Proceedings of the 56th Hawaii International Conference on System Sciences | 2023

Verification of a Distributed Ledger Protocol for Distributed Autonomous Systems using Monterey Phoenix

Nickolas Carter*
Naval Postgraduate School
nickolas.carter@acm.org

Duane Davis
Naval Postgraduate School
dtdavi1@nps.edu

Cynthia Irvine
Naval Postgraduate School
irvine@nps.edu

Peter Pommer*
Naval Postgraduate School
peter.pommer@acm.org

Abstract

Autonomous multi-vehicle systems are becoming increasingly relevant in military operations and have demonstrated potential applicability in civilian environments as well. A problem emerges, however, when logging data within these systems. In particular, potential loss of individual vehicles and inherently lossy and noisy communications environments can result in the loss of important mission data. This paper describes a novel distributed ledger protocol that can be used to ensure that the data in such a system survives and documents verification of the behavioral correctness of this protocol using informal verification methods and tools provided by the Monterey Phoenix project.

Key words: *autonomous systems, distributed ledger protocol, behavioral modeling, verification, informal verification*

1. Introduction

Use of unmanned aerial vehicles (UAVs) in both military and civilian environments has increased dramatically, and these systems have proven effective in roles for which manned systems are impractical or unacceptable. This trend is likely to continue as technology associated with autonomy, computational power, and vehicle endurance improves. In particular, these technologies support the development of swarming systems in which large numbers of vehicles operate cooperatively to accomplish complex tasks.

Maintenance of mission logs for swarm operations can be important for purposes of mission reconstruction, capability development, and performance analysis. The nature of these systems, however, can limit the completeness and accuracy of system-wide logs. Among the purported advantages of swarm systems are the relatively low cost of individual UAVs and the ability of the system to survive vehicle losses [1]. That is, the loss of a few vehicles will not lead to mission failure. Unfortunately, locally maintained data on failed vehicles will likely be unrecoverable. On the other hand, maintaining a complete system-wide log on every vehicle is unrealistic given the unreliable and bandwidth-limited communications architectures upon which multi-UAV systems typically rely [2].

We proposed the Unmanned Vehicle System Logging Protocol (UVSLP) to address this issue in [3] and documented its implementation and testing in [4] and [5]. Here, we document the verification of the protocol's claimed properties using the Monterey Phoenix (MP) behavioral modeling tool [6]. Contributions of this work include:

1. a blockchain-based distributed ledger protocol (DLP) suitable for mission log maintenance for an autonomous multi-UAV system utilizing a highly-constrained communications architecture,
2. verification of the protocol's mathematical properties using light-weight formal methods, and
3. demonstration of the suitability of the MP language and toolset for verification of complex, parallel processes.

Section 2 of this paper discusses the problem, its underlying assumptions, and the technologies upon which our protocol relies. Section 3 provides a summary of the proposed protocol, and Section 4 describes the MP tool. Section 5 describes MP's use in verifying the protocol's properties and discusses the verification test results. Finally, Section 6 discusses the outcomes and implications of this work.

2. Background

This section begins by describing the systems for which the UVSLP is intended and goes on to describe previous work leveraged in its development.

* Contributions to this work were performed while affiliated with the Naval Postgraduate School.

URI: https://hdl.handle.net/10125/103446
978-0-9981331-6-4
(CC BY-NC-ND 4.0)

2.1. System assumptions

Multi-vehicle autonomous systems in general and swarm systems in particular are inherently decentralized. Each unmanned vehicle (UV) can be thought of as an independent agent that processes and interprets its own sensor information, maintains its own "situational awareness," determines its own courses of action, and draws its own conclusions based on local observations. Information is shared with other agents only as required to facilitate collective objectives.

Swarm system advantages include their scalability and ability to gracefully address individual vehicle failures [1]. In fact, vehicle failures must not only be tolerated, they should be expected. This means that mission reconstruction and analysis requirements may necessitate the distribution of important information derived by individual agents among the swarm members to the maximum extent possible to prevent its loss.

Swarm system communications architectures can make information sharing challenging, however. During a mission, the network may become segmented in a highly dynamic manner as individual vehicles and groups of vehicles go in and out of communications with the rest of the swarm. Further, even within connected network segments, communication is frequently unreliable and often characterized as a set of *fair-loss links* over which messages are only probabilistically delivered [2, 7].

