Verification of Eventual Consensus in Synod Using a Failure-Aware Actor Model

Saswata Paul[1], Gul A. Agha[2], Stacy Patterson[1], and Carlos A. Varela[1]

[1] Rensselaer Polytechnic Institute, Troy, New York, 12180, USA
pauls4@rpi.edu, {sep,cvarela}@cs.rpi.edu
[2] University of Illinois at Urbana-Champaign, Champaign, Illinois, 61820, USA
agha@illinois.edu

Abstract. Successfully attaining consensus in the absence of a centralized coordinator is a fundamental problem in distributed multi-agent systems. We analyze progress in the Synod consensus protocol—which does not assume a unique leader—under the assumptions of asynchronous communication and potential agent failures. We identify a set of sufficient conditions under which it is possible to guarantee that a set of agents will eventually attain consensus. First, a subset of the agents must behave correctly and not permanently fail until consensus is reached, and second, at least one proposal must be eventually uninterrupted by higher-numbered proposals. To formally reason about agent failures, we introduce a failure-aware actor model (FAM). Using FAM, we model the identified conditions and provide a formal proof of eventual progress in Synod. Our proof has been mechanically verified using the Athena proof assistant and, to the best of our knowledge, it is the first machine-checked proof of eventual progress in Synod.

1 Introduction

Consensus, which requires a set of processes to reach an agreement on some value, is a fundamental problem in distributed systems. Under *asynchronous* communication settings, where message transmission and processing delays are unbounded, it is impossible to guarantee consensus [18] since message delays cannot be differentiated from process failures. Nevertheless, in distributed *multi-agent* systems, where there is no centralized *coordinator* to manage safe operation, it is necessary for the agents to use *distributed consensus protocols* [58] for coordination. An important application of such systems is decentralized *air-traffic control* (ATC) for *Urban Air Mobility* (UAM) [63].

The integration of *uncrewed aircraft systems* (UAS) and *micro-aircraft* in the *National Airspace System* (NAS) for package delivery, scientific data collection, and urban transportation will significantly increase the density of urban air traffic [44], elevating the possibilities of hazards such as *near mid-air collisions* (NMAC) [35] and *wake-vortex induced rolls* [37]. Since centralized, human-operated ATC is not scalable to high densities and is prone to human errors [22], UAS operating in UAM scenarios must be capable of autonomous *UAS traffic*

management (UTM) [9]. To ensure safety in UTM, they must coordinate with each other by using distributed consensus protocols[3] (*e.g.* – [10, 54]).

In [51], we have proposed an *Internet of Planes* (IoP), consisting of an asynchronous *vehicle-to-vehicle* (V2V) [41] network of aircraft, to facilitate autonomous capabilities such as *decentralized admission control* (DAC) [53]. In DAC, a *candidate* aircraft generates a *conflict-aware flight plan* that avoids NMACs with a set of *owner* aircraft of a *controlled airspace* [52]. The candidate then requests admission into the airspace by proposing this flight plan to the owners. As there may be multiple candidates concurrently competing for admission into an airspace, the owners may only admit candidates sequentially. This is because candidates do not consider each other in their proposals: in fact, a candidate may not even be aware of other candidates. Thus, admitting one candidate potentially invalidates the proposals of all other candidates. Since UAM applications are *time-critical*, any consensus protocol used for DAC must guarantee that a proposal will be eventually chosen. In this paper, we present an analysis of consensus that is primarily motivated by the requirements of DAC.

In [29], Lamport describes three variants of a consensus protocol that strongly guarantees the *safety* property that only one value will be chosen:

- *The Basic Synod protocol* or *Synod* guarantees safety if one or more agents are allowed to initiate new proposals for consensus.
- *The Complete Synod protocol* or *Paxos* is a variant of Synod in which only a distinguished agent (*leader*) is allowed to initiate proposals [30]. Other agents may only introduce values through the leader. This is done to guarantee the *progress* property that consensus will eventually be achieved.
- *The Multi-Decree protocol* or *Multi-Paxos* allows multiple values to be chosen using a separate instance of Paxos for each value, but using the same leader for each of those instances.

Both Paxos and Multi-Paxos use Synod as the underlying consensus protocol.

A progress guarantee contingent upon a unique leader has several drawbacks from the perspective of DAC. First, *leader election* itself is a consensus problem. Therefore, a progress guarantee that relies on successful leader election as a precondition would be circular, and therefore fallacious. Second, the Fischer, Lynch, and Paterson *impossibility result* (FLP) [18] implies that unique leadership cannot be guaranteed in asynchronous systems where agents may unpredictably fail. In some cases, network partitioning may also erroneously cause multiple leaders to be elected [4]. Third, V2V networks like the IoP are expected to be highly dynamic where membership frequently changes. Consensus is used in DAC for agreeing on only a single candidate, after which the set of owners changes by design. Hence, the benefits of electing a stable leader that apply for *state machine replication* (*e.g.* – [27]), where reconfiguration is expected to be infrequent [34], are not applicable to DAC. Finally, channeling proposals through a unique leader creates a communication bottleneck and introduces the leader as a single point of failure: there can be no progress if the specified leader fails. In the absence of a

[3] An approach for *implicit coordination* between two aircraft has been proposed in [43], but that is only applicable for the purpose of *pairwise tactical conflict avoidance*.

unique leader, a system implementing Paxos falls back to the more general Synod protocol. Therefore, in this paper, we focus on identifying sufficient conditions under which the fundamental Synod protocol can make eventual progress.

The failure of *safety-critical* aerospace systems can lead to the loss of human life [24,61] and property [25]. Hence, formal methods must be used for the rigorous verification of any algorithm used in such systems. The *actor model* [1,21] is a theoretical model of concurrent computation that can be used for formal reasoning about distributed algorithms [42]. It assumes asynchronous communication as the most primitive form of interaction and it also assumes *fairness*, which is useful for reasoning about the progress of actor systems [62]. In the context of DAC, aircraft may experience temporary or permanent communication failures where they are unable to send or receive any messages [46]. Our verification of Synod must model the possibility of such failures. To support explicit reasoning about such failures in actors, we introduce a *(predicate-fair) failure-aware actor model* (FAM) that assumes *predicate fairness* [55] in addition to the standard actor model's fairness properties. Predicate fairness states that if a predicate is *enabled infinitely often* in a given path, then it must be eventually satisfied. We use Varela's dialect [62] of Agha, Mason, Smith, and Talcott's actor language (AMST) [3] and modify its semantics to model failures.

In order to ensure that our formalization of Synod and its eventual progress property are correct, we machine-check our proof using the *Athena proof assistant* [5, 7]. Along with a language for expressing proofs, Athena also provides an interactive proof development environment. Athena's theorem proving capabilities are based on *many-sorted first-order logic* [39] and it uses a *natural deduction* [6] style of proofs. All terms have an associated *sort* and Athena can automatically detect and report *ill-sorted* terms and expressions in proofs. Athena is *sound*: all methods that successfully execute produce a theorem that is guaranteed to be a logical consequence of its *assumption base*. It also allows the use of *automated theorem provers* like Vampire [59] and SPASS [64].

The main contributions of this work are:

- We identify a set of conditions under which progress in Synod can be guaranteed in purely asynchronous settings, without assuming a unique leader.
- We introduce a failure-aware actor model to support formal reasoning about temporary or permanent actor failures.
- We show that progress can be guaranteed in Synod under the identified conditions and mechanically verify our proof using Athena. To our knowledge, this is the first machine-checked proof of eventual progress in Synod.

It is important to note here that a guarantee of eventual progress *alone* is insufficient for time-critical UAM applications, but it is a necessary precondition for providing *timely progress* guarantees that can be directly applicable for UAM.

