

Spire: A Cooperative, Phase-Symmetric Solution to Distributed Consensus

EMIL KOUTANOV 

Distributed Systems Research Division, Obsidian Dynamics, Marrickville, NSW 2204, Australia

e-mail: ek@obsidiandynamics.com

ABSTRACT All existing solutions to distributed consensus are organised around a Paxos-like structure wherein processes contend for exclusive leadership in one phase, and then either use their dominant position to propose a value in the next phase or elect an alternate leader. This approach may be characterised as adversarial and phase-asymmetric, requiring distinct message schemas and process behaviours for each phase. In over three decades of research, no algorithm has diverged from this basic model, alluding to it perhaps being the only viable solution to consensus. This paper presents a new consensus algorithm named Spire, characterised by a phase-symmetric, cooperative structure. Processes do not contend for leadership; instead, they collude to iteratively establish a dominant value and may do so concurrently without conflicting. Each successive iteration is structured identically to the previous, employing the same messages and invoking the same behaviour. By these characteristics, Spire buckles the trend in protocol design, proving that at least two disjoint cardinal solutions to consensus exist. The resulting phase symmetry halves the number of distinct messages and behaviours, offering a clear intuition and an approachable foundation for learning consensus and building practical systems.

INDEX TERMS Atomic broadcast, consensus, distributed algorithms, fault-tolerance, replication.

I. INTRODUCTION

The seminal works “*The Part-Time Parliament*” [1] and “*Viewstamped Replication*” [2], have exposed viable solutions for achieving distributed consensus under asynchronous network assumptions in ensemble sizes $n = 2f + 1$, where f is the number of tolerated failures. Paxos, in particular, has catalysed an era of research into consensus, resulting in numerous adaptations of the original protocol, as well as other protocols that serve a similar purpose.


Despite some stated differences, these protocols have a remarkably great deal in common. Evidently, consensus with $2f + 1$ processes and conflicting proposals requires two phases to achieve in the best-case scenario. This is true of all protocols considered here: Paxos [1] (and its known variations, including Mencius [7] and Ring Paxos [8]), Viewstamped Replication (VR) [2], [3], Chandra-Toueg (C-T) [4], Zab [5] and Raft [6].

The two-phase limitation is intrinsic to the problem [9]. Given an $n = 2f + 1$ ensemble size and unprivileged processes, any attempt at securing an agreement in one round of messages is futile. (If one process is privileged, then it may

achieve consensus in one round [1], [9].) Every quorum must overlap by at least one process while allowing for f failures. Suppose that some value v has been chosen in a single round of messages with the support of $f + 1$ processes, while the remaining processes supported u . Intuitively, should just one of the v -supporting processes fail, we are left with ambiguity as to the chosen value. We cannot alter the quorum size without either reducing f or foregoing the quorum intersection property; therefore, our only remaining course of action is to increase the number of messages.

Accepting two phases as the lower bound [9], we explore other similarities. The most notable is the adversarial nature of these protocols concerning process exclusion.

To eliminate the ambiguity resulting from process failures, these protocols operate by first agreeing on some intermediate quantity, before ultimately agreeing on the final value. This intermediate quantity is the identity of the dominant proposer (leader). Once a leader is established in some phase, processes in the subsequent phase seek to either assign the value by the advantage acquired earlier (if they happen to be the leader) or to ‘dethrone’ the leader and elect another (typically themselves), in the hope to assign some value later. The new leader is constrained in the values it may propose; it must cede to a prior value if there is a chance that one

The associate editor coordinating the review of this manuscript and approving it for publication was Taehong Kim .

may have been chosen. Precisely how values are prioritised is protocol specific.

We remark on another similarity: all protocols exhibit an *asymmetric phase structure*, wherein phases alternate and successive phases employ different message schemas and invoke disparate behaviour in the cohorts. For example, the Paxos *phase 1a/1b* message pair differs from the *phase 2a/2b* pair [1]. Correspondingly, the proposer and acceptor behaviours vary between the phases. Similarly, VR view change messages differ from the replication messages, as does the behaviour of the processes [3]. Likewise, Zab [5] and Raft [6] distinguish between leader election and commit messages, and corresponding behaviours. And while VR, Zab and Raft are multi-value protocols for atomic broadcast and state machine replication, their reduction to single-value consensus maintains this distinction.

Further insight suggests that the protocols' asymmetry is a consequence of their adversarial nature. Since proposers compete in establishing dominance in one phase, then propose a value in the next, it is natural that the phase behaviours and the corresponding message structures are distinct.

Indeed, existing protocols are remarkably similar in the ways that count. Paxos and VR have been labelled as "*the same algorithm independently invented*" and "*equivalent*" in [10], [11]. Liskov, in *Viewstamped Replication Revisited* [3], notes that "*VR was originally developed ... at about the same time as Paxos, but without knowledge of that work.*" Likewise, Burrows [12] refers to them as the "*same protocol*" and asserts that "*all working protocols ... we have so far encountered have Paxos at their core.*"

In comparing protocols, van Renesse *et al.* state that Paxos, VR and Zab "*seem to rely on many of the same principles*" [13]. They suggest that C-T is a refinement of Paxos.

In comparing Paxos and Raft, Wang *et al.* label their differences as "*superficial*" [14] and suggest a refinement mapping from Raft to Paxos—showing that they are materially equivalent. van Renesse and Altinbükten [15] also remark on their similarities, noting that Raft favours simplicity.

The claims in [10]–[15] are harmonious with our conclusion that the *similarities between the protocols are due to their adversarial nature*. There are only so many ways one can isolate a dominant process in one phase to propose a value in another. Anecdotaly, Mike Burrows—a distinguished computer scientist and the designer of the Chubby Lock Service at Google—is quoted in [16] as having said: "*In my experience, all distributed algorithms are either: 1) Paxos, 2) Paxos with extra cruft, or 3) broken.*" Until only recently, we would have agreed.

A. PARALLELS OUTSIDE OF CONSENSUS

As a brief detour, we remark on the protocols in the *atomic commitment* domain, as they have been loosely equated to distributed consensus and atomic commit in literature [27], [32]. Not all commitment protocols can be precisely reduced to consensus, as they cover a different functional

scope and offer different guarantees, but nonetheless display similar behaviours.

Two-Phase Commit (2PC) [25], [26] is the mainstay of atomic commitment protocols that has received much attention and numerous optimisations [28]–[31]. The protocol is asymmetrically structured, comprising *voting* and *decision* phases. A single encumbered coordinator is permitted to execute the protocol over a set of resource managers. 2PC has no mechanism for replacing coordinators, leading to the criticism of it blocking indefinitely if the coordinator is unavailable. Notably, in a comparison to consensus-based commitment, Lamport concludes 2PC to be a degenerate case of Paxos Commit with one coordinator [27].

Three-Phase Commit (3PC) [32] has attempted to address 2PC's main deficiency by adding a coordinator election step, while preserving the asymmetric phase structure. Keidar and Dolev have shown 3PC to block after carefully chosen network partition and merge steps [34] and devised E3PC, which uses view-based exclusive coordinator election.

In Alvin [24], Turcu *et al.* outline a commitment protocol comprising a concurrency control layer on top of partial order broadcast, using rotating leaders over a sequence of delivery slots, not dissimilar to Mencius [7]. Each slot has an exclusive leader; its failure requires an election before a value may be proposed in that slot, revealing Alvin's adversarial nature. Alvin is phase-asymmetric, operating over two distinct phases—*proposal* and *decision*.

Atomic commitment protocols have perceptibly evolved in the same vein as their consensus and atomic broadcast counterparts, relying on process exclusivity and multiple distinct phases to attain a commit/abort agreement over a set of resource managers, and in some cases [24], on the deterministic ordering of transactions in a log.

B. SUMMARY OF CONTRIBUTIONS

The main contribution of this paper is an *f*-fault tolerant consensus protocol named **Spire** that exhibits **cooperative value selection** within a **symmetric phase structure**. Rather than fencing each other off, processes may collude to iteratively establish a dominant value. There is an element of contention remaining, as the selection of one value invariably leads to the rejection of others. Nonetheless, *multiple processes may concurrently propose the same value without conflicting. Each successive iteration is structured identically to the previous, employing the same messages and invoking the same behaviour.* By these characteristics, Spire is a material departure from the consensus protocol status quo.

Spire achieves two objectives. Firstly, it reveals a method for solving consensus that is not predicated on process exclusivity, being the first solution in over three decades to do so. It offers an entirely different avenue for further exploration. Considering the volume of academic and industry research catalysed by Paxos and VR, we are hopeful that Spire may play a modest role to a similar effect.

Secondly, Spire's symmetric phase structure requires just two algorithms: one for proposers and one for consenters,

for all phases. This is at most half of what is required by other protocols. Its symmetry may lighten the cognitive load when learning the algorithm or implementing it, reducing the likelihood of software defects. Take Paxos: it has been labelled as difficult to understand [6], leading to much follow up work to paint it in simpler terms [6], [15], [17], [18]. This may be due to its asymmetry and a lack of a clear intuition for why it works [6]. We are confident that Spire's symmetry will greatly aid its absorption.

The rest of the paper describes Spire in detail, with emphasis on the single-value algorithm. We present several examples and follow with a formal proof sketch of its correctness. A TLA⁺ specification with bounded models for safety and liveness checking is included in the supplementary material, accompanied by a rigorous machine-verifiable proof of safety, written in TLAPS. Spire is safe under asynchrony and live under weak synchrony assumptions. To complement the formal narrative, we offer a *plain-English* intuition of why the protocol works.

While a multi-value consensus protocol is more useful from a developer standpoint, we place less emphasis on its elaboration. There are numerous ways such protocols may be derived from single-value instances [1], [7], [9]. For completeness, we offer a reference protocol named **Spanning Privilege**, complete with a TLA⁺ specification.

II. DESIGN

A. MODEL ASSUMPTIONS

We consider a distributed system comprising the set of **proposer** processes $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$ and **consenter** processes $\mathcal{C} = \{c_1, c_2, \dots, c_n\}$, which are prone to failures and communicate asynchronously by message passing in a non-Byzantine environment. $|\mathcal{C}| \geq 2f + 1$, where f is the upper bound on the number of consenter failures. Proposers and consenters may be collocated. To elaborate:

When operational, the network permits bidirectional communication between any process in \mathcal{P} and any process in \mathcal{C} . Network links may fail at any time and the network may be partitioned arbitrarily. Messages may be lost, duplicated, delayed for arbitrary periods, and delivered out of order; however, they cannot be undetectably corrupted.