2.2. Distributed ledger protocols (DLPs) and unmanned vehicle (UV) swarms

Distributed ledger protocols (DLPs) synchronize data among distributed agents through a consensus process that ensures consistency across the entire network [8]. In a UV swarm system, a properly implemented DLP's decentralized storage can prevent data from being lost when individual UVs fail.

Blockchains are among the most common DLP forms and possess cryptographic features that can both ensure authenticity and facilitate consistency checks as new entries are added [9]. A DLP relies on blockchain consensus to ensure that blocks are stored in an agreed upon order. The UVSLP further leverages the blockchain consensus mechanism in the reconcile process described in Section 2.3.

Blockchain-based DLPs have been suggested for a number of multi-robot system issues including network and swarm security, inventory management, and task allocation [10]. These efforts primarily leverage the blockchain consensus mechanism's rejection of malicious or invalid information (this is particularly important if system or vehicle control relies on the information being maintained). In contrast, UVSLP is intended to improve the availability of mission records and makes a number of design tradeoffs that facilitate block distribution but sacrifice Byzantine failure detection. Research efforts that specifically focus on our choice of blockchain utilization are less common.

One effort that does closely align with the UVSLP's objectives is SwarmDAG [11]. Rather than utilizing a blockchain, the SwarmDAG protocol maintains data blocks in a directed acyclic graph to account for swarm network partitioning. Nodes within a partition maintain a partition-specific ledger fork of *confirmed* blocks. When network partitions merge, their respective ledger forks are merged as well. When an overall consensus (defined as two thirds of the swarm) is achievable, confirmed blocks are *finalized* and added to the formal SwarmDAG ledger.

Similar to UVSLP, SwarmDAG provides for an *eventually consistent* swarm-wide ledger that is derived from blocks maintained by individual participants [11]. Unlike UVSLP, data blocks are not exchanged beyond the partition in which they were created until they are finalized and formally added to the ledger. While the two-thirds consensus requirement does make Byzantine failure detection possible, it can result in data being lost if a participant fails before joining a partition large enough to finalize its locally maintained blocks. SwarmDAG also makes assumptions about block exchange success within partitions and ongoing awareness of swarm and partition membership that the UVSLP does not.

2.3. Distributed consensus

Consensus is among the most important requirements for a blockchain-based distributed ledger [9]. In most cases, this means that a majority of agents must agree before a proposed entry can be added to the blockchain. Unfortunately, consensus can only be guaranteed if the system satisfies specific conditions [12]. The UVSLP deals with this by relaxing the notion of consensus to agreement among a plurality of agents (i.e., a majority of currently available agents). This relaxation results in a set of *consistent* blockchains as opposed to a single *correct* blockchain. For purposes of the UVSLP, two blockchains are consistent if and only if they can be unified into a single agreed-upon blockchain through some reconciliation process [4, 5]. This allows the UVSLP to gracefully deal with network discontinuity by ensuring that local blockchains remain consistent and eventually unifiable.

The Paxos family of consensus algorithms is among those most heavily utilized for transactional data

DLPs [13]. Paxos assigns individual agents to proposer, acceptor, and learner roles. A proposer's locally generated entry is formally committed to the ledger by a learner if a majority of acceptors approve. Thus, Paxos maintains a ledger in which all entries have been approved by a majority of the accepting agents [13].

A direct implementation of Paxos is not possible for the envisioned swarm since the system described in Section 2.1 cannot guarantee the eventual availability of a majority. It does, however, form the basis of the UVSLP consensus algorithm in that the UVSLP relies on the three Paxos roles. Individual agents generate and propose blocks for addition to the blockchain (proposers). If a local majority of agents (i.e., among those responding to the proposal) agrees to the addition (acceptors), the proposing agent formally submits the block for addition to the locally maintained blockchains (learners) [3]. Since this approach relies upon a plurality rather than a majority, it is robust in the communications environment for which it is envisioned. In an ideal network, majority-based consensus can be reached, and a single unified blockchain is possible. In a lossy or disconnected system, the pluralities ensure that the locally maintained blockchains are at least consistent.

