FLARE CONSENSUS PROTOCOL

A PREPRINT

Sean Rowan
Flare
sean@flare.network

Naïri Usher
Flare
nairi@flare.network

November 5, 2019

ABSTRACT

The Flare Consensus Protocol (FCP) is a new construction of Federated Byzantine Agreement (FBA) consensus. An FBA construction does not rely on an economic mechanism for securing consensus because it enables individual participants to independently effect quorum slice decisions; the overlap of quorum slice decisions on the network gives rise to the network-wide rule for consensus. FCP is both leaderless and totally-ordered, making it excessively difficult for an attacker to influence which of two transactions will be ordered first in a transaction set. FCP is also asynchronous and much simpler than previous FBA constructions due to leveraging a novel federated consensus mechanism called *federated virtual voting*. These properties make FCP a compelling model for internet-level Turing-complete consensus.

1 Introduction

The question of reaching consensus in a distributed system with potentially malicious parties was first introduced via the Byzantine general's problem [LSP82]. Here, participants in the network are not necessarily trustworthy, a behaviour which more broadly falls into the category of Byzantine failure, whereby the distributed system can unpredictably appear both failed and functioning to different observers or failure detection systems. Agreement between non-Byzantine participants must be reached, a property known as safety, via the exchange of messages [Bra87]. In such scenarios, Byzantine fault tolerance approaches seek to ensure continuity in provision of the system service, assuming there are sufficient correctly operational components to maintain the service i.e. to achieve sufficient agreement or consensus. In other words, Byzantine agreement can ensure consensus even in the presence of a specified fraction of misbehaving components or members in the closed system set.

For example, traditionally, in order to maintain integrity, ledgers require a central authority to determine whether a given transaction should or not be accepted, and thus the ledger updated. Indeed, as the participants are not necessarily honest, a core concern is that of double-spending, whereby an entity would spend their coins twice. The Bitcoin [Nak+08] protocol introduced the blockchain, a cryptographically secure public transaction ledger, distributed across a network. Here, the network participants must reach a consensus as to whether to accept or reject each transaction, as well as agree on the order of transactions. Consensus in an open distributed system can be reached through a variety of mechanisms, such as proof of work [Nak+08], proof of stake [KN12; DN92] or federated Byzantine agreement (FBA) [Maz15], each of which presents both advantages and drawbacks.

Malicious behaviour such as double-spending falls into the broader category of Byzantine failure, whereby the distributed system can unpredictably appear both failed and functioning to different observers or failure detection systems. In such scenarios, Byzantine fault tolerance (BFT) [PSL80] approaches seek to ensure continuity in provision of the system service, assuming there are sufficient correctly operational components to maintain the service i.e. to achieve sufficient agreement or consensus. In other words, Byzantine agreement [PSL80] can ensure consensus even in the presence of a specified fraction of misbehaving components or members in the closed system set.

A pressing concern about purely economics-based blockchain protocols such as proof-of-stake [8] is how such systems can support large amounts of value expressed in the network state when the core set of participants is only secured leveraging a finite and smaller amount of capital, or stake. This is the notion of top-heavy blockchains.

Federated Byzantine Agreement (FBA) [Maz15] extends the original Byzantine agreement framework to an open, permissionless setting while inherently avoiding the top-heavy blockchain risk that is presented by economic solutions for providing safety such as proof of stake. Overlaps of trust in participants on the network then defines the network-wide rule for consensus. In other words, FBA enables asymmetric trust in participants while traditional BA requires symmetric trust in participants. Asymmetric trust in network participants enables each participant to define their own rule for safety when interacting with the network. For example, if a subset of participants in the network are based in a particular legal jurisdiction, they can include representation from local regulatory bodies in their safety set [SD19].

Various schemes for consensus have been developed, such as Bitcoin [Nak+08], practial BFT (pBFT) [CL+99], XRP Ledger consensus protocol [SYB+14], BFT-CUP [Alc+08] or the Stellar Consensus Protocol (SCP) [Maz15], amongst many others, and constitutes an area of ongoing research, where new approaches for improved performance are sought. For instance, virtual voting, first introduced in the Hashgraph consensus protocol [Bai16], and subsequently leveraged in the Blockmania consensus protocol [DH18], is a consensus networking complexity-reduction technique which enables very fast finality with high transaction throughput on a wide area network. However, virtual voting is susceptible to Sybil attacks [Dou02], whereby a participant joins the network multiple times with different identities, thus gaining malicious influence in the voting procedure. This means that virtual voting based networks do not inherently avoid the Sybil attack and must deploy workaround add-on mechanisms such as proof-of-stake [Dai98].

In this paper, we introduce the latest construction of the Federated Byzantine Agreement framework, the Flare Consensus Protocol (FCP), following the Stellar Consensus Protocol (SCP) [Maz15] which was the previous such construction. FCP has several advantages over SCP, notably: FCP is leaderless, asynchronous Byzantine fault tolerant, has totally-ordered transactions and is highly scalable due to its usage of a novel networking complexity-reduction technique called federated virtual voting. Furthermore, FCP is a novel generalisation of virtual voting called *federated virtual voting* that retains the exact same reduced networking complexity as the traditional virtual voting technique, but inherently overcomes the Sybil attack by existing within the Federated Byzantine Agreement framework.

This paper proceeds as follows. First, in section 2, necessary background definitions related to the FBA construction are introduced. Next, in section 3 the Flare Consensus Protocol is introduced and finally, in section 4 the FCP is compared to related consensus algorithms.

2 Preliminaries

The FCP builds upon the framework of FBA as well as virtual voting, as introduced by the Hashgraph consensus protocol [Bai16]. FCP relies on ideas regarding to graphs, which are next introduced. Then, key ideas of FBA are presented, as well as terminology used throughout the paper and some key assumptions underpinning the protocol.

2.1 Graph theory

A graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ consists of a set of vertices $\mathbf{V}$ and directed edges $\mathbf{E}$. A Directed Acyclic Graph (DAG) is a directed graph with no cycles. A directed edge from vertex $u \in \mathbf{V}$ to $v \in \mathbf{V}$ is written (u, v) i.e. where the ordering matters. A *path* of length n from $u \in \mathbf{V}$ to $w \in \mathbf{V}$ is a sequence of vertices $v_0, \ldots, v_n \in \mathbf{V}$ such that $(v_i, v_{i+1}) \in \mathbf{E}$, for $v_i \in \mathbf{V}, i = 0, \ldots, n-1$, with $v_0 = u$ and $v_n = w$. In this case, w is said to be *reachable* from v, written as $u \to w$. This relation induces a *partial ordering* i.e. $a \to b$ and $b \to c$ implies $a \to c$.

Let anc(u) denote the set of ancestors of $u \in \mathbf{V}$ i.e. anc$(u) = \{v \in \mathbf{V} | v \to u\}$. The parents of u are any vertices $v \in \mathbf{V}$ such that $(v, u) \in \mathbf{E}$.

2.2 Federated Byzantine Agreement (FBA)

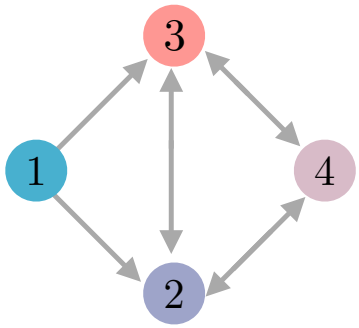
Traditional Byzantine Agreement networks can only withstand (N-1)/3 failures to ensure safety, where N is the number of participants. Federated Byzantine Agreement networks do not necessarily have this restriction, and can withstand significant numbers of node failures depending on the network-wide quorum overlap characteristics. This property enables FBA networks to inherently avoid the Sybil attack, because nodes can independently choose to not include malicious nodes that they do not trust within their quorum slice set.