The paper is structured as follows – Section 2 informally describes Synod and discusses the conditions required for progress; Section 3 introduces FAM; Section 4 presents the formal verification of progress in Synod; Section 5 relates our work to prior research on formal verification of consensus protocols; and Section 6 concludes the paper including potential future directions of work.

2 The Synod Protocol

Synod assumes an asynchronous, non-Byzantine system model in which agents operate at arbitrary speed, may fail and restart, and have stable storage. Messages can be duplicated, lost, and have arbitrary transmission time, but cannot be corrupted [30]. It consists of two logically separate sets of agents:

- *Proposers* - The set of agents that can propose values to be chosen.
- *Acceptors* - The set of agents that can vote on which value should be chosen.

Synod requires a subset of acceptors, which satisfy a *quorum*, to proceed. To ensure that if there is a consensus in one quorum, there cannot also be another quorum with a consensus, any two quorums must intersect. A subset of acceptors which constitutes a simple majority is an example of a quorum. Other methods of determining quorums can be found in [23].

There are four types of messages in Synod:

- *prepare (1a)* messages include a *proposal number*.
- *accept (2a)* messages include a proposal number and a *value*.
- *promise (1b)* messages include a proposal number and a value.
- *voted (2b)* messages include a proposal number and a value.

For each proposer, the algorithm proceeds in two distinct *phases* [30]:

- **Phase 1**
  (a) A proposer P selects a *unique proposal number* b and sends a *prepare* request with b to a subset Q of acceptors, where Q constitutes a quorum.
  (b) When an acceptor A receives a *prepare* request with the proposal number b, it checks if b is greater than the proposal numbers of all *prepare* requests to which A has already responded. If this condition is satisfied, then A responds to P with a *promise* message. The *promise* message implies that A will not accept any other proposal with a proposal number less than b. The promise includes (i) the highest-numbered proposal b' that A has previously accepted and (ii) the value corresponding to b'. If A has not accepted any proposals, it simply sends a default value.
- **Phase 2**
  (a) If P receives a *promise* message in response to its *prepare* requests from all members of Q, then P sends an *accept* request to all members of Q. These *accept* messages contain the proposal number b and a value v, where v is the value of the highest-numbered proposal among the responses or an arbitrary value if all responses reported the default value.
  (b) If an acceptor A receives an *accept* request with a proposal number b and a value v from P, it accepts the proposal unless it has already responded to a *prepare* request having a number greater than b. If A accepts the proposal, it sends P a *voted* message which includes b and v [29].

A proposer *determines* that a proposal was successfully *chosen* if and only if it receives *voted* messages from a quorum for that proposal.

Synod allows multiple proposals to be chosen. Safety is ensured by the invariant - *"If a proposal with value v is chosen, then every higher numbered-proposal that is chosen has value v"* [30]. Therefore, there may be situations where a

proposer P proposes a proposal number after one or more lower-numbered proposals have already been chosen, resulting in some value v being chosen. By design, Synod will ensure that P proposes the same value v in its Phase 2. Since proposers can initiate proposals in any order and communication is asynchronous, the only fact that can be guaranteed about any chosen value is that it must have been proposed by the first proposal to have been chosen by any quorum. From the context of admission control, it implies that if a candidate P_2 successfully completes both phases after another candidate P_1 has completed both phases, then P_2 will simply learn that P_1 has been granted admission. So P_2 will update its set of owners to include P_1, create a new conflict-aware flight plan, and request admission by starting the Synod protocol again.

2.1 Progress in Synod

Some obvious scenarios in which progress may be affected in Synod are:

- Two proposers P_1 and P_2 may complete Phase 1 with proposal numbers b_1 and b_2 such that $b_2 > b_1$. This will cause P_1 to fail Phase 2. P_1 may then propose a fresh proposal number $b_3 > b_2$ and complete Phase 1 before P_2 completes its Phase 2. This will cause P_2 to fail Phase 2 and propose a fresh proposal number $b_4 > b_3$. This process may repeat infinitely [30] (*livelock*).
- Progress may be affected even if one random agent fails unpredictably [18].

Paxos assumes that a distinguished proposer (leader) is elected as the only proposer that can initiate proposals [29]. Lamport states *"If the distinguished proposer can communicate successfully with a majority of acceptors, and if it uses a proposal with number greater than any already used, then it will succeed in issuing a proposal that is accepted."* [30]. The FLP impossibility result [18] implies that in purely asynchronous systems, where agent failures cannot be differentiated from message delays, leader election cannot be guaranteed. Moreover, it is possible that due to network partitioning, multiple proposers are elected as leaders [4]. In the absence of a unique leader, a system implementing Paxos falls back to Synod. Therefore, it is important to identify the conditions under which progress can be formally guaranteed in Synod in the absence of a unique leader.

To guarantee progress, it suffices to show that *some* proposal number b will be chosen. This will happen if b satisfies the following conditions:

P1 When an acceptor A receives a *prepare* message with b, b should be greater than all other proposal numbers that A has previously seen.

P2 When an acceptor A receives an *accept* message with b, b should be greater than or equal to all other proposal numbers that A has previously seen.

P1 and *P2* simply suggest that for b, there will be a long enough period without *prepare* or *accept* messages with a proposal number greater than b, allowing messages corresponding to b to get successfully processed without being interrupted by messages corresponding to a higher-numbered proposal. The non-interruption condition follows from two assumptions. First, we assume that a proposer P will keep retrying until it successfully receives votes from a quorum. Second, we assume that the system has a form of fairness called *predicate fairness* [55]. With predicate fairness, if a predicate is enabled infinitely often in a given path, then

the predicate must be satisfied (this is a recursive definition in that the path could begin from any state along the path). Conditions *P1* and *P2* are infinitely often enabled in any path corresponding to livelock. Therefore, by predicate fairness, these conditions must eventually happen, allowing consensus to be reached.[4]

Since progress cannot be guaranteed if too many agents permanently fail or if too many messages are lost, Lamport [33] presents some conditions for informally proving progress in Paxos. A *nonfaulty* agent is defined as *"an agent that eventually performs the actions that it should"*, and a *good* set is defined as a set of nonfaulty agents, such that, if an agent repeatedly sends a message to another agent in the set, it is eventually received by the recipient. It is then assumed that the unique leader and a quorum of acceptors form a good set and that they infinitely repeat all messages that they have sent. These conditions are quite strong since they depend on the future behavior of a subset of agents and may not always be true of an implementation. However, since they have been deemed reasonable for informally proving progress even in the presence of a unique leader, we partially incorporate them in our conditions under which progress in Synod can be formally guaranteed in the absence of a unique leader. Our complete set of conditions for guaranteeing progress in Synod, therefore, informally states that *eventually, a nonfaulty proposer must propose a proposal number, that will satisfy P1 and P2, to a quorum of nonfaulty acceptors, and the Synod-specific messages between these agents must be eventually received.*

We can see that the conditions for progress in Paxos constitute a special case of our conditions for progress in Synod where *P1* and *P2* are satisfied by a proposal proposed by the unique leader. If the unique leader permanently fails, then the corresponding guarantee is only useful if leader re-election is successful. However, if leader election is assumed to have already succeeded, there will be no need for further consensus, rendering the guarantee moot. Synod's progress guarantee remains useful as long as at least one random proposer is available to possibly propose at least one successful proposal, thereby remaining pertinent even if multiple (not all) proposers arbitrarily fail. Moreover, the conditions do not assume that consensus (leader election) will have already succeeded.

3 A Failure-Aware Actor Model

We use the actor model to formally reason about progress in Synod since it assumes asynchronous communication and fairness, which is helpful for reasoning about progress [3]. Fairness in the standard actor model has the following consequences [62]:

- guaranteed message delivery[5], and

[4] We do not use the predicate fairness assumption in the formal proof associated with this paper. Instead, we use a system-specific derived property: that eventually, at least one proposal must be uninterrupted by higher-numbered proposals.

