficient proof for any of the four validators to safely *brb*-deliver the message m because they would have received a Byzantine quorum of ready messages, $r_1(m)$, $r_2(m)$, and $r_3(m)$.

element O over which we try for consensus. If we look at each validator's message DAG and isolate the body or header messages that relate explicitly to consensus on O, we have a single, isolated case of the original protocol P. Thus, under the same set of assumptions, this modified protocol achieves the same proven properties of the original P. Since proofs are all done by reduction, our modified protocols can be considered direct generalizations of their original counterparts.

We call such a modified version of any protocol "MDM [protocol name]," short for "message DAG modified [protocol name]."

5. Byzantine Reliable Broadcast

In the style of §4, we first illustrate on a well known Byzantine fault tolerant message-passing protocol called Byzantine Reliable Broadcast (BRB)[2]. BRB is a protocol for secure message delivery with the property that if one honest validator *brb*-delivers (*i.e.* accepts) a given message, eventually every other validator will eventually do the same.

The basic algorithm goes as follows: The message-sender broadcasts a message m to the rest of the network via some broadcast primitive *brb*-broadcast. Each validator w_i broadcasts an "echo" message $e_i(m)$ to notify the rest of the network that they received m. Once w_i has received $e_j(m)$ from a Byzantine quorum of validators, it broadcasts a "ready" message, $r_i(m)$ to notify the others that it is willing to *brb*-deliver m if a Byzantine quorum is also willing. Once w_i receives $r_j(m)$ from a Byzantine quorum, it *brb*-delivers m. All-in-all, including the original message broadcast, there are three rounds of message passing before a message can be considered safe to *brb*-deliver. Note that duplicate messages are not a problem, and via simple cryptographic methods we can make sure each validator can tell who sent which messages.

A network running BRB could pass several messages at once, originating from any validator. However, BRB requires at least $3 \times (2f+1)$ consensus-specific messages for a network of size $N = 3f+1$ just to come to consensus on one m. Clearly, message cost becomes prohibitive as the network grows and as the number of messages validators have to send increases.

Figure 2 represents an omniscient viewer's message DAG for a single instance of BRB on

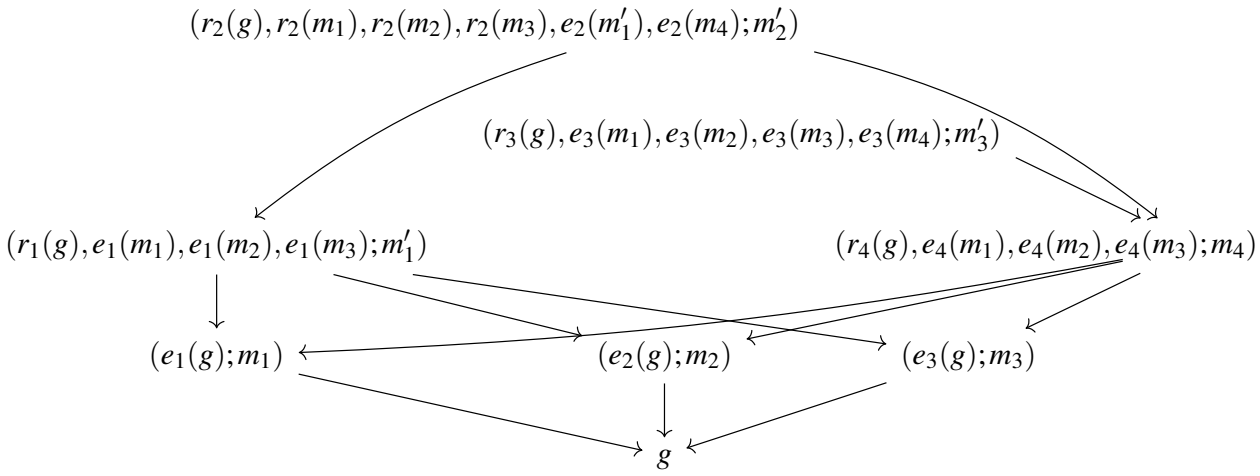

Fig. 3. The message DAG for MDM BRB. Each message a validator w_i sends contains in the body a new message m_i, m'_i, m''_i, *etc.* and in the head messages of the form $e_i(-)$ or $r_i(-)$ relating to consensus on other messages m_j, m'_j, m''_j, *etc.* The validator with this message DAG proceeds by creating a new message that points to messages m'_2 and m'_3, the leaves, and which omits any mention of the genesis message g owing to the fact that there is a Byzantine quorum of $r_j(g)$ messages on the DAG.

a network of size 4 with validators w_i, $1 \leq i \leq 4$. This diagram shows that if any of the four validators receive all the messages that have been sent, it would be sufficient to safely *brb*-deliver m because there is a Byzantine quorum of ready votes. At this point, assuming network conditions specified for BRB, every other honest validator will eventually *brb*-deliver m, since BRB has the *totality* property[2].