The Stellar Consensus Protocol (SCP) [Maz15] is a construction for FBA, see definition 2.1 and illustrated in figures 1 and 2, which achieves optimal safety guarantees such that other FBA constructions can be as safe as SCP but not safer. An important feature is that participants in the network can choose which others they wish to trust, known as a quorum slice. This thus enables individual nodes to independently effect quorum slice decisions - market forces then cause overlaps of quorum slice decisions on the network that give rise to the network-wide rule for consensus. A transaction is settled when enough of the network accepts it, which is achieved via a voting procedure.

Definition 2.1. [Maz15] *FBAS* A federated Byzantine agreement system, or FBAS, is a pair $\langle \mathbf{V}, \mathbf{Q} \rangle$ comprising a set of nodes $\boldsymbol{V}$ and a quorum function $\mathbf{Q} : \mathbf{V} \to 2^{2^{\mathbf{V}}} \setminus \{\varnothing\}$ specifying one or more quorum slices for each node, where a node belongs to all of its own quorum slices - i.e. $\forall v \in \mathbf{V}, \forall q \in \mathbf{Q}(v), v \in q$. (Note 2^X denotes the powerset of X).

A quorum can thus be defined as a set of nodes, where each node has at least one quorum slice contained in the quorum. Note that in a non-federated Byzantine agreement, all nodes need to accept the same slices, and there is thus no distinction between a quorum and a quorum slice.

Definition 2.2. [Maz15] *quorum* A set of nodes $\mathbf{U} \subseteq \mathbf{V}$ in FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$ is a quorum iff $\mathbf{U} \neq \varnothing$ and $\mathbf{U}$ contains a slice for each node - i.e., $\forall v \in \mathbf{U}, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq \mathbf{U}$.



;

Figure 1: An example FBAS with 4 nodes labelled $1, 2, 3$ and 4. The arrows represent a node including another node that it points to in its quorum slice function $\mathbf{Q}(v)$, for $v \in \{1, 2, 3, 4\}$. Nodes 2, 3 and 4 each have a quorum consisting of nodes $\{2, 3, 4\}$. Node 1 only includes 2 and 3 in its quorum slice, but they both rely on 4 within their quorum slice, thus $\mathbf{Q}(1) = \{1, 2, 3, 4\}$.

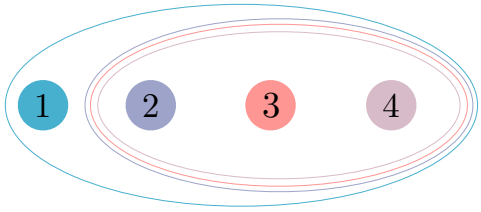


Figure 2: The FBAS example illustrated in figure 1 can alternatively be represented using the above color scheme. Here, the quorum of node 1, colored in blue, consists of nodes $1, 2, 3, 4$, and similarly for the other nodes.

Definition 2.3. [Maz15] *quorum intersection* An FBAS enjoys quorum intersection iff any two of its quorums share a node—i.e., for all quorums $\mathbf{U}_1$ and $\mathbf{U}_2$, $\mathbf{U}_1 \cap \mathbf{U}_2 \neq \varnothing$.

Definition 2.4. [Maz15] *delete* If $\langle \boldsymbol{V}, \mathbf{Q} \rangle$ is an FBAS and $\mathbf{B} \subseteq \mathbf{V}$ is a set of nodes, then to delete $\mathbf{B}$ from $\langle \mathbf{V}, \boldsymbol{Q} \rangle$, written $\langle \mathbf{V}, \mathbf{Q} \rangle^{\mathbf{B}}$, means to compute the modified FBAS $\langle \mathbf{V} \setminus \mathbf{B}, \mathbf{Q}^{\mathbf{B}} \rangle$ where $\mathbf{Q}^{\mathbf{B}} = \{q \setminus \mathbf{B} | q \in \mathbf{Q}(v)\}$.

A set of nodes is then said to be a dispensable set *DSet*, see definition 2.5, if deleting it does not interfere with the network's ability to reach consensus. In other words, the safety and liveness of nodes outside the DSets do not depend on it. Thus, in an ideal scenario, all ill-behaved nodes and failed nodes would belong to a DSet, and thereby not affect the system's ability to reach consensus.

Definition 2.5. [Maz15] *DSet* Let $\langle \mathbf{V}, \mathbf{Q} \rangle$ be an FBAS and $\mathbf{B} \subseteq \mathbf{V}$ be a set of nodes. We say $\mathbf{B}$ is a dispensible set, or D-Set, iff:
(1) (quorum intersection despite $\mathbf{B}$) $\langle \mathbf{V}, \mathbf{Q} \rangle^{\mathbf{B}}$ enjoys quorum intersection, and
(2) (quorum availability despite $\mathbf{B}$) Either $\mathbf{V} \setminus \mathbf{B}$ is a quorum in $\langle \mathbf{V}, \mathbf{Q} \rangle$ or $\mathbf{B} = \mathbf{V}$.

On the other hand, if a given set belongs to each of the quorum slices for a given node, it is then said to be v-blocking.

Definition 2.6. [Maz15]*v-blocking*. Let $v \in \mathbf{V}$ be a node in FBAS $\langle \mathbf{V}, \boldsymbol{Q} \rangle$. A set $\mathbf{B} \subseteq \mathbf{V}$ is v-blocking iff it overlaps every one of v's slices - i.e., $\forall q \in \mathbf{Q}(v), q \cap \mathbf{B} \neq \varnothing$.

2.3 Node terminology

In a distributed setting, a major concern is that of malicious actors seeking to disrupt the system's correct functioning. But, how is a bad actor more formally defined? In order to capture this, the notion of *well-behaved* (sometimes referred to as honest) is introduced, which captures the behaviour of a good actor. Informally, a good actor must informally behave sensibly, that is, obey the protocol, make good choices regarding his or her quorum slices, and respond to requests. Otherwise, it is said to be *ill-behaved* and suffer of Byzantine failure.

Throughout the progression of the protocol over time, the goal is for participants to apply updates, which is called a slot. Next, a node v decides to apply an update x to slot i—where a slot is for instance a transaction number in a ledger—if it has first applied the update to all the dependencies on which slot i depends, and second it believes all other correctly functioning nodes will apply it too. In this case, v is said to have externalised x for slot i.

Definition 2.7. [Maz15] *safety* A set of nodes in an FBAS enjoy *safety* if no two of them ever externalise different values in the same slot. Else, it is *divergent*.

Definition 2.8. [Maz15] *liveness* A set of nodes in an FBAS enjoys liveness if it can externalize new values without the participation of any failed (including ill-behaved)nodes. Else, it is *blocked*.

Definition 2.9. A well-behaved node is said to be *correct* if it is both safe and live. Else it is *failed*.

Definition 2.10. [Maz15] *intact* A node v is said to be intact iff there exists a DSet B containing all ill-behaved nodes and such that $v \notin \mathbf{B}$.

Finally, note that it is possible for a node who was once honest to become ill-behaved and vice-versa.

2.4 Assumptions

The protocol relies on a few common assumptions. The first pertains to the impossibility of forging messages.

Assumption 2.1. The messages nodes send to one another can not be forged, that is, nodes are named by a public key and use cryptographically secure digital signatures.

Next, it is commonly assumed in the context of asynchronous systems that although messages may be delayed, they will eventually get delivered. Note that the order in which they arrive will not necessarily correspond to the order in which these were sent.