[5] "Delivery" here implies that the message will be available to the recipient. The recipient may or may not eventually receive and process the message.

– an actor infinitely often ready to process a message will eventually process the message.

The IoP is an open network in which aircraft communicate asynchronously and may experience permanent or temporary communication failures that render them unable to send or receive any messages. All messages to and from an aircraft may get delayed because of transmission problems or internal processing delays (or processing failures) in the aircraft. In asynchronous communication, since message transmission times are unbounded, it is not possible to distinguish transmission delays from processing delays or failures. However, it is important to take into account if an actor has failed at any given time, *i.e.*, if it is incapable of sending or receiving messages. For this reason, we introduce a (predicate-fair) failure-aware actor model (FAM) that allows reasoning about such actor failures.

FAM models two states for an actor at any given time—*available* or *failed*. Actors can switch states as *transitions* between *configurations*. From the perspective of message transmission and reception, a failed actor cannot send or receive *any* messages, but an available actor can. The failure model of FAM also assumes that every actor has a stable storage that is persistent across failures. In addition to the standard actor model's fairness assumptions, FAM assumes predicate fairness [55], which states that a predicate that is infinitely often enabled in a given path will eventually be satisfied.

Varela [62] presents a dialect of AMST's *lambda-calculus* based actor language [3] whose operational semantics are a set of *labeled transitions* from *actor configurations* to actor configurations[6]. An actor configuration κ is a temporal snapshot of actor system components, namely the individual actors and the messages "en route". It is denoted by $\langle\!\langle \alpha \parallel \mu \rangle\!\rangle$, where α is a map from actor names to *actor expressions*, and μ is a multi-set of messages. An actor expression e is either a value v or a *reduction context* R filled with a *redex* r, denoted as $e = \mathsf{R} \blacktriangleright \mathsf{r} \blacktriangleleft$. $\kappa_1 \xrightarrow{l} \kappa_2$ denotes a *transition rule* where κ_1, κ_2, and l are the initial configuration, the final configuration, and the transition label respectively. There are four possible transitions – **fun**, **new**, **snd**, and **rcv**. To model failures, we modify Varela's dialect of AMST and categorize its original transitions (**fun**, **new**, **snd**, and **rcv**) as *base-level* transitions.

For a base-level transition in FAM to be enabled to occur for an *actor in focus* at any time, the actor needs to be available at that time. To denote available and failed actors at a given time, we redefine an actor configuration as $\langle\!\langle \alpha \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle$, where α is a map from actor names to actor expressions for available actors, $\bar{\alpha}$ is a map from actor names to actor expressions for failed actors, and μ is a multi-set of messages "en route".

To model actor failure and restart, we define two *meta-level* transitions **stp** (*stop*) and **bgn** (*begin*) that can stop an available actor or start a failed actor in its persistent state before failure. The **stp** transition is only enabled for an actor in the available state and the **bgn** transition is only enabled for an actor

[6] Interested readers can refer to section 4.5 of [62] for more details.

$$\frac{e \rightarrow_\lambda e'}{\langle\!\langle \alpha, [\mathsf{R} \blacktriangleright e \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle \xrightarrow{[\mathbf{fun}:a]} \langle\!\langle \alpha, [\mathsf{R} \blacktriangleright e' \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle}$$

$$\langle\!\langle \alpha, [\mathsf{R} \blacktriangleright \texttt{new}(b) \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle \xrightarrow{[\mathbf{new}:a,a']} \langle\!\langle \alpha, [\mathsf{R} \blacktriangleright a' \blacktriangleleft]_a, [\texttt{ready}(b)]_{a'} \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle \quad a' \textit{ fresh}$$

$$\langle\!\langle \alpha, [\mathsf{R} \blacktriangleright \texttt{send}(a', v) \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle \xrightarrow{[\mathbf{snd}:a]} \langle\!\langle \alpha, [\mathsf{R} \blacktriangleright \texttt{nil} \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \mu \uplus \{\langle a' \Leftarrow v \rangle\} \rangle\!\rangle$$

$$\langle\!\langle \alpha, [\mathsf{R} \blacktriangleright \texttt{ready}(b) \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \{\langle a \Leftarrow v \rangle\} \uplus \mu \rangle\!\rangle \xrightarrow{[\mathbf{rcv}:a,v]} \langle\!\langle \alpha, [b(v)]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle$$

Fig. 1: Operational semantics for the base-level transition rules.

$$\langle\!\langle \alpha, [e]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle \xrightarrow{[\mathbf{stp}:a]} \langle\!\langle \alpha|_{\mathit{dom}(\alpha)-\{a\}} \parallel \bar{\alpha}, [e]_a \parallel \mu \rangle\!\rangle$$

$$\langle\!\langle \alpha \parallel \bar{\alpha}, [e]_a \parallel \mu \rangle\!\rangle \xrightarrow{[\mathbf{bgn}:a]} \langle\!\langle \alpha, [e]_a \parallel \bar{\alpha}|_{\mathit{dom}(\bar{\alpha})-\{a\}} \parallel \mu \rangle\!\rangle$$

Fig. 2: Operational semantics for the meta-level transition rules.

in the failed state. Fig. 1 and Fig. 2 show the operational semantics of our actor language as labelled transition rules[7,8].

For an actor configuration $\kappa = \langle\!\langle \alpha \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle$ to be syntactically well-formed in our actor language, it must conform to the following[9]:

1. $\forall a,\ a \in \mathit{dom}(\alpha) \cup \mathit{dom}(\bar{\alpha}),\ \mathit{fv}(\alpha(a)) \subseteq \mathit{dom}(\alpha) \cup \mathit{dom}(\bar{\alpha})$
2. $\forall m,\ m \in \mu,\ m = \langle a \Leftarrow v \rangle,\ \mathit{fv}(a) \cup \mathit{fv}(v) \subseteq \mathit{dom}(\alpha) \cup \mathit{dom}(\bar{\alpha})$
3. $\mathit{dom}(\alpha) \cap \mathit{dom}(\bar{\alpha}) = \emptyset$

The standard actor model's fairness assumptions apply only to the base-level transitions in our language and not to the meta-level transitions.

4 Formal Verification of Eventual Progress in Synod

This section presents our proof of eventual progress in the Synod protocol. The notations used in this section have been introduced in Table 1 and Table 2.

A message is a tuple $\langle \mathtt{s} \in \mathcal{A}, \mathtt{r} \in \mathcal{A}, \mathtt{k} \in \xi, \mathtt{b} \in \mathcal{B}, \mathtt{v} \in \mathcal{V} \rangle$ where $\xi = \{1a, 1b, 2a, 2b\}$, `s` is the sender, `r` is the receiver, `k` is the type of message, `b` is a proposal number, and `v` is a value. $\bar{v} \in \mathcal{V}$ is a null value constant used in $1a$ (*prepare*) messages.

[7] $\rightarrow_\lambda$ denotes lambda calculus semantics, essentially beta-reduction. `new`, `send`, and `ready` are actor redexes. $\langle a \Leftarrow v \rangle$ denotes a message for actor a with value v. $\alpha, [e]_a$ denotes the extended map α', which is the same as α except that it maps a to e. $\uplus$ denotes multiset union. $\alpha|_S$ denotes restriction of mapping α to elements in set S. $\mathit{dom}(\alpha)$ is the domain of α.

[8] More details about actor language semantics can be found in [2], [3], and [62].

[9] $\mathit{fv}(e)$ is the set of free variables in the expression e.

Symbol	Description	Symbol	Description
$\mathcal{A}$	Set of all actors	$\mathcal{M}$	Set of all messages
$\mathbb{P}$	Set of all proposer actors	$\mathbb{A}$	Set of all acceptor actors
$\mathcal{V}$	Set of all values	$\mathcal{Q}$	Set of all quorums
$\mathcal{B}$	Set of all proposal numbers	$\mathbb{M}$	Set of all sets of messages
$\mathcal{C}$	Set of all actor configurations	$\mathcal{S}$	Set of all transition steps
$\mathcal{T}$	Set of all fair transition paths	$\mathbb{N}$	Set of all natural numbers

Table 1: Set symbols for our formal specification.