A process may halt or fail sporadically; however, when operational, its behaviour conforms to the protocol.

Consenters have access to non-volatile storage, and persisted data survive the failure of the attached process.

The protocol assumes a non-empty quorum system \mathcal{Q} , where every quorum Q in \mathcal{Q} is a subset of \mathcal{C} and intersects with every quorum R in \mathcal{Q} by at least one consenter. This is referred to as the *quorum intersection property*. Formally, $\mathcal{Q} \subseteq \wp(\mathcal{C})$ and $\forall Q_1, Q_2 \in \mathcal{Q} : Q_1 \cap Q_2 \neq \emptyset$.

B. PROTOCOL GUARANTEES

The formal guarantees offered by Spire are generally accepted as necessary and sufficient for single-value

consensus [9] and are analogous to the other protocols considered.

- **Validity:** The chosen command must have been offered by at least one process.
- **Agreement:** Once v is chosen by some process, no process may choose a value other than v .
- **Termination**(with high probability): A value will eventually be chosen, provided that $n-f$ consenters and the corresponding links are nonfaulty.

Vis-à-vis *termination*, the asynchronous model is augmented with timeouts to account for the FLP result [19].

C. DEFINING CHARACTERISTICS

We briefly explore Spire's defining characteristics before elaborating on the protocol.

1) PHASE SYMMETRY

Spire requires two rounds of messages to reach a consensus in the best case. There is no distinction in the message structure nor the behaviour of successive **rounds**. A proposer does not refer to its state in the previous round when sending messages in the next. It is more instructive to think of Spire phases as **iterations**—even if this distinction is purely nominal. The sole difference between successive iterations is the exchanged payload and the updated state of the consenters as a result.

2) COOPERATIVE AGREEMENT

Existing protocols use **ballots** (Paxos [1]), **epochs** (VR [3] and Zab [5]) and **terms** (Raft [6]) for what is essentially the same construct—a logical boundary between successive attempts at isolating a dominant process from which a unique value may be safely proposed, as well as a means of propagating previous values. Existing protocols do not impose a gap-free constraint on these ordinals, only that they are *totally ordered*. Paxos additionally requires that ballot numbers are *unique*—two proposers may not use the same ballot number for different values [1], nor is a proposer allowed to recycle a ballot number.

A Spire round cardinality differs from the aforementioned ballot/epoch/term ordinals. Rounds are not used to establish the identity of the dominant proposer; in fact, Spire has no concept of proposer dominance. Proposers are entirely ephemeral, with no stable identity or persistent state. Proposers operate on an equal footing at all times. They converge on a dominant value, which is the main point of contention within the protocol. Multiple proposers may back the same value; that value may be concurrently nominated by multiple proposers in the same round.

Spire rounds are *totally ordered* but not unique—multiple proposers may use and recycle the same round numbers. Rounds are *gap-free* by an invariant of the protocol. *A value may not be offered in a round that is more than one round ahead of the highest round accepted by any quorum of consenters.*

D. THE BASIC PROTOCOL

1) MESSAGE STRUCTURE

Proposers in \mathcal{P} exchange messages with consenters in \mathcal{C} . A message from a proposer is called an **offer**:

$$\langle \text{Offer } \textit{round}, \textit{val}, \textit{primed} \rangle$$

where *round* is a natural number inclusive of zero, *val* is the proposed value in *round*, and *primed* is a Boolean—*TRUE* iff (if and only if) *val* was dominant in the previous round. A value might represent a system-wide configuration or a command that is replicated within a state machine.

A response from a concenter to the soliciting proposer is called an **answer**:

$$\langle \text{Answer } \textit{cons}, \textit{lastRound}, \textit{lastVal}, \textit{lastPrimed} \rangle$$

where *cons* identifies the responder, *lastRound* is the highest round number that *cons* accepted, *lastVal* is the value accepted in *lastRound*, and *lastPrimed* is *TRUE* iff the value accepted in *lastRound* was primed. That is, *lastRound*, *lastVal* and *lastPrimed* mirror the *round*, *val* and *primed* attributes of the highest round offer accepted by *cons*.

2) PROPOSER BEHAVIOUR

The proposer algorithm, depicted in pseudocode, comprises two parts: the initialisation routine (rule 0 in Alg. 1) and the response handler (rules 1 – 4 in Alg. 1):

Algorithm 1 Proposer Behaviour

```

1: upon initialisation:
   // rule 0: offer an arbitrary unprimed value on startup
2: let  $v_n = \text{CandidateValue}()$ 
3: Send((Offer 0,  $v_n$ , FALSE))
-----
4: upon receiving a set of answers from a complete quorum:
5: let  $A = \text{ReceiveQuorumAnswers}()$ 
6: if AllIdenticalRounds(A) then
7:   if AllPrimed(A) then
     // rule 1: uniform answers that are all primed
8:     terminate with PickValue(A) as the chosen value
9:   else if AllIdenticalValues(A) then
     // rule 2: uniform answers that are not all primed
10:    Send((Offer PickRound(A) + 1, PickValue(A), TRUE))
11:  else
     // rule 3: same-round answers with non-identical values
12:    Send((Offer PickRound(A) + 1, SuccessorValue(A),
      FALSE))
13:  end if
14: else
     // rule 4: mixed-round answers
15:  Send((Offer MaxLastRound(A), SuccessorValue(A),
      FALSE))
16: end if

```

We define the following operators (in TLA⁺ syntax):

Send(*m*) sends message *m* to an arbitrary quorum of consenters. A different quorum may be chosen upon successive invocations of **Send**(*m*).

AllIdenticalRounds(*A*) is *TRUE* iff all answers in *A* are of the same round:

$$\begin{aligned} \text{AllIdenticalRounds}(A) &\triangleq \\ &\neg \exists m_1, m_2 \in A : m_1.\textit{lastRound} \neq m_2.\textit{lastRound} \end{aligned} \quad (1)$$

AllIdenticalValues(*A*) is *TRUE* iff all answers in *A* have the same value:

$$\begin{aligned} \text{AllIdenticalValues}(A) &\triangleq \\ &\neg \exists m_1, m_2 \in A : m_1.\textit{lastVal} \neq m_2.\textit{lastVal} \end{aligned} \quad (2)$$

AllPrimed(*A*) is *TRUE* iff all answers in *A* are primed:

$$\text{AllPrimed}(A) \triangleq \forall m \in A : m.\textit{lastPrimed} \quad (3)$$

PickRound(*A*) returns an arbitrary round number among the *lastRound* attributes in *A*. **PickRound**(*A*) is only used where Alg. 1 determines that **AllIdenticalRounds**(*A*) is *TRUE*, hence there is only one round number to pick from. I.e., it is always used deterministically.

PickValue(*A*) returns an arbitrary value among the *lastVal* attributes from the answers in *A*. **PickValue**(*A*) is used deterministically everywhere except in **SuccessorValue**(*A*).

MaxLastRound(*A*) returns the highest round number among the *lastRound* attributes from the answers in *A*:

$$\begin{aligned} \text{MaxLastRound}(A) &\triangleq \\ \text{LET } \textit{SetMax}(S) &\triangleq \text{CHOOSE } t \in S : \forall s \in S : t \geq s \\ \text{IN } \textit{SetMax}(\{m.\textit{lastRound} : m \in A\}) \end{aligned} \quad (4)$$

SuccessorValue(*A*) starts by filtering a subset of answers in *A* whose *lastRound* = **MaxLastRound**(*A*). Let this result be *highestRoundAnswers*. If a primed answer is located among *highestRoundAnswers*, that answer's *lastVal* is returned. Otherwise, **SuccessorValue** returns an arbitrary *lastVal* among the answers in *highestRoundAnswers*.

$$\begin{aligned} \text{SuccessorValue}(A) &\triangleq \\ \text{LET } \textit{highestRound} &\triangleq \text{MaxLastRound}(A) \\ \textit{highestRoundAnswers} &\triangleq \\ \{m \in A : m.\textit{lastRound} = \textit{highestRound}\} \\ \textit{highestRoundPrimedAnswers} &\triangleq \\ \{m \in \textit{highestRoundAnswers} : m.\textit{lastPrimed}\} \\ \text{IN IF } \textit{highestRoundPrimedAnswers} \neq \{\} &\text{THEN} \\ \text{PickValue}(\textit{highestRoundPrimedAnswers}) \\ \text{ELSE} \\ \text{PickValue}(\textit{highestRoundAnswers}) \end{aligned} \quad (5)$$

A proposer presumably acts on behalf of a client that wishes to commit an arbitrary value. In turn, the client may issue its instruction over a network, or it may be collocated with the proposer. Precisely how the proposer arrives at a candidate value in **CandidateValue**() is immaterial to the specification. For a proposed p_n , the candidate value is v_n .

A proposer p_n starts by proposing v_n in round 0, sending **Offer** 0, v_n , **FALSE** to an arbitrary quorum in \mathcal{Q} .

A proposer only considers answers from a complete quorum of consenters. We refer to this as the set of **quorum-answers**, depicted by variable A in Alg. 1. By the weak synchrony assumption, if the proposer fails to gather a complete set of quorum-answers within some period, it may resend its earlier offer or simply start again at rule 0. As a practical consideration, $Send(m)$ may broadcast offers to multiple quorums simultaneously, in the anticipation that some consenters may fail to respond in time. This increases the likelihood of gathering a complete quorum of answers at the expense of additional network traffic.

As messages may arrive late or out of order, the set of quorum-answers may not correspond to the most recent offer: the answers could reflect an earlier offer. As such, a proposer considers the answers independently of its offer.

We use the term **uniform answers** to denote a set of quorum-answers A such that $AllIdenticalRounds(A) \wedge AllIdenticalValues(A)$.

The meaning of the term **primed** is contextual. In the case of an offer, it matches $\langle Offer _, _, TRUE \rangle$. In the case of an answer: $\langle Answer _, _, _, TRUE \rangle$. For brevity, we use the notation g' to signify that some value g is primed.