Assumption 2.2. The network will eventually deliver messages between well-behaved nodes, although these may be arbitrarily delayed and/or re-ordered.

Finally, the last assumption guarantees the existence of quorum slices.

Assumption 2.3. All node's quorums intersect after deleting the DSet.

This concludes the preliminary section. In the next section, the Flare Consensus Protocol is introduced and analysed.

3 Flare Consensus Protocol

3.1 Generating a global graph

The network consists of a set of N nodes henceforth referred to as *participants* for disambiguation, which are represented by a set $\mathbf{V}$, where $|\mathbf{V}| = N$. A quorum function $\mathbf{Q}$ is defined over the set of participants, and these thus form an FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$. Let $\mathcal{U}$ denote the set of all quorums, and more specifically let $\mathcal{U}_x = \{U \in \mathcal{U} | x \in \mathbf{V}\}$ denote the set of quorums of participant $x \in \mathbf{V}$.

The participants wish to perform arbitrary transactions with one another, and crucially, to order these transactions in order to maintain network integrity. In the following, the particular case of *financial* transaction will be used for example purposes. It is important to note that FCP is a generalised protocol that handles any type of stateful deterministic transaction.

For instance, Alice (participant $A \in \mathbf{V}$) wishes to send a transaction to Bob (participant $B \in \mathbf{V}$). In practice, this is achieved by her creating a *message* containing, amongst other information, the amount she wishes to send Bob. Next, this information needs to be communicated to all the participants as efficiently as possible, in order for them to agree on the ordering of transactions. This is the question of consensus, and in FCP is achieved via a *voting procedure* on certain widely shared messages i.e. messages which have been communicated to a wide number of participants.

Thus, messages can either be created, sent or received. For example, at the start of the protocol, the first message created by Alice is referred to as $m_A^{(1)}$. In order for the information to now be communicated to other participants and thus become global knowledge, a *gossip protocol* [Dem+88] is executed. Here, Alice picks another node according to a probability distribution $p(\theta)$ over the remaining participants. In the simplest case, this is the uniform probability distribution over all remaining participants. Say, she picks Charlie (participant $C \in \mathbf{V}$) and sends him the message $m_A^{(1)}$.

Upon reception of this message, Charlie must create a *new* message containing information about the message received, as well as information about the last message *he* himself created, as is for instance done with Merkle trees [Mer87]. The creation of this new message is also his chance to attach a transaction to this message load, which he then propagates via the gossip protocol. The creation and propagation of messages between participants is represented by a DAG, where the vertices represent the messages, and a directed edge represents a message being sent from one participant to another, thus resulting in the creation of a new vertex. Figure 3 illustrates an example of message flow.

An outside observer, watching this procedure and the messages being sent, would thus be able to create the *global graph* $\mathcal{G}$ =(V, E) where V denotes the set of vertices in the graph i.e. messages and E the set of edges i.e. connections between messages. This DAG contains the history of the messages communicated, received and created by each participant. Yet, no participant has direct access to this information: indeed, they can only ever be aware of messages they themselves received, as well as any additional information pertaining to the history contained within it. Thus, each participant will represent this partial information as a *local DAG*, defined in def 3.4. Thus, the protocol consists in each participant building this local structure and then running the federated virtual voting procedure, see section 3.8 for the purpose of ordering these messages.

3.2 Messages

More formally, the message is a data structure labelled with five fields: a hash of each of its two parents, a transaction, a hash of its creator's quorum slice function as well as a digital signature. These five fields uniquely determine the message.

Definition 3.1. *message* A *message* $m_x^{(n)} \in V$ is the nth data structure created by participant $x \in \mathbf{V}$, which consists of:
(1) Payload data,
(2) Hash$[m_x^{(n-1)}]$, where $m_x^{(n-1)}$ is the parent of $m_x^{(n)}$, for $n \geq 2$
(3) Hash$[m_y^{(k)}]$, where $m_y^{(k)}$ is the parent, with $y \neq x$, for $n \geq 2$
(4) Hash$[\mathbf{Q}(x)]$ where $\mathbf{Q}(x)$ is the quorum slice function for participant x
(5) Sign$(m_x^{(n)})$,
where Hash(x) denotes a cryptographically secure hash function, and where Sign$(m_x^{(n)})$ denotes a digital signature scheme for message authentication.

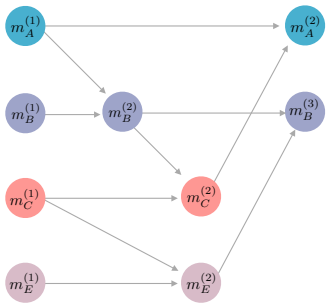


Figure 3: This is an example message flow within the consensus algorithm by the four participants, where messages flow from left to right in time. Here, there are four participants (A, B, C, E), each with its quorum slice: $\mathbf{Q}(A) = \{A, B, C, E\}$ and $\mathbf{Q}(B) = \mathbf{Q}(C) = \mathbf{Q}(E) = \{B, C, E\}$. In this system, the ancestors of message $m_B^{(2)}$ are messages $\{m_A^{(1)}, m_B^{(1)}\}$, and the ancestors of message $m_A^{(2)}$ are $\{m_A^{(1)}, m_C^{(2)}, m_C^{(1)}, m_B^{(2)}, m_B^{(1)}\}$. The parents of message $m_A^{(2)}$ are $m_A^{(1)}$ and $m_C^{(2)}$.

Definition 3.2. *initial message* The *initial message*, i.e. when $n = 1$, is the first message created by a participant, at the start of the protocol.

Note that when the superscript is not relevant, it is dropped for simplicity e.g. a message created by participant $x \in \mathbf{V}$ will sometimes be referred to as m_x.

If a participant creates two messages, neither of which contain information about the other, then double-spending and other malicious behaviour could occur; this is known as a fork.

Definition 3.3. *fork* The pair of messages $(m_x^{(n)}, m_x^{(k)})$ is a fork if $m_x^{(n)}$ and $m_x^{(k)}$ are created by the same participant $x \in \mathbf{V}$, but neither is an ancestor of the other.

More generally, let $\mathcal{F}_x$ denote the set of all forks created by participant x, for any $x \in \mathbf{V}$. Note that, by definition, well-behaved nodes do not create forks.

3.3 Generating local graphs

Yet, each participant will only be aware of certain messages, and a certain portion of the network history, which is represented by a DAG. Let $\mathcal{G}_x$ denote the DAG of participant $x \in \mathbf{V}$, and let V_x denote the set of messages in participant x's DAG, i.e. the vertices in the DAG.

Thus, for instance, Alice is aware of the messages she receives, and the information they contain, which in turn contain information about the history of the network via the hash of the ancestors, similarly to a Merkle tree [Mer87], which is a binary tree of hashes. The information she is aware of is represented by a local graph $\mathcal{G}_A$. This will potentially be different from another participants knowledge of the network, say Bob, which is represented by a DAG $\mathcal{G}_B$.

Definition 3.4. *local graph* Each participant $x \in \mathbf{V}$ generates a *local graph* which is a DAG $\mathcal{G}_x = (V_x, E_x)$, consisting of a set of vertices $V_x = \{m_i^{(k)} | i \in \mathbf{V}, k \in \mathbb{N}\}$ corresponding to the set of messages participant x is aware of, and a set of directed edges $E_x = \{(m_i^{(k)}, m_j^{(k')}) | i, j \in V_x, k, k' \in \mathbb{N}\}$ where (m_i, m_j) is a directed edge from message $m_i^{(k)}$ to $m_j^{(k')}$.