3. The Unmanned Vehicle System Logging Protocol (UVSLP)

The UVSLP is a blockchain protocol that is specified as a set of event handlers to be implented on each participating UV. Event handlers are triggered asynchronously by internal or external events. Data to be recorded in the distributed log is incorporated into blocks containing one or more log entries that are committed to blockchains maintained on each UV. Event handlers are organized into *block generation and commit* and *blockchain reconcile* components.

The UVSLP blockchain differs from most implementations in two ways. First, it does not attempt to construct a single, majority-approved blockchain. Rather, individual agents maintain possibly disparate blockchains and reconcile them as communications permit. Second, the blockchain is not additive only. During reconciliation, blocks can be removed from the reconciling vehicle's blockchain and re-added after extending a locally agreed-upon blockchain.

The UVSLP is specified with flow diagrams and language-independent pseudocode in [4] and [5] and was fully implemented on the Naval Postgraduate School (NPS) Advanced Robotic Systems Engineering Laboratory (ARSENL) swarm system [14]. It was incorporated into the system as a single Robot Operating System [15] node (i.e., process) written in Python.

The implementation was tested in the ARSENL software-in-the-loop (SITL) simulation environment [16] and in live-flight experiments as documented in [5]. SITL testing was conducted with various known packet-loss rates, and live-flight testing validated SITL environment results. Implementation experiments yielded consistent local blockchains (i.e., unifiable by post-flight application of the reconcile process), identified no protocol requirement violations, and provided empirical evidence of protocol correctness. They were not able to test network segmentation performance, however the MP validation documented here does account for segmentation.

The remainder of this section presents the ptotocol's formal properties and describes its components.

3.1. Protocol properties

Given the nature of the systems for which it was developed, the UVSLP must satisfy a number of general requirements. First, it must be compatible with the envisioned communications environment, meaning that it needs to work when communications are unreliable. Also, since each agent operates independently, the protocol must be fully implemented on every vehicle. The following properties formally describe the protocol's requirements [3]:

1. `Block creation`: No block will exist within the system that was not proposed by a participating agent.
2. `No block duplication`: No more than one copy of a particular block will be maintained by any agent at any time. A block can be present in the local blockchain or in a temporary local data structure associated with the protocol implementation.
3. `No block loss`: All blocks proposed by participating agents will be maintained by at least one agent (vehicle loss notwithstanding).
4. `Idle stop`: If an agent's protocol event handlers are in an idle state, then all blocks maintained by that agent must be present in the local blockchain.
5. `Block propagation`: In a fully connected system, all blocks will eventually be committed to all locally maintained blockchains.
6. `Uniform chain`: In a fully connected system where all agents have opportunities to reconcile blockchains, one uniform blockchain will emerge.

The first four properties relate to block generation and maintenance. They assert that blocks will not be erroneously created or duplicated, that they will not be lost once they have been added to the system, and that an individual vehicle's event handlers will not enter an idle state until all blocks maintained by that vehicle have been committed to the local blockchain. The final two properties relate to the distribution of blocks among the participating agents. In a fully connected system with no disjoint segments, every agent's local blockchain will eventually contain a copy of every block that has been added to the system. In addition, a uniform system-wide blockchain can eventually be obtained through repeated iterations of a reconciliation process.

Importantly, these properties do not require that a system-wide blockchain be available at any particular moment. They only require that a system-wide blockchain among surviving agents can be obtained in ideal circumstances. This follows from the relaxed notion of consensus presented in Section 2.3 and implies that, so long as locally maintained blockchains do not contradict one another, differences can eventually be resolved to obtain a single unified blockchain.

It is also worth noting that none of these properties impose specific security requirements. In fact, the UVSLP test implementation documented in [5] uses simple SHA3-256 hashes to ensure integrity of both the blockchain and the individual blocks. A specific implementation may require an agent-specific digital signature (e.g., RSA) in place of the block hash to prevent the addition of inauthentic blocks, and it may replace the blockchain hash with a shared-key message authentication code (MAC) such as hash-based MAC (HMAC) to ensure authenticity of the blockchain itself. The cryptographic specifics are not included in the protocol's specification, however. This means that cryptographic functionality must be built into a particular implementation as required or provided by the target platform's cryptographic implementation.

Further, since the protocol focuses on the availability of the data rather than its provenance, it does not address Byzantine failures (the ability to detect these failures is at best questionable given the connectivity and synchronicity assumptions of this work [12]). Nevertheless, while a malfunctioning or malicious participant might add invalid content to the log, satisfaction of the stated properties by all correct agents will prevent the loss or corruption of valid blocks.