3) CONSENTER BEHAVIOUR

A conserter satisfies two obligations affecting safety:

- **S1**: Accept at most one offer in any round.
- **S2**: Persist the most recent answer before replying.

A naive implementation might satisfy $S1$ by persisting a bitmap that tracks, for each round, whether an offer was accepted. This is suboptimal, as the size of the bitmap is variable and theoretically unbounded. An optimal implementation only accepts an offer if its round number r is greater than $lastRound$ persisted by $S2$. By strict monotonic assignment $lastRound \leftarrow r$, no previous offer could have been accepted in round s , where $s \geq r$, and precludes future offers from being accepted in round q , where $q \leq r$.

Consenters must also satisfy certain liveness properties to ensure eventual termination:

- **L1**: Always accept an offer in a round, unless an offer has been accepted in the same or higher round.
- **L2**: Always respond to an offer with the last recorded answer, irrespective of whether the offer has been accepted or not.

Note: Accepting an offer can be equated to voting, as it is commonly referred to in literature. We may use the term **vote** interchangeably with acceptance of offers.

As a consequence of the obligations above, the conserter behaviour reduces to the following:

In Alg. 2, -1 is a special round number assigned to $lastRound$ if no previous answer was persisted. Since $off.lastRound$ is a natural number inclusive of zero, every uninitialised conserter will respond to the first offer. Pragmatically, initialising variables with -1 simplifies the implementation when using signed integers. $None$ is a special symbol that denotes the absence of a value, equivalent to a

Algorithm 2 Consenter Behaviour

```

1: upon receiving an offer:
2: let lastRound = persisted last accepted round of  $c_n$ ,
   initialised to  $-1$  if nothing was persisted
3: let lastVal = persisted last accepted value of  $c_n$ ,
   initialised to  $None$ 
4: let lastPrimed = persisted last primed status of  $c_n$ ,
   initialised to  $FALSE$ 
5: let off = ReceiveOffer()
6: if off.lastRound > lastRound then
   // only accept offers in higher rounds
7:   lastRound  $\leftarrow$  off.lastRound
8:   lastVal  $\leftarrow$  off.lastVal
9:   lastPrimed  $\leftarrow$  off.lastPrimed
10:   Persist(lastRound, lastVal, lastPrimed)
11: end if
12: Reply((Answer  $c_n$ , lastRound, lastVal, lastPrimed))
    
```

$null$ reference in a prospective implementation. In practice, however, $lastVal$ and $lastPrimed$ may be initialised to any value, as they will be overwritten by the first offer, but we felt the use of $None$ was more instructive.

E. EXAMPLES

It is easy to get lost in the dry formalism, yet it is necessary to state the protocol unambiguously and without imposing undue restrictions upon the implementer. We now present some examples. Assume 5 consenters $\{c_1, \dots, c_5\}$, up to 3 proposers $\{p_1, p_2, p_3\}$ and up to 3 distinct values $\{a, b, c\}$. We visualise conserter states by arranging their answer history in a matrix, where the row corresponds to the conserter and the column to the round number. We refer to rules 0 to 4 of Alg. 1 and obligations $S1, S2, L1$ and $L2$.

1) EXAMPLE 1: CLEAN RUN WITH NO FAILURES

Proposer p_1 offers a in round 0 to the quorum $\{c_1, c_2, c_3\}$ by rule 0. p_2 offers b to $\{c_3, c_4, c_5\}$. p_1 secures a majority of votes, beating p_2 to c_3 . Later, c_3 rejects p_2 's offer by obligation $S1$. The conserter states after round 0 are shown in Fig. 1 (b).

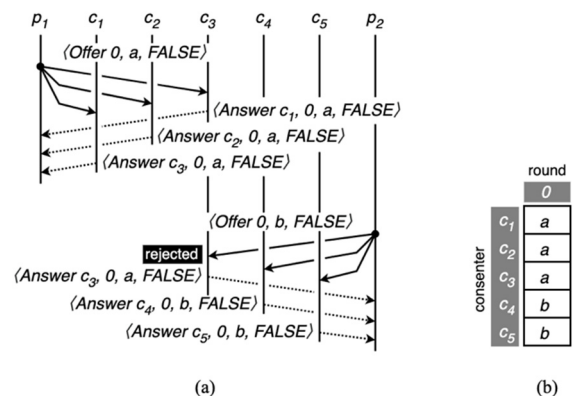


FIGURE 1. Example 1: round 0 message exchanges.

Although c_3 voted for a in round 0, it responds to p_2 by obligation $L2$: it echoes the earlier answer persisted from p_1 by obligation $S2$. p_1 receives a complete set of quorum-answers, observes that the answers are uniform and sends $\langle Offer\ a,\ 1,\ TRUE \rangle$ by rule 2 to $\{c_1, c_2, c_3\}$ in Fig. 2 (a). p_2 observes a non-uniform set of answers comprising values $\{a, b\}$, and offers an arbitrary successor value among $\{a, b\}$ by rule 3. In Fig. 2 (a), p_2 sends $\langle Offer\ b,\ 1,\ FALSE \rangle$ to $\{c_3, c_4, c_5\}$.

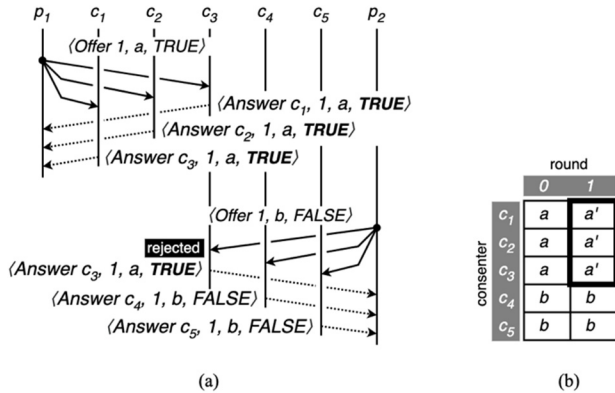


FIGURE 2. Example 1: round 1 message exchanges.

Once again, p_1 beats p_2 . Upon receiving a complete set of quorum-answers, p_1 observes that the answers are both *uniform* and *fully primed*, satisfying rule 1: a is deemed chosen, and the protocol terminates for p_1 . See Fig. 2 (b).

p_2 's set of quorum-answers contains non-identical values in the same round, satisfying rule 3. p_2 selects a successor value from the set $\{a', b\}$. By the definition of $SuccessorValue(A)$, preference is given to the primed subset. Therefore, p_2 must carry a into the next round. It sends $\langle Offer\ 2,\ a,\ FALSE \rangle$ to $\{c_3, c_4, c_5\}$ in Fig. 3.

		round		
		0	1	2
consenter	c_1	a	a'	
	c_2	a	a'	
	c_3	a	a'	a
	c_4	b	b	a
	c_5	b	b	a

FIGURE 3. Example 1: consenter states after round 2.

c_3, c_4 and c_5 accept a in round 2 and respond. Upon receiving the quorum-answers, p_2 observes a uniform response and sends $\langle Offer\ 3,\ a,\ TRUE \rangle$ by rule 2. See Fig. 4.

		round			
		0	1	2	3
consenter	c_1	a	a'		
	c_2	a	a'		
	c_3	a	a'	a	a'
	c_4	b	b	a	a'
	c_5	b	b	a	a'

FIGURE 4. Example 1: consenter states after round 3.

c_3, c_4 and c_5 accept a' in round 3. The set of quorum-answers is both *uniform* and *fully primed*. The protocol terminates for p_2 by rule 1. Both p_1 and p_2 have converged on a .

2) EXAMPLE 2: A VOTING RACE

This is a minor variation of *Example 1*. Rather than allowing p_1 to secure a vote for a' in round 1, we let p_2 beat p_1 in its race to obtain the majority of votes. See Fig. 5.

		round	
		0	1
consenter	c_1	a	a'
	c_2	a	a'
	c_3	a	b
	c_4	b	b
	c_5	b	b

FIGURE 5. Example 2: consenter states after round 1.

At this point, assuming no failures, two things will happen:

- p_1 will offer a in round 2 by rule 3 and the definition of $SuccessorValue(A)$. See Fig. 6 (a).
- p_2 will offer b' in round 2, by rule 2. See Fig. 6 (b).

Either p_1 or p_2 will secure a majority in round 2. In the former, p_1 follows with a' and, provided p_1 again secures a majority,¹ the protocol terminates after round 3 for p_1 with a as the chosen value. In the latter, the protocol terminates after round 2 for p_2 with b .

		round					
		0	1	2	3	4	5
consenter	c_1	a	a'	a	a'		
	c_2	a	a'	a	a'		
	c_3	a	b	a	a'	a	a'
	c_4	b	b	b'	b	a	a'
	c_5	b	b	b'	b	a	a'

		round				
		0	1	2	3	4
consenter	c_1	a	a'	a	b	b'
	c_2	a	a'	a	b	b'
	c_3	a	b	b'	b	b'
	c_4	b	b	b'		
	c_5	b	b	b'		

FIGURE 6. Example 2: some of the possible outcomes after round 2.

Observe how both scenarios play out. p_1 and p_2 interact with different quorums; even though the protocol terminates for one of them, the other carries on—unaware that a value has already been chosen. Due to the *quorum intersection property*, the lagging proposer observes and prioritises the primed value offered by its peer, carrying it to the next round, until it eventually receives a fully primed response.

3) EXAMPLE 3: PROPOSER FAILURE

We now observe what happens when proposers fail amidst sending an offer. We start with p_1 and p_2 , mimicking *Example 1*. Consider Fig. 7.

Here, p_1 halts after round 1, while p_2 falls short of completing round 1—having sent some messages, but not others.

¹ Voting races may recur indefinitely; thus, the algorithm is not guaranteed to terminate in a bounded number of rounds.

		round	
		0	1
consenter	c_1	a	a'
	c_2	a	a'
	c_3	a	a'
	c_4	b	b
	c_5	b	

FIGURE 7. Example 3: consenter states after round 1.

Later, a new proposer, p_3 , decides to initiate the protocol with value c . It starts by sending $\langle Offer\ 0, c, FALSE \rangle$ by rule 0. Assume that p_3 selects the quorum $\{c_3, c_4, c_5\}$. It will receive responses $\{a', b\}$ in round 1 from $\{c_3, c_4\}$, and b in round 0 from c_5 . Due to its quorum selection, it fails to observe that a has already been chosen in round 1.