3.4 Graph consistency

A local graph thus contains partial information of the global graph, and more specifically, $V_x \subseteq V$. Over time, more and more messages are shared between participants, and their local graphs come to increasingly reflect the global graph. It is by this mechanism that each participant will then be able to implement a local voting procedure on their own local DAG.

Given two local DAGs, say $\mathcal{G}_A$ and $\mathcal{G}_B$, certain vertices from each represent the same message. This is formally defined as follows.

Definition 3.5. *identical messages* Given two local graphs $\mathcal{G}_A$ and $\mathcal{G}_B$, two messages $m_x^{(n)} \in \mathbf{V}_A$ and $\tilde{m}_x^{(n)} \in \mathbf{V}_B$, for $x, y \in \mathbf{V}$ are said to be *identical* if they contain the same payload as well as the same three hashes and digital signature. The message is then said to *occur* in both of the graphs.

This then allows for the notion of consistent graphs to be introduced.

Definition 3.6. *consistent* Two local graphs $\mathcal{G}_A$ and $\mathcal{G}_B$ are consistent, written as $\mathcal{G}_A \cong \mathcal{G}_B$ iff for any two identical messages $m_z^{(n)} \in \mathcal{G}_A$ and $\tilde{m}_z^{(n)} \in \mathcal{G}_B$, for $z \in \mathbf{V}$, both contain the same set of ancestors with the same parent edges between those ancestors.

The next lemma says that the local data structures i.e. DAGs held by the individual participants all satisfy this property of consistency.

Lemma 3.1 (consistent DAGs)**.** *All participants have consistent local DAGs.*

Proof. Let message $m_x^{(n)}$ be contained in both two local DAGs $\mathcal{G}_A$ and $\mathcal{G}_A$. By definition 3.5, this means both messages contain the same three hashes and and in particular the same hashes of both parents. This means that both graphs contain the parents of $m_x^{(n)}$. As the cryptographic hash functions are assumed to be secure i.e. no collision, therefore the parents must be the same. By induction, all ancestors of $m_x^{(n)}$ must be the same in $\mathcal{G}_A$ and $\mathcal{G}_B$, thus satisfying condition 1 of consistency. By assumption, the second condition is satisfied. Thus, $\mathcal{G}_A$ and $\mathcal{G}_B$ are consistent. □

3.5 Graph connectivity

As local DAGs are constructed, participants will wish to reach consensus regarding the ordering of transactions. This is achieved by considering messages which are deemed more important than others, in the sense that these are more widely communicated and shared amongst participants. This is determined by the connectivity of the local graph i.e. on the degree of reachability of vertices.

The existence of a fork may threaten the integrity of the system. Thus, in the context of the local DAGs of messages, the definition of vertex reachability is modified to take this into account.

Definition 3.7. *reachable* A message m_x is said to be *reachable* from a message m_y in $\mathbf{V}_A$, written $m_y \rightarrow m_x$ if
1) there exists a path from m_y to m_x, and
2) anc$(m_x) \cap \mathcal{F}_y = \varnothing$.

Given a local graph, let $\mathcal{P}_{u \rightarrow v}$ denote the set of all paths from vertex u to vertex v, which do not include a fork by the creator of v. Then, $|\mathcal{P}_{u \rightarrow v}|$ denotes the number of such paths. Intuitively, the more paths that connect two vertices, the more strongly connected these two are. To capture this, *strong reachability* is introduced, whereby not only do two vertices have to be connected by a sufficient number of paths, but these must furthermore be via a sufficient number of relevant participants. More precisely, two vertices u and v must be connected by paths going through a quorum of vertices of v's creator.

Definition 3.8. *strongly reachable* A message m_x is said to be *strongly reachable* by a message m_y, written $m_y \implies m_x$ if
1) $m_y \rightarrow m_x$, and
2) $\exists U_x \in \mathcal{U}_x$ st $\forall i \in U_x$, $\exists m_i$ st $m_y \rightarrow m_i$ and $m_i \rightarrow m_x$.

Now, given a message m_x, one can consider the set of *all* messages which may be strongly reached from m_x.

$$\mathcal{R}(m_x) = \{m_i | m_x \implies m_i, i \in \mathbf{V}\}. \tag{1}$$

A restriction can be placed on this set in order to only consider strongly reachable vertices from the *same* creator:

$$\mathcal{R}'(m_x^{(k)}) = \{m_x^{(n)} | m_x^{(k)} \implies m_x^{(n)}\}, \tag{2}$$

where $\mathcal{R}'(m_x) \subseteq \mathcal{R}(m_x)$.

Next, consider the case where an ill-behaved participant x creates a fork. What would be the consequences of this for two consistent local graphs? The following lemma tells us that if a participant creates a fork, then only one of these messages can strongly reach a message in the next round.

Lemma 3.2 (strong reachability)**.** *Consider two consistent local graphs $\mathcal{G}_A$ and $\mathcal{G}_B$, and consider participant $x \in \mathbf{V}$ to have created a fork $(m_x^{(n)}, m_x^{(n')})$. Furthermore, let $m_z \in V_A$ be a message from an intact node s.t. $m_x^{(n)} \implies m_z$. Then, $m_x^{(n')} \not\implies m_j, \forall m_j \in V_B$.*

Proof. The proof is by contradiction i.e. assume there exists a message $m_j \in V_B$ st $m_x^{(n')} \implies m_j$. By definition of strongly reaching, this means that $\exists U_j \in \mathcal{U}_j$ st $m_x^{(n')} \rightarrow m_l$ and $m_l \rightarrow m_j, \forall m_l \in U_j$. Furthermore, by assumption, $m_x^{(n)} \implies m_z$, where $m_x^{(n)}, m_z \in V_A$. By point (2) of the definition of strong reachability, there exists a quorum $U_z \in \mathcal{U}_z$ st $\forall i \in U_z, m_x^{(n)} \rightarrow m_i$ and $m_i \rightarrow m_z$. As by assumption $\mathcal{G}_A \approx \mathcal{G}_B$, all quorums have quorum intersection after deleting the DSet $\mathbf{B}$. Let $c \in U_j \cap U_z$ belong to this intersection. By definition of DSet, $\langle \mathbf{V}, \mathbf{Q} \rangle^{\mathbf{B}}$ enjoys quorum intersection. Furthermore, c is well-behaved. And let $m_c \in V_A$ and $\tilde{m}_c \in V_B$ be messages created by this participant in local graphs such that $m_x^{(n')} \rightarrow \tilde{m}_c$ and $\tilde{m}_c \rightarrow m_j$ and $m_x^{(n)} \rightarrow m_c$ and $m_c \rightarrow m_z$. Now, by definition of well-behaved, $(m_c, \tilde{m}_c)$ can not be a fork and thus one is the ancestor of the other. Let m_c be the ancestor and thus we have that $m_c \rightarrow \tilde{m}_c$. Thus, we have that $m_x^{(n)} \rightarrow m_c$ and $m_c \rightarrow \tilde{m}_c$. Thus, $m_x^{(n)}$ is an ancestor of $\tilde{m}_c$. But, as $m_x^{(n')} \rightarrow \tilde{m}_c$. So, both $m_x^{(n')}$ and $m_x^{(n)}$ are ancestors of $\tilde{m}_c$. By assumption, $(m_x^{(n)}, m_x^{(n')})$ is a fork, thus $m_x^{(n')} \not\rightarrow \tilde{m}_c$, hence contradiction. □

3.6 Slot Number

Strongly reachable messages introduce a partial ordering on the set of messages. More specifically, at the start of the protocol, initial messages are created. As more messages are created and transmitted, some of these will be widely reachable, i.e. more strongly connected to previous messages. In order to track the spread and connectivity of messages

in a given local DAG G_A, the message *slot number* $\sigma_A[m]$ is introduced. At the start of the protocol, each message is in the first slot i.e. $\sigma_A[m_i^{(1)}] = 1, \forall i \in \mathbf{V}$. Now, let $m_x^{(n)}$ be the first message strongly reachable from $m_x^{(1)}$ by the same creator i.e. $m_x^{(n)} \in \mathcal{R}'(m_x^{(1)})$. Furthermore, let $m_x^{(n)}$ be strongly reachable from a set of initial messages that forms a quorum for x. In this case, the slot number of message $m_x^{(n)}$ increases by one. If these conditions were not met, the slot number would simply be the maximum slot number of the message parents, which in this case would be 1.