3.2. Block generation and commit

Individual vehicles generate loggable events and add them to the distributed ledger through the *block generation and commit* process depicted in Figure 1. When an agent accumulates enough log entries to form a full block, the block is finalized and proposed to the other participants for addition to the blockchain. If a majority of the responding agents approve of the addition, the block is committed. If, on the other hand, a majority of the respondents reject the addition, the new block is locally pushed to a *reconcile stack*, and the *blockchain reconcile* process is initiated.

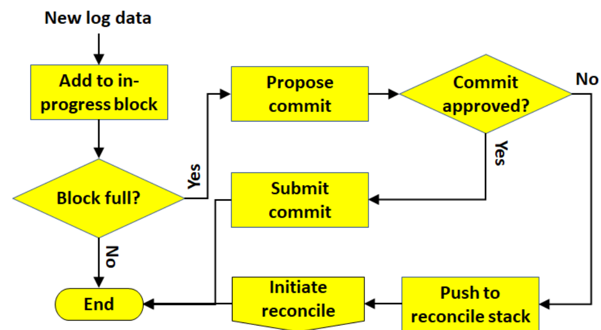

Figure 1. High-level depiction of the UVSLP block generation and commit process.

Commits are performed locally and remotely. Simply adding the block to the local blockchain completes the local commit, while remote commits are initiated by broadcasting a commit request. Receiving agents with local blockchains that match the requestor's precommit blockchain will commit the new block locally. Other agents will simply ignore the request.

3.3. Blockchain reconcile

Rejection of a proposed commit by a local majority indicates that the proposing agent's local blockchain has diverged from that of its neighbors. The *blockchain reconcile* process, see Figure 2, allows the agent to bring its blockchain into agreement with a plurality of its neighbors prior to reattempting the proposed commit.

The *blockchain reconcile* process is completed in three steps: *identify common blockchain*, *extend common blockchain*, and *commit reconcile stack*. The *identify common blockchain* step is applied recursively to identify the point of local blockchain divergence from the neighboring agents' blockchains. With each iteration, the neighbors are queried as to the presence of the local blockchain's high-order block (identified by its blockchain hash) anywhere in their local blockchains. If a majority of respondents possess the block, the point of divergence has been identified. Otherwise, the high-order block is removed from the local blockchain and pushed to a *reconcile stack*, and the process is repeated.

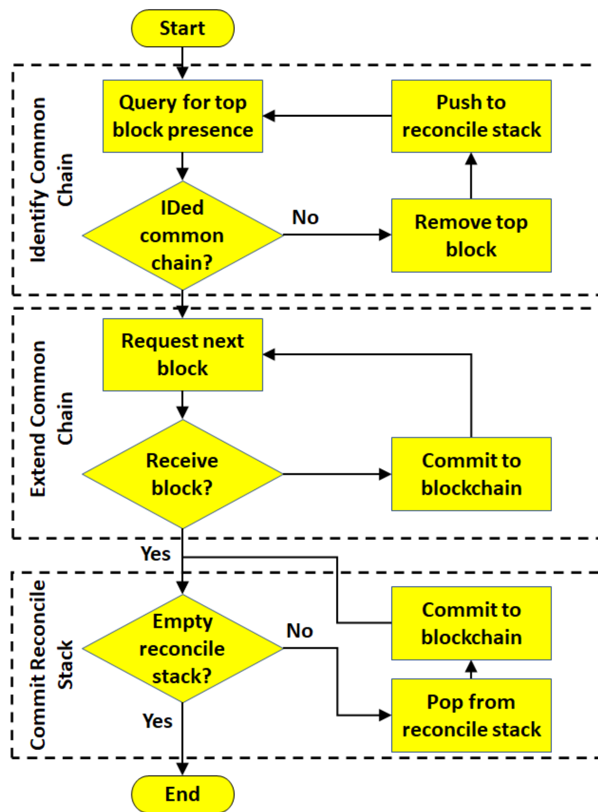

Figure 2. High-level depiction of the UVSLP blockchain reconcile process.