A mixed-round set of quorum-answers triggers rule 4. By the definition of $SuccessorValue(A)$, round 1 is the highest and the successor must be among $\{a', b\}$. Primed values take precedence, hence the successor value is a . By rule 4, the successor round will be the highest among the received answers—round 1. p_3 will hence send $\langle Offer\ 1, a, FALSE \rangle$.

		round			
		0	1	2	3
consenter	c_1	a	a'		
	c_2	a	a'		
	c_3	a	a'	a	a'
	c_4	b	b	a	a'
	c_5	b	a	a	a'

FIGURE 8. Example 3: consenter states after round 3.

Round 2 is completed by rule 3. (See Fig. 8.) Selecting the primed answer in the same-round case is imperative: if there is even a remote chance that v was chosen in round r , then v must be propagated to $r + 1$. The protocol terminates for p_3 by the conclusion of round 3, where a' is offered by rule 2.

On Consenter Failures: We could have presented examples with consenter failures; however, they are largely superfluous as the failure of a consenter c is equivalent to a selection of a quorum that excludes c . Example 3 demonstrates what happens when p_3 sends an offer to a quorum that excludes c_1 and c_2 —it fails to observe the chosen value initially, but eventually converges on that value all the same.

F. PROPOSER RECOVERY

In all admissible behaviours beyond round 0, proposers form their subsequent offers purely from the set of answers received from some quorum of consenter, with no regard to the offer that solicited those answers. We casually refer to this trait as **proposer amnesia**. It relieves the proposer from having to maintain state and simplifies recovery. In this regard, the *proposer algorithm is entirely unimodal*—it does not distinguish between routine operation and post-failure recovery. A proposer always starts with a round-0 offer (rule 0). Having received a set of answers from some quorum of consenter,

it either learns the chosen value or resumes the protocol by offering a successor value.

The answers might not represent the most recent consenter states, as messages may be delivered out of order by the model assumptions. If so, a proposer will vacuously follow with a lapsed round number, resulting in the rejection of the offer by some consenter. Those consenter will relay the most recent answer by obligation L2. If the proposer fails to receive the set of quorum-answers in time (as messages may be delayed), it restarts at rule 0. Eventually, the proposer will discover the most recent quorum-answers and either terminate or resume the protocol.

G. LEARNING THE CHOSEN VALUE

A proposer can learn the chosen value without requiring a new message type. It may offer an arbitrary value or a *no-op* if it does not have a suitable value to propose. In practice, offering a value merely to learn the existence of a potentially chosen value may not be desirable, as it may compel the consenter into accepting the value by obligation L1.

Where discovering a value should not incur a state change, we propose a simple *Query* message:

$$\langle Query\ nonce \rangle$$

where *nonce* is some globally unique identifier used for request-response pair correlation.

Sent to a quorum of consenter, a *Query* solicits a *Query-Answer* response. This is an extension of *Answer* that permits -1 and *None* in its *lastRound* and *lastVal* attributes, respectively, and mirrors the initiator's *nonce* attribute:

$$\langle Query-Answer\ cons, lastRound, lastVal, lastPrimed, nonce \rangle$$

Upon receiving a set of query-answers, the proposer will verify that the nonces match—signifying that the response is to its last query and not some earlier query that may have been delayed. Thereafter—

- If the responses are uniform and primed for a complete quorum, it will learn the chosen value by rule 1 of Alg. 1; or
- Else, if the responses contain a primed value, it will resume the protocol, inducing value selection; or
- Else, it may safely conclude that no value has been chosen up to this point.

III. AN INTUITIVE PROOF

Before presenting a formal proof sketch of correctness, we offer a *plain-English* account, revealing the intuition behind Spire. This is done in the spirit of making consensus more approachable—a theme popularised by Raft [6].

The intuition is tied to the *quorum intersection property*, the *singularity of votes* (obligation S1) and the propagation of primed values (by the definition of *SuccessorValues*). Assume v is primed in r , as shown in Fig. 9.

		round						
		...	$r-1$	r	$r+1$	$r+2$...	t
consenter	c_1		u	u				
	c_2		w	w				
	c_3		v	v'				
	c_4		v	?				
	c_5		v	?				

FIGURE 9. Consenter states after round r .

Because consenter may only vote once in any given round by obligation SI , we know that there can be at most one dominant value accepted in $r-1$, being v ; therefore, every primed value offered in r is v . Because quorums overlap, the acceptance of a primed value by some quorum in round r must spill onto all other quorums in r . By carrying v into $r+1$ in unprimed form (rule 3 of Alg. 1), we honour a hypothetical prior agreement in r . But how can we tell if an agreement really took place?

The answer is: we cannot. Even if one consenter is unreachable, the resulting ambiguity may prohibit a conclusive answer. However, we can be sure that if a value was chosen in r , then it was v . So, we act conservatively. If no agreement took place, then another proposer might pick a different quorum void of a primed answer, and other values might be carried into $r+1$. E.g., u and w in Fig. 10.

		round						
		...	$r-1$	r	$r+1$	$r+2$...	t
consenter	c_1		u	u	u			
	c_2		w	w	w			
	c_3		v	v'	w	w'		
	c_4		v	w	w	w'		
	c_5		v	u	v	w'		

FIGURE 10. Value v was not chosen in round r .

However, if the agreement did take place, then every set of quorum-answers in r will contain at least one primed value: v ; ergo, only v will make it to $r+1$. See Fig. 11.

		round						
		...	$r-1$	r	$r+1$	$r+2$...	t
consenter	c_1		u	u	v		...	
	c_2		w	w	v	v'	...	
	c_3		v	v'	v	v'	...	v'
	c_4		v	v'	v	v'	...	v'
	c_5		v	v'	v	v'	...	v'

FIGURE 11. Value v was chosen in round r .

For agreement to hold, we need to be sure that the chosen value will persist indefinitely. It is not enough to say that only v will be carried to $r+1$; we need a stronger claim that if a value is chosen again in some future round t , where $t \geq r$, then that value is v . For $t = r$, only v may be chosen in t by the quorum intersection property. For $t = r+1$, we have just shown that only v may be offered in t ; thus, only v may be chosen in t . For $t > r+1$, by Alg. 1, the values offered in

every non-zero round s are confined to the values answered in $s-1$, which, in turn, must have been offered in $s-1$ (by Alg. 2). Thus, if only v was offered in $r+1$, then every offer in all subsequent rounds ($r+2, r+3, \dots, t$) contains v .² If a value is chosen again in t , then it must be v .

IV. SKETCHED PROOF OF CORRECTNESS

The structure of the following proof sketch has been adapted from the TLAPS machine-verifiable proof of safety. (See *Supplementary Material*.)

Lemma 1: *If a primed answer exists in r containing a value v , then every primed answer in r contains v .*

Proof: If v is primed in r , then v was the dominant answer in $r-1$. By SI (consenter votes at most once per round) and the quorum intersection property, there may be at most one dominant answer in $r-1$. Thus, any primed offer in r must carry v by rule 2 of Alg. 1. An answer in r may only be in response to some offer in r (by Alg. 2), thus every primed answer in r must carry v . \square

Lemma 2: *If a value v was chosen in r , then for all sets of quorum-answers in r , at least one answer in each set must contain v primed.*

Proof: The choice of v in r implies a quorum of consenter in r voting for v primed. By the quorum intersection property, every set of quorum-answers in r must overlap with every other set in r by at least one answer. \square

Lemma 3: *If a value v was chosen in r , then only v may be offered in $r+1$.*

Proof: Given a set of quorum-answers A in r and earlier, the only offers in $r+1$ occur by rules 2 and 3 of Alg. 1.

In rule 2, $AllIdenticalRounds(A) \wedge AllIdenticalValues(A)$ holds. By Lemma 2, at least one answer must contain v primed, and by $AllIdenticalValues(A)$ it follows that $PickValue(A)$ is v .

In rule 3, $AllIdenticalRounds(A)$ holds. By Lemma 2, at least one answer must contain v primed. By Lemma 1, every primed answer must contain v . By $SuccessorValue(A)$, primed values take precedence; therefore, $SuccessorValue(A)$ is v . \square

Lemma 4: *If a value v was offered in a non-zero round r , then v was also offered in $r-1$.*

Proof: By Alg. 1, an offer in r may only carry a value that is either (1) sourced from an answer in $r-1$ or (2) from an answer in r .

For (1), where an offer in r carries v from an answer in $r-1$, v was offered in $r-1$ (by Alg. 2).

For (2), the proof is by finite set induction. The base case: when there are no prior answers in r , rule 4 is disabled and the first offer in r must pick a value from an answer in $r-1$. The inductive step: a new offer in r must pick a value from an answer in $r-1$, or from an answer in r , which carries from $r-1$. Therefore, all offers in r carry a value from an answer in $r-1$. And by Alg. 2, an answer in $r-1$ corresponds to an offer in $r-1$. \square

²The formal proof relies on induction to show that only v may be carried and subsequently chosen.

Lemma 5: *If a value v was chosen in r , then v was offered in r .*

Proof: A chosen value v in r is conveyed by a set of primed quorum-answers in r (rule 1 of Alg. 1). An answer in r corresponds to an offer in r by Alg. 2; therefore, v is chosen from an offer in r . \square

Theorem 1 (Agreement): *If a value v was chosen in round r , then no value other than v may be chosen in any round s , where $s \geq r$.*

Proof: For the case $s = r$, the choice of v in r implies that every set of quorum-answers in r has at least one primed answer with v by Lemma 2. Every chosen value in r is v by *SuccessorValue(A)*. For $s > r$, it suffices to show that (1) only v may be offered in $r + 1$, (2) a value offered in a non-zero round g was offered in $g - 1$, and (3) a value chosen in g was offered in g . The leap from $r + 1$ to s is by induction over naturals, with (1) as the base case and (2) as the inductive step. Obligations (1), (2) and (3) are discharged by Lemmas 3, 4 and 5, respectively. \square

V. PRACTICAL CONSIDERATIONS

A. EVENTUAL TERMINATION

The basic protocol does not guarantee termination in a bounded number of rounds. Before a value v is proposed in a primed round, the proposer(s) of v must gather support from a quorum of consenters in a previous round. It is easy to imagine a scenario where concurrent proposers offer conflicting values, resulting in a string of *split votes*, as alluded to by Example 2. Furthermore, the likelihood of a round concluding in a quorum of uniform votes diminishes with the number of distinct values and consenters. This is not dissimilar to the so-called ‘duelling proposers’ phenomenon in Paxos [35] and tied elections in Zab [5] and Raft [6].