5.1. MDM BRB—Since all consensus-specific messages in BRB are broadcasts, it is natural to implement the message DAG as described in §3. Our modifications to BRB to make MDM BRB are slight and somewhat cosmetic, but still significant. They are as follows:

Each validator w_i begins with an empty genesis message, g, and begins consensus on g. To do so, w_i produces a message with $e_i(g)$ in the header and a new message m_i in the body. Following the convention from §4 we write this message $(e_i(g); m_i)$. When any other validator w_j receives $(e_i(g); m_i)$, it produces a message with $e_j(m)$ and, if it has already received a Byzantine quorum of echoes for g, $r_j(g)$ in the header. Otherwise it includes $e_j(g)$. In the body it includes a new message m_j. Using our notation, w_j sends $(\ldots, r_j(g), \ldots, e_j(m), \ldots; m')$. The ellipses indicate that any information related to consensus for any other messages received would also go into the header, as it would do each time it produces a message. Note that an echo or ready consensus message is only ever included *once* per body message. Once a validator has seen sufficient votes to *brb*-deliver a message m, it no longer needs to include information on m in any other message headers. Figure 3 gives a sample message DAG which has formed consensus on g. At the point of consensus for g, the network is just two steps from consensus on four other messages m_1, m_2, m_3, and m_4.

If the network need only form consensus on a single body message m, each validator can send

broadcasts with an empty body, *i.e.* either of the form $(e_i(m);\emptyset)$ or $(r_i(m);\emptyset)$, depending on the stage of consensus. This reduces to a single instance of BRB on m where all the echo and ready header messages appear in the header of empty messages. Also, if body messages do not enter the network quickly enough, validators can send broadcasts with empty bodies when they don't have any new body messages to add to consensus. This is, in the worst case, as efficient as using BRB in the traditional sense for each new message. In the optimal case that the network needs to come to consensus on a stream of body messages, originating either from specific sources or from any validator, it can simultaneously come to consensus on several body messages at once, significantly reducing the number of messages required per message.

Under some mild network assumptions, BRB is live only in the sense that an honest validator, proposing a message, will eventually see its message confirmed. However, it can only confirm messages and will never explicitly reject one (the network can only say "yes," and never "no"), so a Byzantine node could spam the network with unconfirmable messages. MDM BRB is at least as live as BRB, since it includes precisely the same number of header messages per proposed body message as BRB does; it just compresses many into a single header.

However, aside from a substantially more efficient protocol, the message DAG offers a way to keep track of these kinds of observable Byzantine faults. If, for example, a Byzantine validator w_j sends broadcasts $(\ldots;m)$ and $(\ldots;m')$ to distinct members of the network which cite *the same set of parents* such that $m \neq m'$, the network will eventually be able to identify this deviant behavior in their message DAG. The network can make some optimization choices against Byzantine behavior by, *e.g.* ignoring Byzantine validators for a time or excluding them from the network. In this way, MDM BRB is at least as live than BRB.

Result 1. *MDM BRB has the same safety and liveness properties as BRB under the same set of assumptions.*

Proof. Consider MDM BRB for a single message m, looking only at the messages and portions of message headers that pertain to consensus on m. The resulting message DAG is the original message DAG we constructed for BRB, meaning that for any individual message m, MDM BRB executes precisely the same consensus protocol for m that BRB does—the difference is that it does everything simultaneously. □

Porting results in this way is a general property of the MDM construction. In our case, it show that MDM BRB achieves consistency, totality, validity, and integrity [2, p. 119], as does BRB.

6. Algorithm 2^4

A construction analogous to the one we've just shown can be done for more general and useful consensus protocols. We show this on a classical protocol that can come to consensus on a multi-valued set, Algorithm 2 due to Dwork, Lynch, and Stockmeyer[4]. Algorithm 2 uses voting rounds and leader selection which can be incorporated into the message DAG much like the rounds of consensus in BRB. Though Algorithm 2 is not used in general for the blockchain, we show a construction compatible with the blockchain. In this case, block proposals combine with messages relating to consensus rounds. Thus, instead of building a blockchain we build a block*DAG*.