Symbol	Description	Input	Output
ς	Get last configuration	$\mathcal{T}$	$\mathcal{C}$
ρ	Get transition path up to index	$\mathcal{T} \times \mathbb{N}$	$\mathcal{T}$
T	Transition path constructor	$\mathcal{T} \times \mathcal{S}$	$\mathcal{T}$
σ	Choose a value to propose based on configuration	$\mathcal{C} \times \mathcal{A}$	$\mathcal{V}$
$\mathfrak{s}$	Construct a `snd` transition step	$\mathcal{A} \times \mathcal{M}$	$\mathcal{S}$
$\mathfrak{r}$	Construct a `rcv` transition step	$\mathcal{A} \times \mathcal{M}$	$\mathcal{S}$
$\mathfrak{m}$	Get set of messages "en route"	$\mathcal{C}$	$\mathbb{M}$
$\mathfrak{a}$	Actor is in α	$\mathcal{C} \times \mathcal{A}$	`Bool`
$\Re$	Actor is ready for a step	$\mathcal{T} \times \mathcal{A} \times \mathcal{S}$	`Bool`
ϕ	Proposer has promises from a quorum	$\mathcal{A} \times \mathcal{B} \times \mathcal{Q} \times \mathcal{C}$	`Bool`
Φ	Proposer has votes from a quorum	$\mathcal{A} \times \mathcal{B} \times \mathcal{Q} \times \mathcal{C}$	`Bool`
đ	Actor is nonfaulty	$\mathcal{A}$	`Bool`
þ	Proposal number satisfies *P1* and *P2*	$\mathcal{B}$	`Bool`
Ł	Proposer has learned of successful consensus	$\mathcal{A} \times \mathcal{B} \times \mathcal{C}$	`Bool`

Table 2: Relation symbols for our formal specification.

The local state of an actor x can be extracted from a configuration κ as a tuple $\langle \eta_\kappa^x \in \mathbb{M}, \beta_\kappa^x \in \mathcal{B}, v_\kappa^x \in \mathcal{V} \rangle$ where η_κ^x is the set of messages received but not yet responded to, β_κ^x is the highest proposal number seen, and v_κ^x is the value corresponding to the highest proposal number accepted.

Transition paths represent the dynamic changes to actor configurations as a result of transition steps [42]. *Indexed positions* in transition paths correspond to logical steps in time and are used to express eventuality.

$\mathbb{P} \subset \mathcal{A}$ and $\mathbb{A} \subset \mathcal{A}$ are the sets of proposers and acceptors respectively and a quorum is a possibly equal non-empty subset of $\mathbb{A}$, *i.e.*,$\forall Q \in \mathcal{Q} : Q \subseteq \mathbb{A} \wedge Q \neq \emptyset$.

4.1 Fairness Assumptions for Transitions

We assume two fairness axioms for the **snd** and **rcv** transitions that follow from the fairness assumptions of FAM. The `F-Snd-axm` and the `F-Rcv-axm` state that if a **snd** or **rcv** transition is enabled at some time, it must either eventually happen or eventually, it must become permanently disabled.

`F-Snd-Axm` $\equiv$
$\forall x \in \mathcal{A}, m \in \mathcal{M}, T \in \mathcal{T}, i \in \mathbb{N} : (\Re(\rho(T, i), x, \mathfrak{s}(x, m)) \implies$
$\qquad ((\exists j \in \mathbb{N} : (j \geq i)$

$$\begin{array}{l} \qquad\qquad \wedge\ \ \rho(T, j+1) = \intercal(\rho(T, j), \mathfrak{s}(x, m))) \\ \qquad \vee\ \ (\exists k \in \mathbb{N} : (k > i) \\ \qquad\qquad \wedge\ \ (\forall j \in \mathbb{N} : (j \geq k)\ \implies\ \neg\Re(\rho(T, j), x, \mathfrak{s}(x, m)))))) \end{array}$$

$$\begin{array}{l} \texttt{F-Rcv-Axm} \equiv \\ \forall x \in \mathcal{A}, m \in \mathcal{M}, T \in \mathcal{T}, i \in \mathbb{N} : ((m \in \mathfrak{m}(\varsigma(\rho(T, i)))\ \ \wedge\ \ \Re(\rho(T, i), x, \mathfrak{r}(x, m)))\ \implies \\ \qquad ((\exists j \in \mathbb{N} : (j \geq i) \\ \qquad\qquad \wedge\ \ \rho(T, j+1) = \intercal(\rho(T, j), \mathfrak{r}(x, m))) \\ \qquad\quad \vee\ \ (\exists k \in \mathbb{N} : (k > i) \\ \qquad\qquad\quad \wedge\ \ (\forall j \in \mathbb{N} : (j \geq k)\ \implies \\ \qquad\qquad\qquad\qquad \neg(m \in \mathfrak{m}(\varsigma(\rho(T, j)))\ \ \wedge\ \ \Re(\rho(T, j), x, \mathfrak{r}(x, m))))))) \end{array}$$

4.2 Rules Specifying the Actions of Synod Actors

The Synod protocol is presented in [29] as a high-level abstraction of the behavior of the agents, while leaving out the implementation details to the discretion of the system developers [47]. We specify rules over actor local states that dictate if an available Synod actor should become ready to send a message. Since Synod does not specify *when* a proposer should send a *prepare* message, we leave that behavior unspecified. For proving progress, we will assume that eventually, a proposer will be ready to send *prepare* messages to a quorum.

$$\begin{array}{l} \texttt{Snd-1b-Rul} \equiv \\ \forall a : \mathbb{A}, p \in \mathbb{P}, T \in \mathcal{T}, i \in \mathbb{N}, b \in \mathcal{B} : \\ (\langle p, a, 1a, b, \bar{v}\rangle \in \eta^a_{\varsigma(\rho(T,i))}\ \ \wedge\ \ \beta^a_{\varsigma(\rho(T,i))} < b\ \ \wedge\ \ \mathfrak{a}(\varsigma(\rho(T, i)), a))\ \implies \\ \qquad\qquad\qquad\qquad\qquad\qquad \Re(\rho(T, i), a, \mathfrak{s}(a, \langle a, p, 1b, b, v^a_{\varsigma(\rho(T,i))}\rangle))) \end{array}$$

$$\begin{array}{l} \texttt{Snd-2a-Rul} \equiv \\ \forall p \in \mathbb{P}, T \in \mathcal{T}, i \in \mathbb{N}, b \in \mathcal{B}, Q \in \mathcal{Q} : \\ (\phi(p, b, Q, \varsigma(\rho(T, i)))\ \ \wedge\ \ \mathfrak{a}(\varsigma(\rho(T, i)), p))\ \implies \\ \qquad\qquad\qquad\qquad (\forall a \in Q : \Re(\rho(T, i), p, \mathfrak{s}(p, \langle p, a, 2a, b, \sigma(\varsigma(\rho(T, i)), p)\rangle)))) \end{array}$$

$$\begin{array}{l} \texttt{Snd-2b-Rul} \equiv \\ \forall a : \mathbb{A}, p \in \mathbb{P}, T \in \mathcal{T}, i \in \mathbb{N}, b \in \mathcal{B}, v \in \mathcal{V} : \\ (\langle p, a, 2a, b, v\rangle \in \eta^a_{\varsigma(\rho(T,i))}\ \ \wedge\ \ \beta^a_{\varsigma(\rho(T,i))} \leq b\ \ \wedge\ \ \mathfrak{a}(\varsigma(\rho(T, i)), a)))\ \implies \\ \qquad\qquad\qquad\qquad\qquad\qquad \Re(\rho(T, i), a, \mathfrak{s}(a, \langle a, p, 2b, b, v\rangle))) \end{array}$$