To avoid a protracted string of rounds that result in split votes, some nondeterminism must be injected into the protocol using a *random oracle*. Specifically, a failure to acquire a quorum vote should be followed by a random delay before pitching the next offer. This is analogous to the measures proposed in [35], [5] and [6]. Furthermore, while a lower bound on the number of message delays has been shown [9], no corresponding upper bound exists, owing to the FLP result [9], [20]. Augmented with random backoff, Spire eventually terminates, provided sufficient consenters and network links are nonfaulty.

To accelerate convergence, we later propose a separate optimisation (see Section V.E), wherein values may be deterministically ranked, rather than picked arbitrarily in *SuccessorValue(A)*.

Note that although contention materially affects single-value consensus, its impact can be largely amortised over multiple values. All multi-value algorithms considered here, including Multi-Spire (see Section VI), address this through some variant of privileged commit, eliminating contention in the steady-state where proposers and network links are nonfaulty for a majority of the time.

B. LEARNER ROLE

The subject of learners has been conveniently sidestepped thus far, as it is largely immaterial to the specification. A proposer may broadcast the result discovered by rule 1 of Alg. 1 to an arbitrary group of learners. Alternatively, consenters may broadcast copies of their answers to the learners which will apply the same logic as a proposer to determine the chosen value—in effect, acting as passive proposers. Learners may be independent processes or collocated with proposers or consenters. Provided the chosen value is determined consistently by the predicate $AllIdenticalRounds(A) \wedge AllPrimed(A)$, precisely when and where this takes place is left to the implementer.

C. ROUND-ZERO PRIVILEGE

It has long been known that a protocol may be reduced to a single phase in the case where at most one proposer holds a privileged status [1], [9]. For example, a distinguished proposer in Multi-Paxos may start with a *phase 2a* message for some pre-agreed ballot b if no other proposer can issue a *phase 2a* message with ballot a , where $a < b$.

Submitting a primed offer in round r indicates the presence of a dominant value in $r - 1$. The latter ensures that at most one value may be primed in r . Ordinarily, because there is no round -1 , proposers are forbidden from submitting a primed offer in round 0.

Assume a contrived, but otherwise valid, protocol variant that includes among its assumptions some constant value z that was dominant in an *imaginary round* -1 . Any proposer may safely offer z' in round 0, terminating the protocol in a single round in the absence of conflicting unprimed offers.

In practice, it is often more useful to propose an arbitrary value in round 0. Assume that at most one proposer, p^* , is **privileged**, and every proposer knows whether or not they are privileged (but not necessarily the identity of p^*) then p^* may safely propose an arbitrary primed value in round 0 and terminate the protocol in a single round.

There is a caveat: *at most one distinct primed value may be proposed in round 0*. If a primed value z was offered in round 0, every primed value offered in round 0 must be z . We refer to this invariant as the **singularity of privilege**.

D. ORDERED VALUES

In *SuccessorValue(A)*, where $highestRoundPrimedAnswers = \{ \}$, an arbitrary value from $highestRoundAnswers$ was offered in rules 3 and 4. This introduces a degree of nondeterminism, implying that a group of proposers observing an identical set of quorum-answers may submit a different offer, reducing the likelihood of a uniform quorum vote.

In the spirit of cooperative agreement, we can do better. Rather than picking values arbitrarily, proposers are configured with some pairwise ordering relation over the domain of values; for example, lexicographical order if values are byte arrays. When offering the next value, proposers consistently

pick the highest (or lowest) value among *highestRoundAnswers*, thereby expediting convergence.

E. PRIMED CARRY

By rule 4 of Alg. 1, a proposer may safely set *primed* to *TRUE* in its following offer if the successor value came from *highestRoundPrimedAnswers*. We know this to be safe because the successor round is given by *MaxLastRound(A)*, which is identical to *highestRound* used to build *highestRoundPrimedAnswers*. Since at most one distinct value may be primed in a given round (by *Lemma 1*), then propagating the primed status in *highestRound* does not introduce a conflict of primes.

This accelerates convergence: if p_1 fails amidst offering v' in r , a surviving proposer p_2 propagates v' in r , thus increasing the likelihood of the protocol terminating in r .

F. COMPARISON WITH PAXOS

Paxos [1] is perhaps the most studied and widely deployed of distributed consensus algorithms [6], [15], [17], [18], often cited as the archetype upon which many of the other algorithms are based [10]–[15]. Paxos and Spire share identical system assumptions and satisfy the same safety and liveness criteria.

Paxos scopes a set of *proposers* that may submit potentially conflicting values to a set of *acceptors*. Proposers are *a priori* assigned an unbounded series of unique *ballot numbers* that are totally ordered. Proposers do not share ballot numbers.

The basic algorithm operates in two distinct phases. In the first, a proposer p sends a *phase 1a* (prepare) message to a quorum of acceptors, containing a ballot number n that is higher than any previous ballot used by p .

An acceptor positively replies to a *phase 1a* message with a *phase 1b* (promise) message if n is higher than any previously received ballot number. Furthermore, if a value was accepted in an earlier ballot, the acceptor's reply includes that ballot number and the value accepted in that ballot. Otherwise, if n is lower than a previous ballot, the acceptor is free to either ignore p 's *phase 1a* message, or to send back a NACK. In responding positively, however, an acceptor promises to reject all future ballots lower than n . To make its promise durable, an acceptor persists it to stable storage before responding.

Upon receiving a positive *phase 1b* response from a quorum of acceptors, p follows with a *phase 2a* (propose) message. If any of the phase 1 responders had previously accepted a value, p must propagate the accepted value with the highest ballot number among the responses; otherwise, p is free to propose its own value in phase 2.

An acceptor positively replies to a *phase 2a* message with a *phase 2b* (accepted) message if it has not already promised in a higher numbered ballot and, in doing so, accepts the proposed value. It also stably persists the accepted value and the corresponding ballot number before replying, overwriting any previously accepted value. Otherwise, the acceptor can ignore the message or reply with a NACK.

Intuitively, Paxos works by installing one of the proposers as an exclusive leader³ in phase 1, using ballot numbers as a means of ranking competing proposers. By 'voting' for a proposer, an acceptor fences proposers with lower ballot numbers. Once a quorum of acceptors has voted in a ballot, the emerging leader gains a temporary advantage over its peers—only it can propose a value in phase 2. Note, messages are structured differently across phases, as are the corresponding proposer and acceptor behaviours; this being the reason that we characterise Paxos as *phase-asymmetric*.

A lower balloted proposer may beat the leader to phase 2, only to be blocked; that proposer can subsequently repeat *phase 1a* with an even higher ballot number, possibly taking over leadership. It is easy to see how proposers might contend over leadership instead of making progress; however, once a leader gains support from a quorum in phase 2, its proposed value is chosen. At most one leader can succeed in phase 2; therefore, at most one value may be chosen. It is precisely this behaviour, wherein a single proposer ultimately decides on a value, that we label as *adversarial* in our characterisation of Paxos and its derivatives.

Phase 1 serves a dual purpose, however. In addition to installing a leader, it propagates an earlier value that may have been chosen by the actions of a previous leader—a circumstance that the new leader may not be aware of. Perhaps the leaders chose different quorums; nonetheless, due to quorum intersection, the chosen value will persist in every admissible quorum. This ensures that if a leader is ousted, the incoming leader continues from where its predecessor left off—conservatively reaffirming a value that *may* have previously been chosen. This guarantees the survival of chosen values and directly supports the *agreement* property.

The two alternate protocol families (Paxos-like vs Spire-like⁴) have distinct characteristics that may make one more suitable than the other in certain applications. The rest of this section serves as a comparative guide to assist in optimal protocol selection.

1) NON-CONFLICTING CONCURRENT PROPOSALS

When multiple processes propose values in quick succession, they may enter a parasitic cycle, whereby one process 'undoes' the effect of the other in order to make progress and terminate the protocol. In Paxos, this condition is known as *duelling proposers* [35]. Similarly, Zab [5] and Raft [6] are susceptible to tied elections. Spire exhibits an equivalent behaviour, described as a *voting race* in Example 2 (Fig. 6). This is a fundamental limitation of consensus [9] that can be parried with random backoff to induce nondeterminism and alleviate contention.

Protocols like Paxos that rely on process exclusivity may exhibit this behaviour irrespective of the choice of input

³The provision for leader exclusivity here differs from the weaker *distinguished proposer* role that may be used to ensure progress [18].

⁴*Paxonian vs Spirillian*, perhaps.

values. Even if two proposers offer *identical values*, they may still conflict in the course of establishing a dominant process.

A distinct advantage of the cooperative model is that a voting race cannot occur among concurrent proposers that offer identical values. In Spire, two proposers with identical values do not interfere, terminating the protocol in two rounds. Spire does not treat this as a special case; it is de facto catered for by rules 1 and 2 of Alg. 1.

The benefits of efficiently committing non-conflicting values become apparent when working with *a priori* constant values that convey a special meaning within some higher-level protocol, typically in state machine replication, atomic broadcast and other multi-value consensus systems.

One common scenario—Lamport’s so-called ‘ α method’ [36] of reconfiguring a distributed state machine described by a sequence of consensus-based commands—employs a special *no-op* command to fast-track configuration changes. This method is routinely applied to Multi-Paxos [1], [15] and is equally applicable to (Multi-)Spire due to its generality (Section VI). In a different scenario, Mao *et al.* present a throughput-optimised, load-balanced variant of Multi-Paxos—Mencius—that uses special *skip* commands to allow slow servers to keep up with their faster counterparts [7]. While Mencius has been implemented over Paxos, its design is largely agnostic of the underlying consensus protocol. Our own Spanning Privilege multi-value consensus (Section VI) uses *no-op* commands to expediently terminate uncommitted slots.

Cooperative consensus is not just less chatty in the presence of non-conflicting concurrent proposals, it also reduces commit latency by terminating the protocol in the optimal number of rounds (2).