The reason we do this is that a blockchain provides a stream of value sets V over which the

Algorithm 1 MDM BRB

1: $objectsOnHand$ ▷ Elements $O \in \mathbb{O}$ accessible to w_i
2: $pending, queue \leftarrow \emptyset$
3: MDM-BRB-receive(- ; *genesis*) ▷ Initiate consensus
4: **procedure** MDM-BRB-RECEIVE(header ; m)
5: **if** $parents(m) \subset DAG$ **then** $queue \leftarrow queue \cup \{e_i(m)\}$, $receivedEchoes(m) \leftarrow 1$
6: **else** $pending \leftarrow pending \cup \{m\}$ ▷ Wait for $parents(m)$
7: **end if**
8: **for** $e_j(m')$ in header **do**
9: **if** $parents(e_j(m')) \subset DAG$ **then**
10: **if** $receivedEchoes(m') \geq 2f+1$ **then** $queue \leftarrow queue \cup \{r_i(m')\}$
11: **else**
12: **if** $receivedEchoFrom(m'; w_j)$ is false **then** $receivedEchoes(m') \leftarrow receivedEchoes(m') + 1$
13: **end if**
14: **end if**
15: **else** $pending \leftarrow pending \cup \{m\}$ ▷ Wait for $parents(m')$
16: **end if**
17: **end for**
18: **for** $r_j(m')$ in header **do**
19: **if** $parents(e_j(m')) \subset DAG$ **then** ▷ $receivedEchoes(m') \geq 2f+1$
20: **if** $receivedReady(m') \geq 2f+1$ **then** *brb*-deliver m'
21: **else**
22: **if** $receivedReadyFrom(m'; w_j)$ is false **then** $receivedReady(m') \leftarrow receivedReady(m') + 1$
23: **end if**
24: **end if**
25: **else** $pending \leftarrow pending \cup \{m\}$ ▷ Wait for $parents(m')$
26: **end if**
27: **end for**
28: **end procedure**
29: **procedure** MDM-BRB-BROADCAST($queue \neq \emptyset$)
30: **if** $objectsOnHand \neq \emptyset$ **then** choose $O \in \mathbb{O}$, *brb*-broadcast $(queue; O)$
31: **else** *brb*-broadcast $(queue; -)$
32: **end if**
33: **end procedure**

network needs to form consensus; a value set contains a block b and all blocks that it conflicts with (*e.g.* other blocks which constitute double spends, *etc.*). However, this can be used in a general setting where value sets come in constant supply. As before, validators can send messages with empty bodies (block proposals), with consensus-related information in the headers if they don't have content for some V accessible.

6.1. Algorithm 2—For those unfamiliar with Algorithm 2, we define the basic notions, summarizing the original exposition [4, p. 298].

- *PROPER sets*: Possible results of consensus. Initially, each validator's PROPER set contains just its own initial value. It adds to it a value $v' \neq v$ if it receives claims from $t+1$ validators during the protocol that v is in each of their PROPER sets. If V is well-defined, a validator adds all of V to its PROPER set after receiving $2t+1$ initial values from different validators, among which there are not $t+1$ with the same value.
- *Rounds and Phases*: A phase consists of three *trying* rounds and one *lock-release* round. We say a phase h belongs to validator w_i if $h \equiv i \mod N$, where N is the total number of validators. In the language of modern consensus protocols, we would call w_i the leader during round h.
- *Locks*: A validator w can lock a value v at certain points of the protocol, each time associated to a phase number and thus to a phase owner. Intuitively, a validator w locks on v during phase i if it thinks that the phase owner, w_i, will decide on v during phase i; validator w only releases its lock if it learns this supposition was false. Initially, w locks no value. A value v is *acceptable* to w if w has locked on either nothing or v.

During round 1 of phase i, each validator sends a list of all its acceptable values that are also in its proper set to validator w_i. Validator w_i tries to find a vlaue to *propose*, for which it needs value sets from $N-f$ validators (possibly including itself) that have a nonempty intersection. If such an intersection exists, w_i chooses a value v from the intersection to propose, broadcasting a lock message of the form *(lock value, round number, phase number)*.