Since the response to a message in Synod is a finite set of actions, if there is a message in the multi-set for a Synod actor and the actor is also available, then a receive transition is enabled.

$$\begin{array}{l} \texttt{Rcv-Rul} \equiv \\ \forall s, r \in \mathcal{A}, T \in \mathcal{T}, i \in \mathbb{N}, k \in \xi, b \in \mathcal{B}, v \in \mathcal{V} : \\ (\langle s, r, k, b, v\rangle \in \mathfrak{m}(\varsigma(\rho(T, i)))\ \ \wedge\ \ \mathfrak{a}(\varsigma(\rho(T, i)), r)))\ \implies\ \Re(\rho(T, i), r, \mathfrak{r}(r, \langle s, r, k, b, v\rangle)) \end{array}$$

4.3 Assumptions About the Future Behavior of Agents

To prove progress in Synod, we borrow some assumptions about the future behavior of nonfaulty agents used by Lamport for informally proving progress in

Paxos [33]. It is worth noting that being nonfaulty does not prohibit an agent from temporarily failing. It simply means that for every action that needs to be performed by the agent, eventually the agent is available to perform the action and the action happens. In FAM, a nonfaulty actor can be modelled by asserting that an enabled **snd** or **rcv** transition for the actor will eventually happen. However, given the `F-Snd-Axm` and `F-Rcv-Axm` axioms, it suffices to assume that for a nonfaulty actor, if a **snd** or **rcv** transition is enabled, then it will either eventually occur or it will be infinitely often enabled. As FAM does not model message loss, any message in the multi-set will persist until it is received.

We introduce a predicate $\mathfrak{d}$ to specify an actor as nonfaulty, such that:

- $\mathfrak{d}(x)$ implies that the actor x will be eventually available if there is a message "en route" that x needs to receive.
- $\mathfrak{d}(x)$ implies that the actor x will be eventually available if x's local state dictates that it needs to send a message.
- $\mathfrak{d}(x)$ implies that if a **snd** or **rcv** transition is enabled for the actor x, it will either eventually occur or it will be infinitely often enabled.

$$
\begin{array}{l}
\texttt{Prp-NF-Axm} \equiv \\
\forall p \in \mathbb{P} : \mathfrak{d}(p) \implies \\
\quad (\forall b \in \mathcal{B}, k \in \xi, v \in \mathcal{V}, T \in \mathcal{T}, i \in \mathbb{N}, a \in \mathbb{A}, Q \in \mathcal{Q} : \\
\qquad\qquad (\phi(p, b, Q, \varsigma(\rho(T, i))) \\
\qquad\qquad \vee\ \langle a, p, k, b, v\rangle \in \mathfrak{m}(\varsigma(\rho(T, i)))) \implies \\
\qquad\qquad\qquad\qquad\qquad\qquad (\mathfrak{a}(\varsigma(\rho(T, i)), p) \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \vee\ (\exists j \in \mathbb{N} : (j > i)\ \wedge\ \mathfrak{a}(\varsigma(\rho(T, j)), p))))
\end{array}
$$

$$
\begin{array}{l}
\texttt{Acc-NF-Axm} \equiv \\
\forall a \in \mathbb{A} : \mathfrak{d}(a) \implies \\
\quad (\forall p \in \mathbb{P}, k \in \xi, v \in \mathcal{V}, T \in \mathcal{T}, i \in \mathbb{N}, b \in \mathcal{B} : \\
\qquad\qquad ((\langle p, a, 1a, b, \bar{v}\rangle \in \eta^a_{\varsigma(\rho(T,i))}\ \wedge\ (\beta^a_{\varsigma(\rho(T,i))} < b)) \\
\qquad\qquad \vee\ (\langle p, a, 2a, b, v\rangle \in \eta^a_{\varsigma(\rho(T,i))}\ \wedge\ (\beta^a_{\varsigma(\rho(T,i))} \leq b)) \\
\qquad\qquad \vee\ (\langle p, a, k, b, v\rangle \in \mathfrak{m}(\varsigma(\rho(T, i))))) \implies \\
\qquad\qquad\qquad\qquad\qquad\qquad (\mathfrak{a}(\varsigma(\rho(T, i)), a) \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \vee\ (\exists j \in \mathbb{N} : (j > i)\ \wedge\ \mathfrak{a}(\varsigma(\rho(T, j)), a))))
\end{array}
$$

$$
\begin{array}{l}
\texttt{NF-IOE-Axm} \equiv \\
\forall x \in \mathcal{A} : \mathfrak{d}(x) \implies \\
\quad (\forall T \in \mathcal{T}, i \in \mathbb{N}, m \in \mathcal{M} : \\
\qquad ((m \in \mathfrak{m}(\varsigma(\rho(T, i)))\ \wedge\ \Re(\rho(T, i), x, \mathfrak{r}(x, m))) \implies \\
\qquad ((\exists j \in \mathbb{N} : (j \geq i) \\
\qquad\qquad\qquad \wedge\ \rho(T, j + 1) = \mathfrak{T}(\rho(T, j), \mathfrak{r}(x, m))) \\
\qquad\qquad \vee\ (\forall k \in \mathbb{N} : (k > i) \\
\qquad\qquad\qquad\qquad \implies\ (\exists j \in \mathbb{N} : (j \geq k) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ (m \in \mathfrak{m}(\varsigma(\rho(T, j)))\ \wedge\ \Re(\rho(T, j), x, \mathfrak{r}(x, m))))))) \\
\quad \wedge\ (\Re(\rho(T, i), x, \mathfrak{s}(x, m)) \implies \\
\qquad\qquad ((\exists j \in \mathbb{N} : (j \geq i) \\
\qquad\qquad\qquad\qquad \wedge\ \rho(T, j + 1) = \mathfrak{T}(\rho(T, j), \mathfrak{s}(x, m))) \\
\qquad\qquad\qquad \vee\ (\forall k \in \mathbb{N} : (k > i) \\
\qquad\qquad\qquad\qquad \implies\ (\exists j \in \mathbb{N} : (j \geq k)\ \wedge\ \Re(\rho(T, j), x, \mathfrak{s}(x, m))))))
\end{array}
$$

We then introduce a predicate þ that is true of a proposal number if and only if it satisfies the conditions P1 and P2 described informally in Section 2.1.

$$
\begin{array}{l}
\texttt{P1-P2-Def} \equiv \\
\forall b \in \mathcal{B} : \text{þ}(b) \iff \\
\quad (\forall p \in \mathbb{P}, T \in \mathcal{T}, i \in \mathbb{N}, v \in \mathcal{V}, a \in \mathbb{A} : \\
\qquad\qquad ((\langle p, a, 1a, b, \bar{v}\rangle \in \eta^a_{\varsigma(\rho(T,i))} \implies \beta^a_{\varsigma(\rho(T,i))} < b) \\
\qquad\qquad \wedge\ (\langle p, a, 2a, b, v\rangle \in \eta^a_{\varsigma(\rho(T,i))} \implies \beta^a_{\varsigma(\rho(T,i))} \leq b)))
\end{array}
$$

Finally, the conditions for formally guaranteeing progress state that – *in all fair transition paths, some nonfaulty proposer p will be eventually ready to propose some proposal number b, that will satisfy P1 and P2, to some quorum Q whose members are all nonfaulty.*

$$
\begin{array}{l}
\texttt{CND} \equiv \\
\forall T \in \mathcal{T} : \\
\quad (\exists i \in \mathbb{N}, p \in \mathbb{P}, b \in \mathcal{B}, Q \in \mathcal{Q} : \\
\qquad (\text{đ}(p) \ \wedge\ \text{þ}(b) \ \wedge\ (\forall a \in Q : (\text{đ}(a) \ \wedge\ \Re(\rho(T,i), p, \mathfrak{s}(p, \langle p, a, 1a, b, \bar{v}\rangle)))))))
\end{array}
$$

4.4 The Proof of Progress

To prove progress in Synod, it suffices to prove that eventually, at least one proposal number will be chosen by some quorum (Section 2). In our set of conditions CND, we have assumed that some proposer p will eventually propose a proposal number b, that will satisfy *P1* and *P2*, to a quorum Q. Our proof strategy is to show that eventually, p will learn that b has been chosen by all members of Q. Theorem 1 formally states our main progress guarantee while Lemma 1 and Lemma 2 state progress in Phase 1 and Phase 2 respectively.