Definition 3.9. *Slots* The *slot number* $\sigma_A[m_x] : m_x \in V_A \rightarrow \mathbb{N}$, $x \in \mathbf{V}$, $A \in \mathbf{V}$, is defined as

$$\sigma_A[m_x^{(n)}] = \begin{cases} 1, & \text{if } n = 1, \\ s+1, & \text{if } \exists U_x \in \mathcal{U}_x \text{ st } \forall i \in U_x, m_x^{(n)} \in \mathcal{R}'(m_i^{(k)}), \text{ where } k \min \text{ st } \sigma_A[m_i^{(k)}] = s, \\ s, & \text{else.} \end{cases} \tag{3}$$

and where

$$s = \max_{m \in \text{anc}(m_x^{(n)})} \sigma_A[m]. \tag{4}$$

Recall that members have consistent local graphs at all times. The next lemma guarantees that if a message is contained in two local graphs, then each participant will agree on its slot number.

Lemma 3.3 (consistent slots)**.** *Let $\mathcal{G}_A$ and $\mathcal{G}_B$ be two consistent local graphs containing message $m_x^{(n)}$. Then, both will assign the same slot created number to $m_x^{(n)}$ i.e. $\sigma_A[m_x^{(n)}] = \sigma_B[m_x^{(n)}]$.*

Proof. By assumption, both $\mathcal{G}_A$ and $\mathcal{G}_B$ contain message $m_x^{(n)}$. As $\mathcal{G}_A \approx \mathcal{G}_B$, then by definition of consistency, they both contain the same set of ancestors with the same parent edges between these. This includes the first message created. Then the proof is by induction: both graphs agree on the slot number of that first message, which is 1 by definition. Now, let m_y be an arbitrary message belonging to both graphs. Then, as both graphs agree on the slot numbers of its ancestors, then they will also agree on what is the maximum slot number of the parents of m_y, say s. Next, they need to determine whether there exists a quorum of participants $U_y \in \mathcal{U}_y$ in slot s such that it is strongly reachable from the core messages of slot s. As the graph connectivity on the ancestor set is the same, both graphs will agree on the slot number of m_y. Therefore they will agree on the slot number of all messages they share, including m_x. □

Thus, henceforth, the subscript will be dropped from the slot number function i.e. $\sigma[x]$ will be written for $\sigma_A[x]$, for all participants $A \in \mathbf{V}$.

3.7 Core messages

Thus, from the slot number definition, the set of first messages created in each in each slot is of particular importance. These are referred to as *core messages*.

Definition 3.10. *Core message* The set of *core messages* is defined as

$$\mathcal{C} = \{m_i^{(k)} | \min k \text{ st } \sigma[m_i^{(k)}] = s, \forall s \in \mathbb{N}, \forall i \in \mathbf{V}\}. \tag{5}$$

Furthermore, if a core message m_x is not a fork, it is then said to be ***unique***, i.e. $m_x \in \mathcal{C}$ st $m_x \notin \mathcal{F}_x$.

The set of core messages is then partitioned into disjoint sets according to each slot number

$$\mathcal{C} = \bigcup_k C_k, \tag{6}$$

where $C_i = \{m \in \mathcal{C} | \sigma[m] = i\}$. Furthermore, introduce $\Delta(m, n) = \sigma[m] - \sigma[n]$, for $\sigma[m] \geq \sigma[n]$.

The next step, will be for the participants to reach consensus on the importance of strongly reached messages. This is achieved via the *federated virtual voting* procedure.

3.8 Federated Virtual Voting

3.8.1 Overview

Now that each participant has constructed their own local DAG, the goal is for them to reach consensus regarding the ordering of the messages. This is achieved via the *federated virtual voting* algorithm, whereby certain core messages are determined to be *pivots*, via a voting process. In figure 4, the path to confirmation of a statement a is illustrated. Next, once these have been evaluated, messages are assigned an order number, which then allows for all messages to be ordered and for consensus to be achieved.

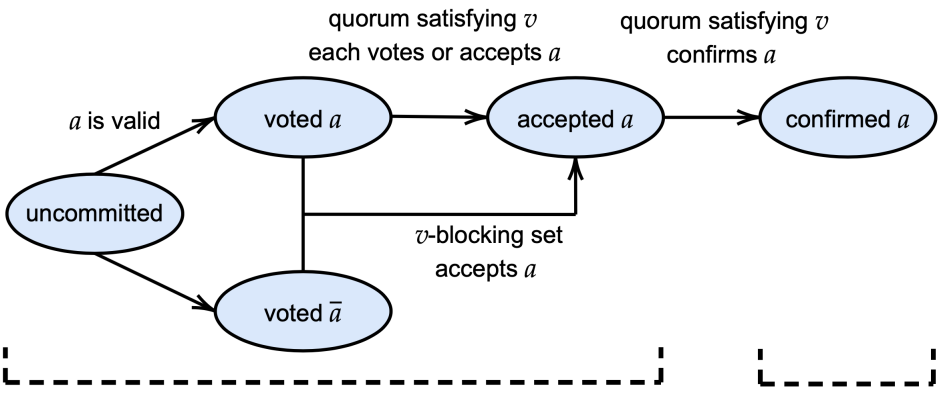


Figure 4: The FBA path to confirmation of a statement a [Maz15].

3.8.2 Vote

The protocol consists in each participant implementing a voting procedure, whereby each core message casts a vote regarding the connectivity of a previous core message, via the vote function. For instance, some core message m_y will vote on a core message m_x from a previous slot (recall each participant has one core message at each round). The vote cast is determined by the voting function, see def 3.12, and this process is called an election.

Definition 3.11. *Election* An *election* refers to the evaluation of the vote function i.e. the outcome of one core message casting a vote on a core message from a previous slot.

The vote function is first defined for two core messages in consecutive slots. If m_y is reachable from m_x, then m_y votes *yes* ($a = 1$) and otherwise *no* ($a = 0$). If messages m_y and m_x are separated by more than one slot, then the vote is implemented via a *tally function*, see definition 3.14, which tallies up the votes cast in the *previous* slot. It does so through the construction of voter sets, see definition 3.13, whose votes are then tallied up. When the voting message, say m_y, finds that their voter set is a quorum for the message being voted on and that these all vote the same way, then m_y will also vote as such. When such a decision is made, m_y furthermore determines m_x to be a pivot, via the evaluation of the *pivot function*, see definition 3.15. Finally, in the voting function, a coin flip is introduced in order to guarantee termination of the voting procedure.