If any validator receives such a lock message at round 2 of phase i from the phase owner, it locks v, associating the phase number with the lock, and sends an acknowledgment to w_i in round 3 of phase i, releasing any earlier lock. Validator w_i decides on v if it receives acknowledgments from at least $f+1$ processors during round 3 of phase i. It continues to participate in consensus, holding v as its only acceptable value.

Finally, at round 4 of phase i, the lock-release round, each validator broadcasts a message (v,i) if it locked on v during phase i. If any validator has a lock on v for phase i and receives a message (v',j) with $v' \neq v$ and $j \geq i$, it releases its lock on v.

6.2. Running Algorithm 2 natively on the message DAG—Algorithm 2 has proofs of safety and liveness, referenced in the original paper as *consistency*, *strong unanimity*, and *termination* [4, p. 297]. MDM Algorithm 2 inherits these proofs of safety and liveness in the same way that MDM BRB inherits the properties of BRB. As we streamline Algorithm 2, value sets and PROPER sets defined in §6.1 function as follows:

- *Value sets*: We want to optimize Algorithm 2 for the case that there is a constant stream of value sets V over which the network needs to find consensus. The obvious example for this is the blockchain, but in principle this could apply to any application for which there is a constant stream of value sets. In the case of a blockchain, a particular instance

Algorithm 2 MDM Algorithm 2

1: $blocksOnHand$ ▷ Elements $b \in \mathbb{O}$ (blocks) accessible to w_i
2: $queue \leftarrow \emptyset$
3: MDM-Algorithm-2-receive(- ; *genesis*) ▷ Initiate consensus
4: **procedure** MDM-ALGORITHM-2-RECEIVE(*header* ; b)
5: **if** b conflicts with block b' in PROPER set **then**
6: **if** received b from $\geq t+1$ distinct validators **then** add b to PROPER set
7: **end if**
8: **else** add b to PROPER set; $queue \leftarrow queue \cup \{(b, \text{round} = 1, \text{phase} = 1)\}$
9: **end if**
10: **for** ($acceptableValues$, round $= 1$, phase $= j$) in *header* **do**
11: **if** $j \equiv i \mod N$ **then** ▷ w_i owns phase j
12: **if** b in $acceptableValues$ from $\geq N - f$ distinct validators **then**
$queue \leftarrow queue \cup \{(lock(b)$, round $= 2$, phase $= j)\}$
13: **end if**
14: **end if**
15: **end for**
16: **for** ($lock(b)$, round $= 2$, phase $= j$) in *header* **do**
17: **if** $j \not\equiv i \mod N$ **then** ▷ w_i does *not* own phase j
18: **if** already locked on some $b' \neq b$ during phase j', and b', b conflict **then**
19: **if** $j \geq j'$ **then** release lock on b'
20: **end if**
21: **else** lock b for phase j;
$queue \leftarrow queue \cup \{(ack(lock(b))$, round $= 3$, phase $= j)\}$
22: **end if**
23: **end if**
24: **end for**
25: **for** ($ack(lock(b))$, round $= 3$, phase $= j$) in *header* **do**
26: **if** $j \equiv i \mod N$ **then** ▷ w_i owns phase j
27: **if** received $ack(lock(b))$ from $\geq f+1$ distinct validators for phase j **then**
decide b
28: **end if**
29: **end if**
30: **end for**
31: **end procedure**
32: **procedure** MDM-ALGORITHM-2-BROADCAST(*queue* , $blocksOnHand$)
33: **for** new round **do** ▷ the network proceeds in fixed-interval rounds
34: **if** $blocksOnHand \neq \emptyset$ **then** choose $b \in blocksOnHand$; broadcast (*queue* ; b)
35: **else**
36: **if** $queue \neq \emptyset$ **then** broadcast (*queue* ; -)
37: **end if**
38: **end if**
39: **end for**
40: **end procedure**

of consensus deals with a block b; V contains b and any blocks proposed by the network conflicting with b. Since no honest validator will propose a block conflicting with that which it has already seen, any value set V will have a maximum size of $(N-f)+f(N-f)$. As before, we can run consensus on several value sets concurrently.

- *PROPER sets*: Function as defined previously. A validator's initial value for consensus pertaining to V is the first element of V it encounters, either by proposing or by receiving a message. If it encounters more than one value of V simultaneously, it can choose one arbitrarily.