Theorem 1. *Given* CND*, in all fair transition paths, eventually some proposer p will learn that some proposal number b has been chosen.*

$$
\begin{array}{l}
\texttt{Theorem-1} \equiv \\
\texttt{CND} \implies (\forall T \in \mathcal{T} : (\exists i \in \mathbb{N}, p \in \mathbb{P}, b \in \mathcal{B} : \text{Ł}(p, b, \varsigma(\rho(T,i)))))
\end{array}
$$

Lemma 1. *In a fair transition path, if eventually a nonfaulty proposer p becomes ready to propose a proposal number b, that satisfies P1 and P2, to a quorum Q whose members are all nonfaulty, then eventually p will receive promises from all members of Q for b.*

$$
\begin{array}{l}
\texttt{Lemma-1} \equiv \\
\forall T \in \mathcal{T}, i \in \mathbb{N}, p \in \mathbb{P}, b \in \mathcal{B}, Q \in \mathcal{Q} : \\
\quad (\text{đ}(p) \ \wedge\ \text{þ}(b) \ \wedge\ (\forall a \in Q : (\text{đ}(a) \ \wedge\ \Re(\rho(T,i), p, \mathfrak{s}(p, \langle p, a, 1a, b, \bar{v}\rangle))))) \\
\qquad\qquad \implies (\exists j \in \mathbb{N} : (j \geq i) \ \wedge\ \phi(p, b, Q, \varsigma(\rho(T,j))))
\end{array}
$$

Lemma 2. *In a fair transition path, if eventually a nonfaulty proposer p receives promises for a proposal number b, that satisfies P1 and P2, from a quorum Q whose members are all nonfaulty, then eventually p will receive votes from all members of Q for b.*

$$\texttt{Lemma-2} \equiv$$
$$\forall T \in \mathcal{T}, i \in \mathbb{N}, p \in \mathbb{P}, b \in \mathcal{B}, Q \in \mathcal{Q}:$$
$$(\text{đ}(p) \ \wedge \ \text{þ}(b) \ \wedge \ \phi(p, Q, \varsigma(\rho(T, i))) \ \wedge \ (\forall a \in Q : \text{đ}(a))) \implies (\exists j \in \mathbb{N} : (j \geq i) \ \wedge \ \Phi(p, b, Q, \varsigma(\rho(T, j))))$$

Given below are the proof sketches of Theorem 1, Lemma 1, and Lemma 2:

Theorem 1 ***Proof Sketch*** -

(1) By Lemma 1 and CND, some nonfaulty proposer p will eventually receive promises from some quorum Q, whose members are all nonfaulty, for some proposal number b that satisfies *P1* and *P2*.
(2) By Lemma 2, p will eventually receive votes from Q for b and learn that b has been chosen.

□

Lemma 1 ***Proof Sketch*** -

(1) By Prp-NF-Axm, NF-IOE-Axm, and F-Snd-Axm *prepare* messages from p will eventually be sent to all members of Q.
(2) By Acc-NF-Axm, NF-IOE-Axm, F-Rcv-Axm all members of Q will eventually receive the *prepare* messages.
(3) By P1-P2-Def, Snd-1b-Rul, and Acc-NF-Axm, each member of Q will eventually be ready to send *promise* messages to p.
(4) By Acc-NF-Axm, NF-IOE-Axm, and F-Snd-Axm the *promise* messages from each member of Q will eventually be sent.
(5) By Prp-NF-Axm, F-Rcv-Axm, and NF-IOE-Axm p will eventually receive the *promise* messages from all members of Q.

□

Lemma 2 ***Proof Sketch*** -

(1) By Snd-2a-Rul, and Prp-NF-Axm, p will eventually be ready to send *accept* messages to all members of Q with proposal number b.
(2) By Prp-NF-Axm, NF-IOE-Axm, and F-Snd-Axm, *accept* messages from p will eventually be sent to all members of Q.
(3) By Acc-NF-Axm, NF-IOE-Axm, F-Rcv-Axm all members of Q will eventually receive the *accept* messages.
(4) By P1-P2-Def, Snd-2b-Rul, and Acc-NF-Axm, each member of Q will eventually be ready to send *voted* messages to p.
(5) By Acc-NF-Axm, NF-IOE-Axm, and F-Snd-Axm the *voted* messages from each member of Q will eventually be sent
(6) By Prp-NF-Axm, F-Rcv-Axm, and NF-IOE-Axm p will eventually receive the *voted* messages from all members of Q.

□

We have formalized all the theory and proof sketches presented in this section using Athena. The proofs of Theorem 1, Lemma 1, and Lemma 2 have been mechanically verified for correctness. The high-level structures of the proofs were developed in a hierarchical manner consisting of well-connected steps. The SPASS [64] automatic theorem prover was then guided with appropriate premises for mechanically verifying each step (more details can be found in the companion technical report [50]). We have made extensive use of Athena's existing library

of natural number theory for reasoning about indexed points in transition paths. The complete proof consists of about 6000 lines of Athena code[10].

5 Related Work

Prior work on verification of Synod-related protocols exists in the literature. Prisco *et al.* [15] present a rigorous hand-written proof of safety for Paxos along with an analysis of time performance and fault tolerance. Chand *et al.* [12] provide a specification of Multi-Paxos in TLA$^+$ [31] and use TLAPS [14] to prove its safety. Padon *et al.* [48] have verified the safety property for Paxos, *Vertical Paxos* [34], *Fast Paxos* [33], and *Stoppable Paxos* [38] using deductive verification. Küfner *et al.* [28] provide a methodology to develop machine-checkable proofs of fault-tolerant round-based distributed systems and verify the safety property for Paxos. Schiper *et al.* [60] have formally verified the safety property of a Paxos-based totally ordered broadcast protocol using EventML [11] and the Nuprl [45] proof assistant. Howard *et al.* [23] have presented *Flexible Paxos* by introducing flexible quorums for Paxos and have model checked its safety property using the TLC model checker [32]. Rahli *et al.* [56, 57] have used EventML and Nuprl to formally verify the safety of an implementation of Multi-Paxos. Attiya *et al.* [8] provide bounds on the time to reach progress in consensus by assuming a synchronous model with known bounds on message delivery and processing time of non-faulty processes. Keidar *et al.* [26] consider a partial synchrony model with known bounds on processing times and message delays and use it to guarantee progress in a consensus algorithm when the bounds hold. Malkhi *et al.* [38] introduce Stoppable Paxos, a variant of Paxos for implementing a stoppable state machine and provide an informal proof of safety and progress for Stoppable Paxos with a unique leader. McMillan *et al.* [40] machine-check and verify the proofs of safety and progress properties of Stoppable Paxos [38] using Ivy [49]. Dragoi *et al.* [17] introduce PSync, a language that allows writing, execution, and verification of high-level implementations of fault-tolerant systems, and use it to verify the safety and progress properties of *LastVoting* [13]. LastVoting is an adaptation of Paxos in the *Heard-Of model* [13] that guarantees progress under the assumption of a single leader. A machine-checked proof of safety and progress of LastVoting also appeared in [16]. Hawblitzel *et al.* [19, 20] introduce a framework for designing provably correct distributed algorithms called IronFleet and use it to prove the safety and progress properties for a Multi-Paxos implementation called IronRSL by embedding TLA$^+$ specifications in Dafny [36]. Their proof of progress relies on the assumption that eventually, all messages will arrive within a maximum network delay and leader election will succeed.