Definition 3.12. *Voting* The *voting function* $v(m_y, m_x) : \mathcal{C} \times \mathcal{C} \to \{0, 1, \perp\}$, defined as

$$v(m_y, m_x) = \begin{cases} [m_x \to m_y], & \text{if } \Delta(m_y, m_x) = 1, \\ \tau(m_y, m_x), & \text{if } \Delta(m_y, m_x) > 1 \text{ and } \Delta(m_y, m_x) \mod c \neq 0, \\ r(m_y, m_x), & \text{if } \Delta(m_y, m_x) > 1 \text{ and } \Delta(m_y, m_x) \mod c = 0, \\ \perp, & \text{else.} \end{cases} \tag{7}$$

where $[P]$ denotes truth of statement P, $\tau(m_y, m_x)$ is the tally function defined in 11, $r(m_y, m_x)$ is the randomise function defined in 12, $\perp$ refers to the uncommitted state, and $c \in \mathbb{N}$ is the coin flip.

The voting function relies on the tally function, which tallies up the votes of messages separated by at least two or more slots i.e. in this case, if $\sigma[m_x] = s$ then $\sigma[m_y] = s' \geq s + 2$, for $s \in \mathbb{N}$. Recall that $v(m_y, m_x)$ denotes the vote of message m_y regarding core message m_x. Here, message m_y needs to tally up the votes from the relevant i.e. in y's quorum, core messages in the *previous* slot, which can strongly reach m_y. These messages are called the voter set.

Definition 3.13. *Voter set* The *voter* set for core message (and voter) m_y in slot s' is defined as

$$\mathcal{V}(m_y) = \{m_i | m_i \in C_{s'-1}, i \in U_y, m_i \implies m_y\}. \tag{8}$$

Furthermore, the voter set $\mathcal{V}(m_y)$ for m_y can be partitioned as a follows $V(m_y) = V_1(m_x) \cup V_0(m_x)$, where

$$V_a(m_x) = \{m \in \mathcal{V}(m_y) | v(m, m_x) = a\}, \tag{9}$$

where $a \in \{0, 1\}$. Note that these sets depend on message m_y in slot s'.

Now, consider the participants who have created messages belonging to $V_a(m_x)$, for $a \in \{0, 1\}$.

$$P_a(m_x) = \{j \in \mathbf{V} | m_j \in V_a(m_x)\}. \tag{10}$$

This, then allows the tally function to be defined as follows.

Definition 3.14. *Tally* The tally function $\tau(m_y, m_x) : \mathcal{C} \times \mathcal{C} \to \{0, 1\}$ is defined

$$\tau(m_y, m_x) = \begin{cases} 1, & \text{if } P_1(m_x) \in \mathcal{U}_x \text{ or else } \exists \mathcal{B}(x) \subseteq P_1(m_x), \\ 0, & \text{if } P_0(m_x) \in \mathcal{U}_x \text{ or else } \exists \mathcal{B}(x) \subseteq P_0(m_x). \end{cases} \tag{11}$$

where $\mathcal{B}(x)$ is an x-blocking set, see definition 2.6.

Finally, in the case of coin flips, the randomise function $r(m_y, m_x) : \mathcal{C} \times \mathcal{C} \to \{0, 1\}$ is given by

$$r(m_y, m_x) = \begin{cases} 1, & \text{if } P_1(m_x) \in \mathcal{U}_x, \\ 0, & \text{if } P_0(m_x) \in \mathcal{U}_x, \\ r, & \text{else.} \end{cases} \tag{12}$$

where r is the middle bit of the signature of m_y. This thus defines the first phase of the voting procedure.

3.8.3 Accept

In the next phase, pivot messages are introduced, which are essentially core messages of particular importance.

Definition 3.15. *Pivot* A message m_x is said to be a *pivot* if $\pi_A(m_x) = 1$, where $\pi_A : \mathcal{C} \to \{0, 1\}$ is the *pivot function* defined as

$$\pi_A(m_x) = \begin{cases} 1, & \text{if } P_1(m_x) \in \mathcal{U}_x, \\ 0, & \text{if } P_0(m_x) \in \mathcal{U}_x, \\ \perp, & \text{else.} \end{cases} \tag{13}$$

Thus, pivot messages requires for both the vote and tally functions to have been computed. Note that pivot messages are by definition core messages. Let $\mathcal{P}$ denote the set of all pivots, which similarly to core messages $\mathcal{P} = \cup P_i$, where $P_i = \{m \in \mathcal{P} | \sigma[m] = i, i \in \mathbb{N}\}$. In figure 5, an example of DAG for 5 participants is illustrated, where both core messages and pivots have been computed.

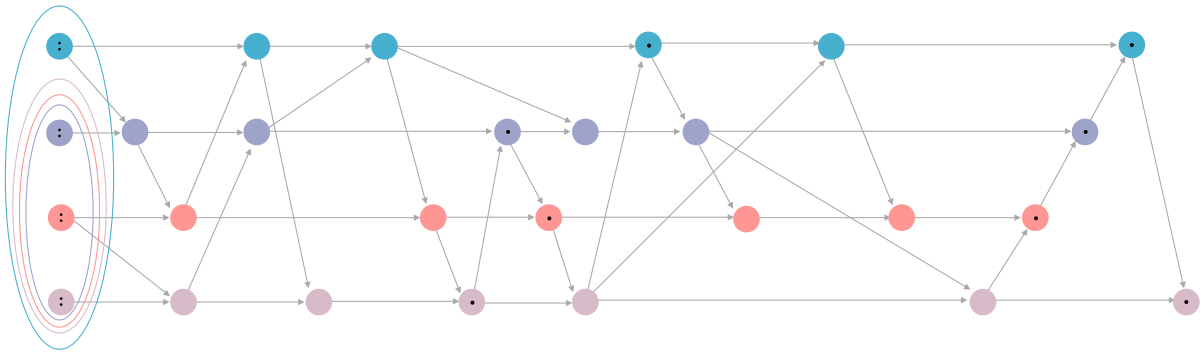


Figure 5: An example message flow in the consensus algorithm. Vertices with one black dot are core messages and vertices with two black dots have been determined to be pivots.

Thus, when a quorum of core messages votes identically regarding a given core message, agreement is said to have been reached and the election has a result. Thus, participants reach agreement regarding which core messages are pivots, and which are not.

Definition 3.16. *Election result* An election has result a in slot s when a core message in a previous slot is determined as either being or not a pivot ($a = 1$ if it is and $a = 0$ if not).

Definition 3.17. *Deciding* An election is said to have been *decided* when a core message has been determined to either be or not be a pivot. This is known as Byzantine agreement.

Core messages thus evaluate whether previous core messages are pivots or not. A priori, one could imagine that different core messages could come to different results i.e. a given message could be evaluated as a pivot or not depending on the specific voter set. The following lemma states that this is not the case, and that once a core message is determined to be a pivot for a given voter set, then any other voter set will reach the same conclusion.

Lemma 3.4. (Consistent voting) *Let $\mathcal{G}_A$ and $\mathcal{G}_B$ be two consistent local graphs. And let the algorithm running on $\mathcal{G}_A$ show that for a given election that a slot s core message m_x sends a vote $a_x \in \{0,1\}$ to a core message m_y in slot $s+1$. And let the algorithm running on $\mathcal{G}_B$ show that a slot s core message $\tilde{m}_x$ sends a vote $\tilde{a}_x \in \{0,1\}$ to a witness $\tilde{m}_y$ in slot $s+1$. Then $a_x = \tilde{a}_x$.*

Proof. Consider $m_x \in V_A$ and $\tilde{m}_x \in V_B$. The first option is that $(m_x, \tilde{m}_x)$ is a fork created by x. By lemma 3.2, then only one of these messages can strongly reach another. But, by assumption, both messages m_x and $\tilde{m}_x$ send their votes i.e. both must be strongly connected to m_y, and hence contradiction. Thus, this means that one of the messages created by x is an ancestor of the other. But, these are by assumption core messages and thus must have minimum slot number, which means they are the same. Therefore, the two votes must be coming from the identical message m_x in both graphs. The vote a message sends is calculated purely as a function of its ancestors. As the two graphs are consistent, these are the same and thus two graphs must agree on the vote, and $a_x = \tilde{a}_x$. □

Different core messages thus reach the same conclusion regarding the pivot nature of a previous core message. Furthermore, the following lemma guarantees that they will, at most, reach this conclusion within two slots of one another.