MDM Algorithm 2 initiates (in a similar way to MDM BRB) by initiating consensus on a genesis singleton value set $V_{gen} = \{\star\}$. In vanilla Algorithm 2, each validator w_i would send $\{\star\}$ to w_1 to start consensus. Instead of sending a message exclusively to w_1, a validator w broadcasts its message DAG indicating that it is seen the genesis block, including its set of acceptable values for w_1. Note that by modifying Algorithm 2 to broadcast all messages, we introduce redundancy but no core changes. In particular, all the results on safety and liveness still hold. Furthermore, doing so allows each validator to keep an accurate message DAG locally.

Again, we make every message a block proposal, with any consensus related messages in the header. Message-passing happens in rounds, like in the original algorithm, where each validator can send at most one message per round. Rounds proceed as normal, except they process several blocks simultaneously.

When any particular validator w receives a new block proposal b, it begins consensus for b on phase 1. In its next block proposal, it includes in the header a message for w_1, the owner of round 1, for consensus. Producing one block per round, it continues to include the data in the header corresponding to b as it cycles through the round leaders. Once consensus terminates on b, it no longer includes information about b in its header.

If w_i receives two conflicting blocks it sends as its initial value in consensus the first block it saw. If it saw more than one simultaneously, it chooses one arbitrarily—though one could expedite consensus by making a canonical choice, *e.g.* the block with the lowest hash. Since no honest validator will propose a block conflicting with that which it has already seen, any value set V will have a maximum size of $(N-f)+f(N-f)$. The message DAG again provides us with a way to track observable faults, as the network can tell which nodes propose different initial values to different nodes. This could improve efficiency and incentivize honest behavior.

Like with BRB, to show that MDM Algorithm 2 is safe and live, styled in the original paper as consistency, termination, and strong unanimity, we isolate one set of values V containing a block b and all blocks that conflict with b. Doing this shows us the message DAG corresponding to an individual instance of Algorithm 2 on a single set of values V. Thus under the same assumptions, we get the same results, and we conclude that MDM BRB is safe and live.

7. Conclusion

We present the message DAG, a tool to fundamentally restructure classical consensus protocols based on message-passing rounds. These do not scale in general due to a prohibitively large message volume. This is unfortunate, since they have proofs of safety and liveness, something generally not true of blockchain consensus protocols such as those of Bitcoin and Ethereum. By modifying these algorithms with the message DAG, we showed a way to streamline them to

work on consensus for many messages, blocks, or value sets at once. This modification makes it possible to achieve consensus by only ever broadcasting message, or block, proposals with some auxiliary information in the header.

We present MDM BRB, a streamlined Byzantine fault tolerant message-passing protocol which has the same safety and liveness guarantees as Byzantine Reliable Broadcast (BRB).[2] We also introduc MDM Algorithm 2, a streamlined Byzantine fault tolerant cosnensus protocol for consensus on a stream of value sets V which shares safety and liveness properties with Algorithm 2[4]. MDM Algorithm 2 is particularly suited for a blockDAG, which builds blocks in a way similar to a blockchain but with in a DAG as opposed to a chain.

Depending on the use case, MDM Algorithm 2 can be optimized. An example of such an optimization is our recent work, the consensus protocol *Casanova*[1]. Importantly, Casanova eliminates redundant rounds of consensus, optimizing for a transaction-based blockchain. We would like to see a coherent theory of consensus that characterizes algorithms to which this process can be applied.

8. Acknowledgment

We thank Michael Stay and Nash Foster, resp. the CTO and CEO of Pyrofex, for many enlightening conversations and insightful observations.

Notes and References

[1] Butt, K., Sorensen, D., Stay, M. *Casanova*. No publisher (2018) https://arxiv.org/abs/1812.02232

[2] Cachin, C., Guerraoui, R., Rodrigues, L. *Introduction to Reliable and Secure Distributed Programming*. New York: Springer (116-119) 2011

[3] Chen, J., Micali, S. *Algorand*. No publisher (2017) https://arxiv.org/pdf/1607.01341.pdf

[4] Dwork, C., Lynch, N., Stockmeyer, L., "Consensus in the Presence of Partial Synchrony." *Journal of the Association for Computing Machinery* **35.2** 288-323 (1988) doi:10.1145/42282.42283

[5] Nakamoto, S. "Bitcoin: A Peer-to-Peer Electronic Cash System." No Publisher (2008) https://bitcoin.org/bitcoin.pdf