2) VERBOSITY OF MESSAGE EXCHANGES

Spire’s symmetric round structure implies that all rounds carry values in both directions. By comparison, Paxos messages only carry a value in *phase 1b* and *phase 2a* messages [1], [18]; furthermore, *phase 1b* will omit a value if none had been accepted. When values grow sufficiently large, to the point where they dominate the message payload (e.g., large binary objects), Paxos is between two and four times as efficient as unoptimised Spire for network utilisation. Even when both protocols commit in privileged mode, Paxos is twice as efficient for very large messages because a *phase 2b* message does not echo the accepted value.

We now present an optimisation that addresses this shortcoming with some increase in design complexity and a minor change to message schemas.

Surrogate identifiers: For this modification, consider a proposer p , its candidate value v , consenter c and its last accepted value w . Proposer processes are uniquely identified in the system. Values are uniquely tagged on proposers with ‘surrogate’ identifiers. That is, two proposers may assign different surrogate identifiers to the same value, but any given proposer consistently maps from a value to its own surrogate identifier. The *Offer* schema is amended to include

the proposer’s process identifier and v ’s surrogate identifier $s_{p,v}$ (both as optional attributes), and the existing *val* attribute is made optional. The *Answer* schema is amended to include an optional $s_{p,w}$ surrogate identifier and the *lastVal* attribute is made optional.

An offer from p to c includes both v and $s_{p,v}$ if p never received an answer from c containing $s_{p,v}$; otherwise, p sends only $s_{p,v}$ to c . For an accepted value w , c keeps a persistent ternary mapping of w onto the associated $\langle p, s_{p,w} \rangle$ tuples, as each proposer tracks a different surrogate identifier for w .

An answer to p must contain w if c lacks a mapping from w onto $\langle p, _ \rangle$; otherwise, c replies with the resolved $s_{p,w}$.

The proposer identifier and local surrogate identifiers can be transient; if the proposer restarts, it can assume a new unique identity. Unoptimised participants may coexist with surrogate-optimised ones, provided they support the expanded message schema. (For compatibility, we treat the new attributes as optional.)

This optimisation reduces the transmission of complete values in a typical two-round message exchange by up to a factor of four, or two when used with round-zero privilege. In a routine, uncontended scenario (no voting races or failures), only the first message transports the complete value; all subsequent messages refer to its surrogate identifier.

Surrogate identifiers are not entirely free—requiring additional storage space on the consenters. Also, Paxos accords the option of using multicast or broadcast network primitives to send *phase 1a* and *2a* messages to either a specific quorum or the entire ensemble, respectively. With surrogates, Spire selectively sends either $[v$ and $s_{p,v}]$ or just $s_{p,v}$ depending on the communication history with individual consenters; Spire can multicast, but not broadcast offers in this case. Where the use of network broadcast is preferred, Paxos may be a better fit for large values.

VI. MULTI-SPIRE

When presenting a consensus algorithm, it is customary to address the replicated state machine scenario [1], [2], [5], [6]—a stable agreement over multiple values, forming a totally ordered log of commands.

The projection of a single-value protocol to a replicated log first appeared in Multi-Paxos [1]. This approach is equally viable in Spire using the *round-zero privilege* optimisation—amortising the cost of the initial phase over the extent of the log. The distinguished proposer (leader) may also piggy-back *learn* messages of prior values on successive offers—reducing the average complexity of learning a single log entry to one message exchange.

The main drawback of this approach is outlined in [21]: the loss of the leader results in a period of downtime until the next election is induced. All protocols based on exclusive leaders are susceptible to this. The system’s responsiveness is influenced by the timeout setting of the internal failure detector—a delicate balance between incorrectly suspecting a functioning leader and failing to suspect a malfunctioning leader. When a proposer suspects that a leader has failed, its

only options are to 1) wait for the leader to respond, 2) wait for another leader to take over, or 3) induce leader election. In any case, progress will stall, leading to an accumulation of backlogged commands—impacting the performance of the system for some time after a leader is reinstated.

In projecting Spire to a multi-value log, it was tempting to imitate Multi-Paxos with its trademark *distinguished proposer*. Opting for an exclusive leader would have retained the drawbacks described above. We wanted to build on the non-exclusive nature of Spire.

Instead, we propose a reference Multi-Spire design named **Spanning Privilege (SP)**. The name refers to the mechanism by which a proposer may overload an instance of single-value consensus to *commit* a command (i.e., get a command chosen), acquire or extend its round-0 privilege, and impart its status onto its peers. A privileged status acquired by p is non-binding to its peers: they may forward their commands to p but are not obligated to do so. Failure of p may lead to degraded performance but does not cause a blackout.

A. SP-1

We first present a simple, exclusive leader-based protocol named SP-1, which will be progressively generalised.

SP-1 treats a log as an unbounded contiguous sequence of single-value consensus instances, bound to **slots**, starting at slot 1. Proposers are given a stable identity, and may act as disseminators of chosen values as well as learners—broadcasting the chosen value via a $\langle Learn\ s,\ v \rangle$ message, where s is the slot number and v is the value chosen in s . Proposers and consenters may be collocated. Proposers maintain the following variables:

- θ : A locally unique value that varies with each process initialisation. θ may be a strictly monotonic timestamp, a UUID, or a persisted value that is incremented and saved when the process starts.
- **lastProposed**: The last slot number in which the process proposed a value.
- **lastChosen**: The last slot in which the chosen value was marked with the proposer’s identity and θ .

The value offered by p to a slot s is the triplet $\langle m_{p,s}, \rho_p, \theta_p \rangle$, where $m_{p,s}$ is p ’s command destined for s , ρ_p is its stable identity and θ_p is its current θ .

Committing a value containing ρ and θ is called **marking**. By marking s , p implicitly obtains the round-zero privilege in $s + 1$. By the underlying *agreement property*, at most one value may be accepted in s , implying that at most one proposer may acquire privilege over $s + 1$. Likewise, if p commits a value in $s + 1$ (by privilege or without), its privilege will be extended to $s + 2$. As long as p marks values in successive slots, its privilege will be continuously extended until its marking streak is preempted by another proposer. Other proposers learn the committed values and eventually discover the identity of the privileged proposer.

Fig. 12 illustrates the contents of a hypothetical log. Slot 1 is occupied by the command $x \leftarrow 3$, submitted by

		slot									
		1	2	3	4	5	6	7	8	...	
m		$x \leftarrow 3$	$y \leftarrow 4$	$z \leftarrow 1$	$x \leftarrow 2$	$x \leftarrow 7$	$z \leftarrow 3$	$z \leftarrow 4$			
ρ		p_1	p_1	p_1	p_2	p_2	p_1	p_1			
θ		1	1	1	1	1	2	2			
		p_1		p_1	p_1	p_2	p_2	p_1	p_1		
		privilege holder									

FIGURE 12. Sample committed values in SP-1.

p_1 with $\theta = 1$. This grants p_1 privileges in slot 2. Later, p_1 commits $y \leftarrow 4$ in slot 2, extending its privilege to slot 3. (The commands of the form $_ \leftarrow _$ are arbitrary examples, opaque to the protocol.)

At some point, p_2 suspects a failure of p_1 and preempts it by committing $x \leftarrow 2$ in slot 4, while simultaneously asserting its privilege in slot 5. Eventually, p_1 restarts and increments its θ , and later preempts p_2 in slot 6 with $z \leftarrow 3$.

When a proposer q wishes to commit some command on behalf of its client, it may forward that command to a privileged proposer p that it may be aware of. A new message is devised for this purpose: $\langle Forward\ m,\ nonce \rangle$. The response is either a $\langle Forward-ACK\ nonce \rangle$ or a $\langle Forward-NACK\ nonce \rangle$, where m is the command payload and $nonce$ is a request-response correlator. Proposers favour forwarding when possible, as serialising commands through a single proposer eliminates contention for slots and capitalises on the round-0 optimisation.

When p receives a *Forward* message from q , it must first verify its own status. As the privileged proposer is discovered by learning, it is conceivable that q might call upon a proposer that has been preempted. p only uses its privilege in $lastProposed + 1$ when $lastProposed = lastChosen$, halting until this condition is met. In practice, p will not halt indefinitely, replying with a *Forward-NACK* after a timeout or if p detects that it has been preempted. Otherwise, p commits in $lastProposed + 1$ and replies with a *Forward-ACK*. If proposers and consenters are collocated, p may also piggyback *Learn* messages on successive offers.

p may be out of reach, slow to respond, or respond with a NACK. q may either 1) retry with the current proposer, 2) look for another proposer, or 3) attempt to commit the value directly and acquire the privileged status. Assuming options (1) and (2) have been exhausted, we proceed to (3).

To commit a value with an accompanied *transfer of privilege*, q first selects an uncommitted slot, which should ideally follow the last committed slot in the log. q can conservatively discover an uncommitted slot by **a)** looking at its copy of the log, **b)** asking its peers for the highest slot number they are aware of, or **c)** querying the consenters for the highest slot number that is occupied. In (c), an occupied slot on a consenter does not imply that a value has been chosen in that slot, so q resumes the protocol at that slot.

Once a (possibly) vacant slot is established, q will propose a value in that slot. At this point, q is non-privileged; therefore, it must start with an unprimed offer. This does not imply that q succeeds: another proposer may beat q to it, perhaps

even the original proposer p that marked the previous slot. At any rate, q will learn the outcome and accept the new proposer.

SP-1 combines the notions of *privilege transition* and *value selection* into an atomic step. Moreover, there is no need to explicitly broadcast the identity of the new privileged proposer to its peers; it is disseminated through the conventional learning of values by proposers.

1) THE PURPOSE OF θ

Drawing from the example in Fig. 12, consider the following scenario. Having committed $x \leftarrow 3$ in slot 1, p_1 proposes $y \leftarrow 4$ in slot 2 and crashes immediately afterwards. Slot 2 is left in a pending state, with a primed round-0 offer from a sole proposer. Having restarted, p_1 again contends for privilege, and conservatively picks 1 as the next vacant slot. It proposes the command $x \leftarrow 3$ and succeeds, by the stable agreement retained from the previous execution.

In the absence of a distinct θ , p_1 's committed value in slot 1 is indistinguishable from its pre-crash proposal, as both the command and the identity ρ_p are unchanged. p_1 re-acquires its status in slot 1, having no recollection of its pre-crash proposal. This time, p_1 proposes some other value, $z \leftarrow 7$, in slot 2. Two distinct primed values end up occupying the same round in slot 2, violating the *singularity of privilege* invariant of Spire.