Lemma 3.5 (Consistent decisions)**.** *Let $\mathcal{G}_A$ and $\mathcal{G}_B$ be two consistent local graphs. And let $\mathcal{G}_A$ decide a federated Byzantine agreement election with result a in slot s and $\mathcal{G}_B$, has not decided prior to s, then $\mathcal{G}_B$ will decide a in slot $s+2$ or before.*

Proof. By assumption, $\mathcal{G}_A$ decided a federated Byzantine agreement election with result $a \in \{0,1\}$ in slot s. This means that there exists a core message m_x in an earlier slot i.e. $\sigma[m_x] < s$ which, in slot s, is determined as being or not a pivot i.e. $\pi(m_x) = a$. By definition of the pivot function, there then exists a set of participants $P_a(m_x)$ in slot $s-1$ which form a quorum for x and which are all voting the same way i.e. a. By lemma 3.4, in consistent graphs, core messages from same creators in same slots will vote the same way. Recall that the vote sent depends purely on ancestors, and thus the entire quorum will thus also send a to all core messages in both graphs. By definition of consistent, all quorums in $\mathcal{G}_A$ and $\mathcal{G}_B$ have quorum intersection after deleting the DSet. Thus, every core message in slot s in both $\mathcal{G}_A$ and $\mathcal{G}_B$ will vote for a (and some may decide a). Then, if slot $s+1$ is a regular slot, every core message in $\mathcal{G}_A$ and $\mathcal{G}_B$ in that slot will receive unanimous votes of a and will decide a. If slot $s+1$ is a coin flip, then all will receive unanimous votes of a, so it will not be a random selection of votes, and all will vote a, and then all will decide a in slot $s+2$. □

This thus guarantees that if one participant comes to a conclusion regarding the pivot status of a previous core message, then all participants it has quorum intersection with will concur shortly thereafter. But, could it be possible that certain core message maintain an undetermined pivot status? The next theorem answers this question by the negative.

Theorem 3.6 (Pivot computation)**.** *Pivot functions are determined eventually with probability 1.*

Proof. Consider some core message m_z. By lemma 3.5, if a participant x determines whether m_z is a pivot i.e. $\pi(m_z) = a$ for $a \in \{0,1\}$, then all participants with quorum intersection with x will decide the same way within 2 slots. So, the only way that the pivot status of m_x remains undetermined i.e. $\pi(m_x) = \perp$ is if one of these participants never decides, because no core message ever receives a quorum of unanimous votes. However, in a coin flip, if such a quorum has not yet been achieved, then all the intact nodes randomly choose their vote, and will have a nonzero probability of all choosing the same vote. As every c slot is a coin flip, and coin flips occur periodically forever, then eventually the intact nodes will become unanimous, with probability one, and then by lemma 3.5 consensus will be reached within 2 slots. □

Core messages, via the vote function and voter sets, determine the value of the pivot function for previous core messages. The following lemma guarantees that whether or not a core message is a pivot will be determined within a few slots.

Lemma 3.7 (Pivot existence)**.** *For any slot number s, for any local DAG that has at least one message m_z in slot $s+3$ created by participant z, there will be at least one core message in slot s that will be decided to be a pivot, and this decision will be made by every core message from a participant with quorum intersection with z in slot $s+3$, or earlier.*

Proof. Consider one of the local DAGs, and let $s \in \mathbb{N}$ st $m_z \in C_{s+3}$. Consider the following set of core messages $S_i = \{m_x \in C_i | m_x \implies m_y, m_y \in C_{i+1}\}$ where $i < s+3$. If $S_y \neq \varnothing$, by definition of strong reachability, this means that in slot i, there exists a $U_y \in \mathcal{U}_y$ st $m_z \to m_i, m_i \to m_y, \forall i \in U_y$. As strong reachability implies reachability, then each participant with a message in S_{s+1} is reached by a quorum of messages in S_s. Now, by lemma 3.2, none of the participants can create more than one core message in a given slot that is strongly reaching. The

existence of a core message in slot $i+1$ guarantees that a quorum of nodes is strongly reaching in slot i, and none of the participants can create more than one core message in a given slot that is strongly reaching. By assumption, m_z is a core message in $s+3$. Thus, all quorums in S_i have quorum intersection with z, $\forall\, i \leq s+2$.

Furthermore, as strong reachability implies reachability, so each participant with a message in S_{s+1} can be reached from a quorum of messages in S_s. Due to the quorum intersection property for the quorums within S_{s+1}, there must exist at least one message in S_s, call it m_x, which reaches a quorum of participants in S_{s+1}. This is because a quorum of core messages in S_{s+1} is each reached by a quorum of messages in S_s, and there must be at least a v-blocking set in S_s containing at least one message that reaches all of the core messages from the quorum in S_{s+1}. Therefore, a quorum of participants within S_{s+1} will vote *yes* in the election for m_x being a pivot. Therefore, every message in S_{s+2} will receive at least a v-blocking set of *yes* votes for whether m_x is a pivot which causes the system-wide status to be *yes*-valent, and will therefore vote for m_x being a pivot (and may or may not decide that m_x is a pivot). Therefore, the message in S_{s+3} will receive unanimous votes for m_x being a pivot, which will cause it to decide that m_x is a pivot. Therefore, every participant with an message in slot $s+3$ that has quorum intersection with the x will first decide that m_x is a pivot in either slot $s+2$ or $s+3$. □

Finally, the next lemma considers the impact of adding new messages to the DAG on the set of pivots of a given slot.

Lemma 3.8 (No new pivots)**.** *Let $\mathcal{G}_A$ be such that it does not contain message m_x, but does contain all the ancestors of m_x, and let ${\mathcal{G}'}_A$ be the result of adding m_x to $\mathcal{G}_A$. Furthermore, let m_x be a core message created in slot s, and $\mathcal{G}_A$ has at least one core message in slot s that has been decided as being either a pivot or as not a pivot. Then, m_x will be decided as not a pivot in ${\mathcal{G}'}_A$.*

Proof. By assumption, there exists a core message, say $m_w \in \mathcal{C}_s$, in $\mathcal{G}_A$ in slot s which determines if a core message in slot s is a pivot. None of the ancestors of m_w is reachable from m_x, as $m_x \notin \mathcal{G}_A$. As all messages except m_x are contained in both graphs, then they all have the same ancestors in both, which means m_x can not be an ancestor of any message. Thus, m_x can not reach any message in either of the local DAGs. Thus, the ancestors will vote 0 in $s+1$, and it will thus not be a pivot in $\mathcal{G}'_A$ in $s+2$. □

Thus, each core message is evaluated as either being a pivot or not via. Crucially, it has been shown that core messages from different participants will all agree as to whether a message is a pivot or not. These pivots are then used in order to determine the ordering of messages in the DAG.

3.8.4 Confirm

An *order number* is finally attached to each message. For this to be computed, the pivot function must have been determined for all core messages with slots smaller or equal to the message in question. The domain of the order function is then given by

$$D = \{m_x \in V_A | \forall m_y \in \mathcal{C} \text{ st } \sigma[m_y] \leq \sigma[m_x], \pi(m_y) \neq \perp\ \forall y \in U_x, \forall U_x \in \mathcal{U}_x\}. \tag{14}$$

In other words, these are the messages for which the order number can be computed, and it is determined as follows.