All of the aforementioned work has either analyzed the safety property or both the safety and progress properties of Synod-related protocols. Where progress has been verified, the authors have either assumed a unique leader, synchrony, or both. Our work improves upon existing work by identifying a set of asynchronous conditions under which the fundamental Synod consensus protocol can

[10] Complete Athena code available at http://wcl.cs.rpi.edu/pilots/fvcafp

make eventual progress in the absence of a unique leader, and providing the first mechanically verified proof of eventual progress in Synod.

6 Conclusion

We have identified a set of sufficient conditions under which the Synod protocol can make progress, in asynchronous communication settings and in the absence of a unique leader. Leader election itself being a consensus problem, our conditions generalize Paxos' progress conditions by eliminating their cyclic reliance on consensus. Consequently, our weaker assumptions do not impose a communication bottleneck or proposal restrictions. We have introduced a failure-aware actor model (FAM) to reason about communication failures in actors. Using this reasoning framework we have formally demonstrated that eventual progress can be guaranteed in Synod under the identified conditions. Finally, we have used Athena to develop the first machine-checked proof of progress in Synod.

It is important to note that a guarantee of eventual progress only states that consensus will be achieved, but does not provide any bound on the amount of time that may be required for the same. Since air traffic data usually has a short useful lifetime and aircraft have limited time to remain airborne, a guarantee of eventual progress *alone* is insufficient for UAM applications. To be useful, a progress guarantee should have some associated time bounds that the aircraft can use to make important decisions, *e.g.*, if there is a guarantee that consensus will take at least 5 seconds, then a candidate can decide to only compute flight plans that start after 5 seconds. Nevertheless, we see this work as a valuable exercise in perceiving the nuances involved in guaranteeing eventual consensus in the presence of multiple unrestricted proposers. This is important because a guarantee of eventual progress is a necessary precondition for providing a guarantee of timely progress that can be directly applicable for UAM.

A potential direction of future work would be to investigate formal proofs of probabilistic guarantees of timely progress by using data-driven statistical results. Such properties may be provided by using statistical observations about message transmission and processing delays, which cannot be deterministically predicted in asynchronous conditions but can be observed at run-time. Another potential direction of work would be to model message loss in FAM by introducing additional meta-level transitions. This would allow us to weaken the conditions further by requiring the guaranteed delivery of only a subset of messages, thereby weakening the current fairness assumptions of FAM. To avoid livelocks, it may also suffice to replace predicate fairness with a weaker assumption that *infinitely often enabled finite transition sequences must eventually occur*.

Acknowledgment: This research was partially supported by the National Science Foundation (NSF), Grant No. – CNS-1816307 and the Air Force Office of Scientific Research (AFOSR), DDDAS Grant No. – FA9550-19-1-0054. The authors would like to express their gratitude to Elkin Cruz-Camacho, Dan Plyukhin, and the anonymous reviewers of NFM 2021 for their helpful comments on improving the manuscript.

References

1. Agha, G.: Actors: A Model of Concurrent Computation in Distributed Systems. The MIT Press (1986)
2. Agha, G., Mason, I.A., Smith, S., Talcott, C.: Towards a Theory of Actor Computation. In: International Conference on Concurrency Theory. pp. 565–579. Springer (1992)
3. Agha, G.A., Mason, I.A., Smith, S.F., Talcott, C.L.: A Foundation for Actor Computation. Journal of Functional Programming **7**(1), 1–72 (1997)
4. Alquraan, A., Takruri, H., Alfatafta, M., Al-Kiswany, S.: An Analysis of Network-Partitioning Failures in Cloud Systems. In: 13th USENIX Symposium on Operating Systems Design and Implementation. pp. 51–68 (2018)
5. Arkoudas, K.: Athena, `http://proofcentral.org/athena`
6. Arkoudas, K.: Simplifying Proofs in Fitch-Style Natural Deduction Systems. Journal of Automated Reasoning **34**(3), 239–294 (2005)
7. Arkoudas, K., Musser, D.: Fundamental Proof Methods in Computer Science: A Computer-Based Approach. MIT Press (2017)
8. Attiya, H., Dwork, C., Lynch, N., Stockmeyer, L.: Bounds on the Time to Reach Agreement in the Presence of Timing Uncertainty. Journal of the ACM (JACM) **41**(1), 122–152 (1994)
9. Aweiss, A.S., Owens, B.D., Rios, J., Homola, J.R., Mohlenbrink, C.P.: Unmanned Aircraft Systems (UAS) Traffic Management (UTM) National Campaign II. In: 2018 AIAA Information Systems-AIAA Infotech@ Aerospace, p. 1727 (2018)
10. Balachandran, S., Muñoz, C., Consiglio, M.: Distributed Consensus to Enable Merging and Spacing of UAS in an Urban Environment. In: 2018 International Conference on Unmanned Aircraft Systems (ICUAS). pp. 670–675. IEEE (2018)
11. Bickford, M., Constable, R.L., Rahli, V.: Logic of Events, a Framework to Reason About Distributed Systems. In: Languages for Distributed Algorithms Workshop (2012)
12. Chand, S., Liu, Y.A., Stoller, S.D.: Formal Verification of Multi-Paxos for Distributed Consensus. In: International Symposium on Formal Methods. pp. 119–136. Springer (2016)
13. Charron-Bost, B., Schiper, A.: The Heard-Of Model: Computing in Distributed Systems With Benign Faults. Distributed Computing **22**(1), 49–71 (2009)
14. Chaudhuri, K., Doligez, D., Lamport, L., Merz, S.: Verifying Safety Properties with the TLA+ Proof System. In: International Joint Conference on Automated Reasoning. pp. 142–148. Springer (2010)
15. De Prisco, R., Lampson, B., Lynch, N.: Revisiting the PAXOS Algorithm. Theoretical Computer Science **243**(1-2), 35–91 (2000)
16. Debrat, H., Merz, S.: Verifying Fault-Tolerant Distributed Algorithms in the Heard-Of Model. Archive of Formal Proofs **2012** (2012)
17. Drăgoi, C., Henzinger, T.A., Zufferey, D.: PSync: A Partially Synchronous Language for Fault-Tolerant Distributed Algorithms. In: ACM SIGPLAN Notices. vol. 51, pp. 400–415. ACM (2016)
18. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of Distributed Consensus With One Faulty Process. Journal of the ACM (JACM) **32**(2), 374–382 (1985)
19. Hawblitzel, C., Howell, J., Kapritsos, M., Lorch, J.R., Parno, B., Roberts, M.L., Setty, S., Zill, B.: IronFleet: Proving Practical Distributed Systems Correct. In: Proceedings of the 25th Symposium on Operating Systems Principles. pp. 1–17. ACM (2015)