Armed with a unique θ , p_1 would have realised that its apparent success in slot 1 referred to a previous execution and would have abstained from the privilege in slot 2.

B. GENERALISATION TO MULTI-MODAL COMMIT

The main problem with protocols based on exclusive leaders is the interruption experienced during leadership transitions [21], requiring a compromise in failure detection. One either errs on the side of stability, opting for conservative timeouts, or on the side of responsiveness with more aggressive failure detection. The issue is exacerbated in WANs, with increased variability of link latencies. A system may be functioning correctly (with all processes and network links operational) while exhibiting poor performance, due to resource saturation under high load. Inducing leader election under these conditions does not remedy the situation; it compounds the problem by prolonging the outage and growing the command backlog. This phenomenon is commonly known as *congestive collapse* in networking literature [22].

Unfortunately, SP-1 relies on the leader for progress. Ideally, any proposer q should be able to expediently commit a value without preempting the encumbered privileged proposer p . Perhaps q is working to a soft deadline, and cannot wait for privilege transition, nor has it waited long enough to suspect p as failed with sufficient probability. A non-preemptive commit is achieved by q marking a message with p 's $\rho - \theta$ pair, thereby extending p 's current privilege while simultaneously getting q 's own command committed. Processes may apply varying timeouts: a longer overarching timeout for inducing privilege transitions, and

several shorter timeouts for probabilistically meeting specific deadlines. This is the **multi-modal** variant: SP-M.

Compare SP-M with exclusive leadership models, such as Multi-Paxos [1], VR [2], Zab [5] and Raft [6]: although acquiring privileged status is comparable to a leadership change, the privileged proposer here is non-exclusive. Any proposer may commit directly to the log, forgoing the benefit of uncontended commits, but avoiding a disruptive perturbation of the ensemble. A *multi-modal consensus protocol dispenses with the notion of an exclusive leader while preserving the key benefits*. When the system is stable, the use of a privileged proposer is widely beneficial. As the system exhibits instability—ranging from minor performance degradations to complete process failures—surviving processes may still commit values directly, albeit at a reduced rate due to the increased contention of non-serialised proposals. Depending on the severity or duration of the problem, processes may, at their discretion, initiate a leadership transition.

C. GENERALISATION TO SLIDING-WINDOW PRIVILEGE

A notable drawback of SP-1 is that it allows for at most one in-flight proposal. Namely, a proposer cannot exercise its privilege in a slot until it learns that it has successfully marked the previous slot. Thus, the commit rate cannot exceed the inverse of the round-trip time.

We now propose another generalisation: SP- Γ . Rather than acquiring the commit privilege over one slot at a time, a proposer is implicitly allocated a contiguous extent of Γ slots. It may concurrently submit proposals over those slots without waiting for confirmations, provided it does not wander further than Γ slots ahead of its *lastChosen* offset. Its privilege is extended as it learns that its prior marked offers were chosen, in a manner that is comparable to the *sliding window flow control* protocol [23].

SP- Γ requires several amendments to the base protocol. The image of the triplet $\langle m, \rho, \theta \rangle$ is extended so that ρ and θ allow a \emptyset (null) value. The attributes ρ and θ must be jointly null; i.e., $\rho = \emptyset \Leftrightarrow \theta = \emptyset$. A proposal with non- \emptyset ρ and θ is called **marked**, as before; conversely, a proposal where $\rho = \emptyset \wedge \theta = \emptyset$ is called **unmarked**.

		slot												
		1	2	3	4	5	6	7	8	9	10	...	$\Gamma = 3$	
m		$x \leftarrow 3$	$y \leftarrow 4$	$z \leftarrow 1$	$x \leftarrow 2$	$x \leftarrow 7$	$z \leftarrow 3$	$z \leftarrow 4$						
ρ		p_1	p_1	p_1	\emptyset	\emptyset	p_2	p_2						
θ		1	1	1	\emptyset	\emptyset	1	1						
		$p_1 \quad p_1 \quad p_1 \quad \emptyset \quad \emptyset \quad p_1 \quad p_1 \quad p_2 \quad p_2 \quad p_2 \quad p_2$												
		privilege holder												

FIGURE 13. Sample committed values in SP with $\Gamma = 3$.

The sliding-window behaviour is illustrated in Fig. 13, with $\Gamma = 3$. As per the example in Fig. 12, slot 1 is marked by p_1 . This time, however, the privilege is granted over the range 2..4. It is then extended to slots 3..5, 4..6, and so on.

Suppose p_2 suspects a failure of p_1 by slot 4. It may directly offer its command in slot 4 but p_2 cannot simply preempt p_1

as its privilege endures through to slot 6. Instead, p_2 offers null ρ and θ in slots 4..5; slot 6, being the earliest point at which *privilege transfer* may take place.

Behaviour 1. For p to exercise its privilege in s , at least one marked slot in the range $(s - \Gamma)..(s-1)$ (truncated at slot 1) must contain p 's ρ and current θ .

Behaviour 2. For q to preempt an encumbered privileged proposer p (presumably) parked at s , some proposer (possibly q) must commit a contiguous range of unmarked slots in $(s + 1)..(s + \Gamma - 1)$. Slot $s + \Gamma$ may be marked—either with q 's own ρ - θ pair, or some other proposer's, as per the SP-M generalisation. If $s + \Gamma$ is unmarked, then p will be preempted without assigning a successor.

To elaborate, q may only mark $s + \Gamma$ if it is certain that the slots in the range $s + 1$ to $s + \Gamma - 1$ are unmarked, which is trivially ascertained by learning the committed outcomes of those slots. By the *agreement property*, once a slot is committed it can never be altered; therefore, the transition protocol maintains a gap of $\Gamma - 1$ unmarked slots between any successive pair of distinctly marked slots. Once q secures slot $s + \Gamma$, it acquires privilege over the slots in the range $s + \Gamma + 1$ to $s + 2\Gamma$. No two proposers may exercise round-0 privileges in the same slot due to the separation accorded by the transition protocol.

When $\Gamma > 1$, gaps may form in the log. When a proposer offers multiple values concurrently for different slots, a slot with a higher number can be committed before a lower-numbered slot. The proposer may also fail, leaving the lower-numbered slot in an uncommitted state.

Behaviour 3. Before delivering the command in slot s , learners must await commands in all prior slots $1..(s-1)$, and deliver those commands before s .

This is accomplished by allowing a grace period before terminating the protocol for any pending slots. Termination occurs by offering a special *no-op* value in an uncommitted slot, which may be initiated by any proposer. *Concurrent no-op slot termination by multiple proposers is cheap*: Spire's cooperative nature leads to rapid convergence when the offered values do not differ.

It should be apparent that SP-1 is a special case of SP- Γ , with $\Gamma = 1$ and no unmarked values. When $\Gamma = 1$ the log is also gap-free, provided proposers conservatively select the next vacant slot and resume at a slot they suspect may be uncommitted before moving on to the next slot.

D. GENERALISATION TO ANY SINGLE-VALUE ALGORITHM

The final generalisation of SP is perhaps the most straightforward, but also the most telling. We have, thus far, worked on the assumption that the privilege acquired by p in s over slots $s + 1$ to $s + \Gamma$ applies to the round-0 optimisation of Spire, and that each discrete consensus instance executes the Spire protocol. This too can be generalised.

SP is neither coupled to the underlying single-value protocol nor requires a uniform single-value protocol across the entire log. Conceivably, one might implement SP over several dissimilar single-value protocols. Conveniently, the

concept of slot-privilege is protocol-agnostic: Lamport's *Lower Bound on Asynchronous Consensus* [9] recognises a special case wherein a value may be stably learned in one message delay provided that it is submitted by one proposer. We now know of at least two protocols that support this optimisation. Likewise, there may be other protocols similar to SP or Multi-Paxos that compose over arbitrary single-value consensus instances.

This result highlights an important observation: *the specification of a single-value protocol and its projection to a replicated log represent two distinct problems*. It is objectively easier to reason about the two problems in isolation, extend and optimise them individually and prove their correctness discretely.

Compare the modular approach of SP to a monolithic design. For the latter [2], [5], [6], one has to prove correctness for a greater number of admissible states and behaviours. Any prospective optimisations or refinements may invalidate prior reasoning and require a new proof of the complete algorithm. From a pedagogical viewpoint, the entire specification needs to be conveyed, subjecting the audience to increased cognitive load.

E. PROOF SKETCH

Lamport famously narrated in [1] that “*consistency and progress properties of the parliamentary protocol follow immediately from the corresponding properties of the Synod protocol from which it was derived*,” and that “*the Paxons never bothered writing a precise description of the parliamentary protocol because it was so easily derived from the Synod protocol*.”

In this section, we sketch a proof of SP's correctness. Note that of the propositions below, only *Lemma 6* (used by *Theorem 4*) is remotely interesting. The others essentially convey the properties of the underlying single-value protocol. A case in point: a proof this compact would have been unattainable, were SP of a monolithic design. Needless to say, our own experience is congruous with that of the ancient Paxons.

Theorem 2 (Agreement): *If a process delivers a command m in slot s , then all processes eventually deliver m in s .*

Proof: *A priori*, all processes observe the log as an identical sequence of addressable slots. For any s , all processes consistently observe the value v of s by the properties of the underlying single-value protocol. m is a constituent of v , being in the triplet $v = \langle m, _, _ \rangle$. By the *agreement property* of the underlying protocol (*Theorem 1* for Spire), if v is chosen in s , then every process eventually observes v , and by extension, m in s . \square

Theorem 3 (Total order): *If two processes p and q both deliver commands m and n , then p delivers m before n , if and only if q delivers m before n .*

Proof: By *Theorem 2*, all processes observe identical commands for the same slot. By *Behaviour 3*, no process delivers a command in any slot until it has delivered all commands in prior slots. Therefore, if a process delivers m before n , then every process delivers m before n . \square

Lemma 6: For any contiguous span S of length Γ , for all marked slots in S , the $\rho - \theta$ pairs are identical.

Proof: Take S to occupy the range $s..(s + \Gamma - 1)$. Consider the two cases where a slot may be marked: by (1) a non-privileged proposer q , or (2) a privileged proposer p .