Definition 3.18. *Order number* The *order number* of a message m_x is determined by the order function $o_A : D \to \mathbb{N}$, $\forall A \in \mathbf{V}$, and defined as

$$o_A[m_x] = s, \tag{15}$$

where $s = \min_{t \in \mathbb{N}} \big(\sigma[m_z] = t\big)$, where $m_x \in \text{anc}(m_z)$, $\pi(m_z) = 1, m_z \notin \mathcal{F}_z\ \forall z \in U_x, \forall U_x \in \mathcal{U}_x$ and D is defined in 14.

Once an order number has been attributed, a *causal order* is computed via a deterministic ordering process, such as for instance one leveraging Lamport timestamps [Lam79], a well-known algorithm which determines the order of events in a distributed setting.

Thus, messages can now be sorted first by order number, and then by causal order, and ultimately if necessary, by their hash. The next theorem guarantees that all messages will be assigned an order number, thus guaranteeing for an ordering of the messages to be reached.

Theorem 3.9 (Message ordering)**.** *Each message $m_x^{(n)}$ created by an intact participant x will eventually be assigned an order number in the total order of messages, with probability 1.*

Proof. Consider message $m_x^{(n)}$ created by a well-behaved participant x. By virtue of being well-behaved, it will sync often, and by assumption 2.2 these messages will eventually be delivered. Thus, all well-behaved participants, amongst

which are those with quorum intersection with x, will eventually learn of $m_x^{(n)}$. Therefore, there will eventually be a slot, say s', where all the unique pivots by participants with quorum intersection with x are descendants of $m_x^{(n)}$. Therefore in slot s', or possibly earlier, there will be a slot s where all the pivots are descendants of $m_x^{(n)}$. This is then sufficient for $m_x^{(n)}$ to be assigned an order number s (as well as a causal ordering) within that slot. Its consensus place in history will be fixed. Furthermore, it is not possible to later discover a new message m_y that will come before $m_x^{(n)}$ in the consensus order. Indeed, in order to come earlier in the consensus history, m_y would have to have an order number less than or equal to s. For this to occur, all the pivots in slot s must have received m_y. But, once the set of pivots is known for a slot, all of their ancestors are also known, and there is thus no way of discovering new ancestors for them in the future as the DAG grows. Furthermore, once every known core message in a slot is decided to be a pivot or not, it isn't possible for a slot to gain new pivots in the future. Any new slot s core message, whose creator has quorum intersection with the creators of the known slot $s+1$ core messages, that is discovered in the future will not be an ancestor of the known slot $s+1$ core messages. So, the consensus will immediately be reached that it is not a pivot because it cannot be strongly reached by any quorum of slot $s+1$ core messages. Therefore, once a message is assigned a place in the total order, it will never change its position, neither by swapping with another known message, nor by new messages being discovered later and being inserted before it. □

4 Conclusions

FCP is a simplified, and fair FBA construction. It exists within the context of FBA agreements, and is effectively a generalisation of virtual voting, as introduced by Hashgraph. FCP is now compared to Hashgraph, the SCP as well as Blockmania.

Suppose there is a fixed number of participants N in the network. If they each set their quorum slices to all combinations of $(2N+1)/3$ participants, i.e. such that each member's set of slices includes that member themself, then this special case reverts down to the Hashgraph Consensus Protocol. The Hashgraph Consensus Protocol can only exist in this (2N + 1)/3 configuration i.e. traditional Byzantine Agreement, and does not inherently overcome the Sybil attack. The Flare Consensus Protocol inherently overcomes the Sybil attack due to existing in the Federated Byzantine Agreement framework, enabling participants to independently effect quorum slice decisions.

Blockmania [DH18] is a traditional Byzantine agreement consensus protocol, formulated in a PBFT [CL+99] framework, that leverages the virtual voting technique established in the Hashgraph Consensus Protocol to yield a significant communication complexity savings over normal PBFT.

The Flare Consensus Protocol achieves a significant reduction in communication complexity over ballot-based approaches (such as Stellar Consensus Protocol) through the Federated Virtual Voting technique. The Stellar Consensus Protocol is also leader-based, with leader-selection being based on how centrally-located the leader-candidate is within the FBA network. This causes participants that exist on the periphery of the network to have minimal to no power in the network, significantly limiting the impact that high numbers of participants can have on consensus. The Flare Consensus Protocol however is completely leaderless, and gives fair control over the network that directly aligns with increasing numbers of participants on the network. This is very important for achieving fairness in many contexts, for example darkpool trading.

The Flare Consensus Protocol (FCP) leverages the Federated Byzantine Agreement notation and theoretical guarantees established in the Stellar Consensus Protocol paper in order to introduce federated virtual voting. FCP is a simplified and fair Federated Byzantine agreement construction, which is proven to Byzantine Fault Tolerance. These properties make FCP a compelling model for internet-level Turing-complete consensus.

Thank you to Hugo Philion and Artem Vorobiev for helpful feedback on this paper.

References

[Alc+08] E. A. Alchieri et al. "Byzantine consensus with unknown participants". In: *International Conference On Principles Of Distributed Systems*. Springer. 2008, pp. 22–40.

[Bai16] L. Baird. "The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance". In: *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep.* (2016).

[Bra87] G. Bracha. "Asynchronous Byzantine agreement protocols". In: *Information and Computation* 75.2 (1987), pp. 130–143.

[CL+99] M. Castro, B. Liskov, et al. "Practical Byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999, pp. 173–186.

[Dai98] W. Dai. “B-money”. In: *Consulted* 1 (1998), p. 2012.

[DH18] G. Danezis and D. Hrycyszyn. “Blockmania: from Block DAGs to Consensus”. In: *arXiv preprint arXiv:1809.01620* (2018).

[Dem+88] A. Demers et al. “Epidemic algorithms for replicated database maintenance”. In: *ACM SIGOPS Operating Systems Review* 22.1 (1988), pp. 8–32.

[Dou02] J. R. Douceur. “The sybil attack”. In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260.

[DN92] C. Dwork and M. Naor. “Pricing via processing or combatting junk mail”. In: *Annual International Cryptology Conference*. Springer. 1992, pp. 139–147.

[KN12] S. King and S. Nadal. “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake”. In: *self-published paper, August* 19 (2012).

[Lam79] L. Lamport. *Constructing digital signatures from a one-way function*. Tech. rep. Technical Report CSL-98, SRI International Palo Alto, 1979.

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

[Maz15] D. Mazieres. “The stellar consensus protocol: A federated model for internet-level consensus”. In: *Stellar Development Foundation* (2015).

[Mer87] R. C. Merkle. “A digital signature based on a conventional encryption function”. In: *Conference on the theory and application of cryptographic techniques*. Springer. 1987, pp. 369–378.

[Nak+08] S. Nakamoto et al. “Bitcoin: A peer-to-peer electronic cash system”. In: (2008).

[PSL80] M. Pease, R. Shostak, and L. Lamport. “Reaching agreement in the presence of faults”. In: *Journal of the ACM (JACM)* 27.2 (1980), pp. 228–234.

[SYB+14] D. Schwartz, N. Youngs, A. Britto, et al. “The ripple protocol consensus algorithm”. In: *Ripple Labs Inc White Paper* 5 (2014).

[SD19] A. Sonnino and G. Danezis. “SybilQuorum: Open Distributed Ledgers Through Trust Networks”. In: *arXiv preprint arXiv:1906.12237* (2019).