20. Hawblitzel, C., Howell, J., Kapritsos, M., Lorch, J.R., Parno, B., Roberts, M.L., Setty, S., Zill, B.: IronFleet: Proving Safety and Liveness of Practical Distributed Systems. Communications of the ACM **60**(7), 83–92 (2017)
21. Hewitt, C.: Viewing Control Structures as Patterns of Passing Messages. Artificial Intelligence **8**(3), 323–364 (1977)
22. Hopkin, V.D.: Human Factors in Air Traffic Control. CRC Press (2017)
23. Howard, H., Malkhi, D., Spiegelman, A.: Flexible Paxos: Quorum Intersection Revisited. arXiv preprint arXiv:1608.06696 (2016)
24. Imai, S., Varela, C.A.: A Programming Model for Spatio-Temporal Data Streaming Applications. In: Dynamic Data-Driven Applications Systems. pp. 1139–1148. Omaha, NE, USA (2012)
25. Imai, S., Blasch, E., Galli, A., Zhu, W., Lee, F., Varela, C.A.: Airplane Flight Safety Using Error-Tolerant Data Stream Processing. IEEE Aerospace and Electronics Systems Magazine **32**(4), 4–17 (2017)
26. Keidar, I., Rajsbaum, S.: Open Questions on Consensus Performance in Well-Behaved Runs. In: Future Directions in Distributed Computing, pp. 35–39. Springer (2003)
27. Kirsch, J., Amir, Y.: Paxos for System Builders: An Overview. In: Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware. pp. 1–6 (2008)
28. Küfner, P., Nestmann, U., Rickmann, C.: Formal Verification of Distributed Algorithms. In: IFIP International Conference on Theoretical Computer Science. pp. 209–224. Springer (2012)
29. Lamport, L.: The Part-Time Parliament. ACM Transactions on Computer Systems (TOCS) **16**(2), 133–169 (1998)
30. Lamport, L.: Paxos Made Simple. ACM Sigact News **32**(4), 18–25 (2001)
31. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc. (2002)
32. Lamport, L.: Real-Time Model Checking is Really Simple. In: Advanced Research Working Conference on Correct Hardware Design and Verification Methods. pp. 162–175. Springer (2005)
33. Lamport, L.: Fast Paxos. Distributed Computing **19**(2), 79–103 (2006)
34. Lamport, L., Malkhi, D., Zhou, L.: Vertical Paxos and Primary-Backup Replication. In: Proceedings of the 28th ACM Symposium on Principles of Distributed Computing. pp. 312–313 (2009)
35. Lee, S.M., Park, C., Johnson, M.A., Mueller, E.R.: Investigating Effects of Well Clear Definitions on UAS Sense-And-Avoid Operations in Enroute and Transition Airspace. In: 2013 Aviation Technology, Integration, and Operations Conference. p. 4308 (2013)
36. Leino, K.R.M.: Dafny: An Automatic Program Verifier for Functional Correctness. In: International Conference on Logic for Programming Artificial Intelligence and Reasoning. pp. 348–370. Springer (2010)
37. Luckner, R., Höhne, G., Fuhrmann, M.: Hazard Criteria for Wake Vortex Encounters During Approach. Aerospace Science and Technology **8**(8), 673–687 (2004)
38. Malkhi, D., Lamport, L., Zhou, L.: Stoppable Paxos. Tech. rep., Microsoft Research (2008)
39. Manzano, M., Manzano, T.d.L.M.: Extensions of First-Order Logic, vol. 19. Cambridge University Press (1996)
40. McMillan, K.L., Padon, O.: Deductive Verification in Decidable Fragments with Ivy. In: International Static Analysis Symposium. pp. 43–55. Springer (2018)

41. Molisch, A.F., Tufvesson, F., Karedal, J., Mecklenbrauker, C.F.: A Survey on Vehicle-to-Vehicle Propagation Channels. IEEE Wireless Communications **16**(6), 12–22 (2009)
42. Musser, D.R., Varela, C.A.: Structured Reasoning About Actor Systems. In: Proceedings of the 2013 Workshop on Programming Based on Actors, Agents, and Decentralized Control. pp. 37–48. Agere! 2013, ACM, New York, NY, USA (2013)
43. Narkawicz, A., Muñoz, C., Dutle, A.: Coordination Logic for Repulsive Resolution Maneuvers. In: 16th AIAA Aviation Technology, Integration, and Operations Conference. p. 3156 (2016)
44. National Academies of Sciences, Engineering, and Medicine: Assessing the Risks of Integrating Unmanned Aircraft Systems (UAS) into the National Airspace System. The National Academies Press, Washington, DC (2018)
45. Naumov, P., Stehr, M.O., Meseguer, J.: The HOL/NuPRL Proof Translator. In: International Conference on Theorem Proving in Higher Order Logics. pp. 329–345. Springer (2001)
46. Okcu, H.: Operational Requirements of Unmanned Aircraft Systems Data Link and Communication Systems. Journal of Advances in Computer Networks **4**(1), 28–32 (2016)
47. Ongaro, D., Ousterhout, J.: In Search of an Understandable Consensus Algorithm. In: 2014 USENIX Annual Technical Conference (USENIX ATC 14). pp. 305–319 (2014)
48. Padon, O., Losa, G., Sagiv, M., Shoham, S.: Paxos made EPR: Decidable Reasoning About Distributed Protocols. Proceedings of the ACM on Programming Languages **1**(OOPSLA), 1–31 (2017)
49. Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: Safety Verification by Interactive Generalization. ACM SIGPLAN Notices **51**(6), 614–630 (2016)
50. Paul, S., Agha, G.A., Patterson, S., Varela, C.A.: Verification of Eventual Consensus in Synod using a Failure-Aware Actor Model. Tech. rep., Rensselaer Polytechnic Institute, Department of Computer Science (Mar 2021)
51. Paul, S., Kopsaftopoulos, F., Patterson, S., Varela, C.A.: Dynamic Data-Driven Formal Progress Envelopes for Distributed Algorithms. In: Dynamic Data-Driven Application Systems (InfoSymbiotics/DDDAS 2020). pp. 245–252 (2020)
52. Paul, S., Patterson, S., Varela, C.A.: Conflict-Aware Flight Planning for Avoiding Near Mid-Air Collisions. In: The 38th IEEE/AIAA Digital Avionics Systems Conference. pp. 1–10. San Diego, CA (2019)
53. Paul, S., Patterson, S., Varela, C.A.: Collaborative Situational Awareness for Conflict-Aware Flight Planning. In: The 39th IEEE/AIAA Digital Avionics Systems Conference. pp. 1–10 (2020)
54. Peters, A., Balachandran, S., Duffy, B., Smalling, K., Consiglio, M., Muñoz, C.: Flight Test Results of a Distributed Merging Algorithm for Autonomous UAS Operations. In: The 39th IEEE/AIAA Digital Avionics Systems Conference. pp. 1–7 (2020)
55. Queille, J.P., Sifakis, J.: Fairness and Related Properties in Transition Systems — A Temporal Logic to Deal with Fairness. Acta Informatica **19**(3), 195–220 (1983)
56. Rahli, V., Guaspari, D., Bickford, M., Constable, R.L.: Formal Specification, Verification, and Implementation of Fault-Tolerant Systems Using EventML. Electronic Communications of the EASST **72**, 1–15 (2015)
57. Rahli, V., Guaspari, D., Bickford, M., Constable, R.L.: EventML: Specification, Verification, and Implementation of Crash-Tolerant State Machine Replication Systems. Science of Computer Programming **148**, 26–48 (2017)

58. Ren, W., Beard, R.W.: Distributed Consensus in Multi-Vehicle Cooperative Control. Springer (2008)
59. Riazanov, A., Voronkov, A.: The Design and Implementation of VAMPIRE. AI communications **15**(2, 3), 91–110 (2002)
60. Schiper, N., Rahli, V., Van Renesse, R., Bickford, M., Constable, R.L.: Developing Correctly Replicated Databases Using Formal Tools. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 395–406. IEEE (2014)
61. Sommerville, I.: Software Engineering. Addison-Wesley/Pearson (2011)
62. Varela, C.A.: Programming Distributed Computing Systems. The MIT Press (2013)
63. Vascik, P.D., Hansman, R.J., Dunn, N.S.: Analysis of Urban Air Mobility Operational Constraints. Journal of Air Transportation **26**(4), 133–146 (2018)
64. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS Version 3.5. In: International Conference on Automated Deduction. pp. 140–145. Springer (2009)