For (1), we examine *Behaviour 2*, stating that for some slot s marked by p , for q to preempt p , it must commit a contiguous sequence of slots in the range $(s + 1)..(s + \Gamma)$, such that slots $(s + 1)..(s + \Gamma - 1)$ are unmarked and $s + \Gamma$ is marked with ρ_q and θ_q . s is adjacent to $(s + 1)..(s + \Gamma - 1)$, and their concatenation yields a contiguous span G of length Γ . If s is marked, then G has exactly one marked slot; otherwise, G is unmarked in its entirety. Next, $(s + 1)..(s + \Gamma - 1)$ and $s + \Gamma$ are adjacent; their concatenation is a contiguous span H of length Γ containing at most one marked slot.

For (2), by rewriting the indices of the range $s..(s + \Gamma - 1)$, it suffices to show that p will not offer a marked value in slot t unless the offered $\rho - \theta$ pair matches all marked pairs in $(t - \Gamma)..(t - 1)$ (truncated at 1). However, *Behaviour 1* only states that p will not offer a marked value in t until it has asserted that $(t - \Gamma)..(t - 1)$ has at least one slot marked with ρ_p and θ_p . We make the leap from at least one marked slot to all marked slots by induction. The base case: p has just acquired privileged status and every marked slot in $(t - \Gamma)..(t - 1)$ contains ρ_p and θ_p by *Behaviour 2*. The inductive step: for a span $(g - \Gamma)..(g - 1)$ in which the invariant holds, offering a marked value in g with some $\rho - \theta$ pair sourced from $(g - \Gamma)..(g - 1)$ trivially preserves the invariant. \square

Theorem 4 (Singularity of Privilege): If a value v is offered in s by a process exercising its slot privilege, no value other than v may be privilege-offered in s .

Proof: By *Behaviour 1*, for p to exercise its privilege in s , at least one marked slot in the range $S = (s - \Gamma)..(s - 1)$ (truncated at slot 1) must contain ρ_p and θ_p . For p to be the sole privileged proposer in s , all marked slots in S must be identical. The length of S is Γ without truncation, or at most Γ otherwise. By *Lemma 6*, all marked slots in any contiguous span of length Γ are identical. \square

F. MEMBERSHIP CHANGES

One practical consideration of a multi-value consensus protocol is the live amendment of the ensemble configuration—altering the number of consenterers. We do not contribute specific approaches to group membership in this paper, as these are orthogonal to the consensus protocol. The α approach of [1], [18], [36] is compatible with SP and will suffice.

VII. CONCLUSION

This paper demonstrates that the characteristic similarities among existing consensus protocols are largely attributable to the incidentals of their design, rather than to a fundamental constraint inherent to the problem space. This claim has been substantiated via a counterexample: a single-value consensus protocol that exercises *cooperative agreement* across a *phase-symmetric* round structure.

We suggest that by halving the number of distinct messages and behaviours, Spire makes consensus more approachable to the broader audience; it may serve as an alternate foundation for teaching consensus to beginners. The reduction in behaviours and message schemas may also simplify the process of implementation and testing.

Spire's singular characteristics offer a genuinely uncharted territory for future research and exploration. This paper has barely scratched the surface of phase symmetry and cooperative agreement. There are significant opportunities for further elaboration of the protocol—not only by proposing novel variants but also by selectively adapting or retrofitting optimisations used in existing protocols.

We closed with a high-level protocol for composing a sequence of arbitrary single-value consensus instances into a replicated log. The resulting modularity allowed us to devise and reason about the two protocols independently. We are confident that this technique will further aid the comprehension of distributed consensus and streamline its implementations.

REFERENCES

- [1] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998.
- [2] B. Oki and B. Liskov, "Viewstamped replication: A new primary copy method to support highly-available distributed systems," in *Proc. 7th Annu. ACM Symp. Princ. Distr. Comp.*, 1988, pp. 8–17.
- [3] B. Liskov and C. James, "Viewstamped replication revisited," MIT, Cambridge, MA, USA, Tech. Rep. MIT-CSAIL-TR-2012-021, 2012.
- [4] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, no. 2, pp. 225–267, Mar. 1996.
- [5] F. P. Junqueira, B. C. Reed, and M. Serafini, "Zab: High-performance broadcast for primary-backup systems," in *Proc. IEEE/IFIP 41st Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2011, pp. 245–256.
- [6] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 305–319.
- [7] Y. Mao, F. P. Junqueira, and K. Marzullo, "Mencius: Building efficient replicated state machines for WANs," in *Proc. 8th USENIX Conf. Oper. Syst. Design Implement.*, 2008, pp. 369–384.
- [8] P. Jalili Marandi, M. Primi, N. Schiper, and F. Pedone, "Ring paxos: A high-throughput atomic broadcast protocol," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2010, pp. 527–536.
- [9] L. Lamport, "Lower bounds for asynchronous consensus," Microsoft Res., Redmond, WA, USA, Tech. Rep. MSR-TR-2004-72, 2004.
- [10] B. W. Lampson, "How to build a highly available system using consensus," in *Proc. 10th Int. Workshop Distr. Algorithms*, 1996, pp. 1–17.
- [11] C. Cachin, "Yet another visit to Paxos," IBM Res., Zurich, Switzerland, Tech. Rep. RZ3754, 2009.
- [12] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proc. 7th Symp. Oper. Syst. Design Implement.*, 2006, pp. 335–350.
- [13] R. van Renesse, N. Schiper, and F. B. Schneider, "Vive la différence: Paxos vs. viewstamped replication vs. zab," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 4, pp. 472–484, Aug. 2015.
- [14] Z. Wang, C. Zhao, S. Mu, H. Chen, and J. Li, "On the parallels between paxos and raft, and how to port optimizations," *Proc. 2019 ACM Symp. Princ. Distr. Comput.*, 2019, pp. 445–454.
- [15] R. van Renesse and D. Altinbeken, "Paxos made moderately complex," *ACM Comput. Surv.*, vol. 42, no. 3, pp. 1–36, 2015.
- [16] H. Howard, "ARC: Analysis of raft consensus," Univ. Cambridge Comput. Lab., Cambridge, U.K., Tech. Rep. UCAM-CL-TR-857, 2014.
- [17] R. De Prisco, B. W. Lampson, and N. A. Lynch, "Revisiting the Paxos algorithm," in *Proc. 11th Int. Workshop Distrib. Algorithms*, 1997, pp. 111–125.
- [18] L. Lamport, "Paxos made simple," *ACM SIGACT News*, vol. 32, no. 4, pp. 18–25, Dec. 2001.

- [19] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, 1988.
- [20] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985.
- [21] I. Moraru, D. Andersen, and M. Kaminsky, "There is more consensus in egalitarian parliaments," in *Proc. 24th ACM Symp. Oper. Syst. Princ.*, Nov. 2013, pp. 358–372.
- [22] J. Nagle, "Congestion control in IP/TCP Internetworks," Network Working Group, USA, Tech. Rep. RFC-896, 1984. [Online]. Available: <https://tools.ietf.org/html/rfc896>
- [23] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Trans. Commun.*, vol. COM-22, no. 5, pp. 637–648, May 1974.
- [24] A. Turcu, S. Peluso, R. Palmieri, and B. Ravindran, "Be general and don't give up consistency in geo-replicated transactional systems," in *Proc. OPODIS*, 2014, pp. 33–48.
- [25] B. Lamport and H. Sturgis, "Crash recovery in a distributed data storage system. Technical report," Comput. Sci. Lab., Xerox Palo Alto Research Centre, Palo Alto, CA, USA, Tech. Rep., 1976.
- [26] J. N. Gray, "Notes on database operating systems," in *Operating Systems—An Advanced Course (Lecture Notes in Computer Science)*, vol. 60, M. J. Flynn, Eds. London, U.K.: Springer, 1978, pp. 393–481.
- [27] J. Gray and L. Lamport, "Consensus on transaction commit," *ACM Trans. Database Syst.*, vol. 31, no. 1, pp. 133–160, Mar. 2006.
- [28] N. Nouali, H. Drias, and A. Doucet, "A mobility-aware two-phase commit protocol," *Int. Arab J. Inf. Technol.*, vol. 3, no. 1, pp. 1–8, 2006.
- [29] J. N. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*. Burlington, MA, USA: Morgan Kaufman, 1993.
- [30] L. Liu, D. Agrawal, and A. El Abbadi, "The performance of two-phase commit protocols in the presence of site failures," Dept. Comput. Sci., Univ. California, Los Angeles, CA, USA, Tech. Rep. TRCS94-09, Apr. 1994.
- [31] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Reading, MA, USA: Addison-Wesley, 1987.
- [32] D. Skeen, "A quorum-based commit protocol," Cornell Univ., New York, NY, USA, Tech. Rep. 82-483, 1982.
- [33] S. Maiyya, F. Nawab, D. Agrawal, and A. E. Abbadi, "Unifying consensus and atomic commitment for effective cloud data management," *Proc. VLDB Endowment*, vol. 12, no. 5, pp. 611–623, Jan. 2019.
- [34] I. Keidar and D. Dolev, "Increasing the resilience of distributed and replicated database systems," *J. Comput. Syst. Sci.*, vol. 57, no. 3, pp. 309–324, Dec. 1998.
- [35] A. Mocanu and C. Bădică, "Bringing Paxos consensus in multi-agent systems," in *Proc. 4th Int. Conf. Web Intell., Mining Semantics (WIMS)*, 2014, pp. 1–6.
- [36] L. Lamport, D. Malkhi, and L. Zhou, "Reconfiguring a state machine," *ACM SIGACT News*, vol. 41, pp. 63–73, Mar. 2010.



EMIL KOUTANOV received the B.E. degree *summa cum laude* from the University of Technology Sydney, Australia, in 2007.

He is currently a consulting software architect and a part-time researcher. His professional career has spanned several diverse industries including wagering, telecommunications, construction and finance, focusing on the design and implementation of reliable, and scalable distributed systems. He has authored several articles on open-source middleware, application design and architecture, and a reference book on the subject of event streaming. His research interests include the analysis and design of distributed algorithms and models of computation, fault-tolerance and continuous availability, formal methods, concurrent and parallel computing, consistency models, and consensus.

• • •