The *extend common blockchain* step is used to add missing blocks to the local blockchain. At each iteration, the reconciling agent requests the next block from the other participants (i.e., the block that they have adjacent to the current common chain high-order block). If responses are received, the reconciling agent adds the most common response block to its local blockchain and repeats the process. If not, the *blockchain reconcile* process proceeds to the *commit reconcile stack* step.

In the *commit reconcile stack* step, blocks are iteratively popped from the *reconcile stack* and committed to the blockchain locally and remotely as in the *blockchain generation and commit* process.

4. Monterey Phoenix (MP)

MP is a NPS-developed formal language and toolset for developing executable behavior models for systems, software, hardware, people, and organizations that capture their dependencies on one another and on the environment [6]. Its development was motivated by a desire to enable the detection, classification, prediction, and control of emergent behaviors arising from the interaction between individual processes or agents in complex systems [17].

MP system models deal with the behaviors of individual components separately from their interactions [18]. A *behavior* is formally defined as a collection of related events, set operations, and predicate logic that captures input requirements, decision points, and potential outcomes of those decisions. Activities that occur within the environment, on the other hand, are specified as sets of *interactions* between components. A specific interaction is represented as a sequence of events that occur as the a result of the interacting components' behaviors. MP uses Monte Carlo simulation to explore the combinatoric possibilities of the behaviors associated with interactions and provides results in the form of event traces [17]. This separation of component behaviors and their interactions leads to system models in which independent components interact without the imposition of assumptions about those interactions that might otherwise overconstrain the system [19].

MP's reliance on abstractions is premised on the *small scope hypothesis* [20] which surmises that a high proportion of errors can by identified by exhaustive evaluation of a process over a small scope of possible inputs. Stated differently, most errors can be exposed by testing with a small subset of the possible inputs. Properly defined models, therefore, amount to executable abstractions with which exhaustive traces provide what are referred to as lightweight proofs [21].

MP's lightweight proofs do not equate to formal mathematical proofs. Rather, they only verify that no errors exist for the small-scope testing. Thus, MP results are considered verification that an algorithm works correctly, not that it is completely error free [6]. Also, since MP uses an algorithm's abstraction, it cannot provide assurance about an algorithm's implementation.

MP has proven useful in analyzing a diverse range of scenarios involving complex interactions between agents. Among other applications, it has been used to identify emergent behaviors associated with business processes [22], layperson execution of first responder actions [23], and UAV search and rescue mission failure scenarios [24]. Its use in the validation of the UVSLP is a natural extension of its demonstrated utility in identifying unexpected behaviors in other types of complex systems. More complete descriptions of the MP vocabulary, syntax, semantics, and use are available in [6], [17], and [21].

5. Design validation

Verification of the UVSLP amounts to confirmation that it satisfies the properties specified in Section 3.1.

While MP's small-scope testing falls short of a formal proof, its use in the verification of other complex systems supports the assertion that it can provide assurances that the protocol behaves as desired in most situations and does not violate the requirements in any identifiable scenarios.

The requirement that the system include only legitimately proposed blocks (i.e., the `block creation` property) is only partially verifiable with MP since it is dependent on the security of the underlying system (i.e., on the authenticity of the data being logged). MP could be used to verify that blocks do not spontaneously appear in the system; however, this can be verified trivially by noting that blocks are *only* created by the *block generation and commit* event handlers as a result of log entries being submitted. Correct implementation of the protocol's event handlers, therefore, will preclude the generation of invalid blocks by the protocol itself [4].

Verification of the remaining properties is accomplished through testing with two MP models: a consensus model and a state model. This section discusses the use of the consensus model to verify the `no block duplication`, `no block loss`, `block propagation`, and `uniform chain` properties and the use of the state model to verify the `idle stop` property.

5.1. The UVSLP consensus model

The MP consensus model was derived from the UVSLP *blockchain reconcile* process and was developed to capture the semantics of the protocol's plurality-based approach to consensus.

5.1.1. Consensus model abstraction The UVSLP process abstraction upon which the MP consensus model was based is depicted in Figure 3. Each oval represents one or more event handlers, and each arrow represents generated events that trigger the next set of handlers. Event handlers are implemented across multiple UVs and include actions taken by the reconciling vehicle and results of actions taken by other vehicles. They are represented in the model from the perspective of the reconciling vehicle. Depicted event-based data flows represent local events (i.e., events triggered on the reconciling vehicle).

The implementation of Figure 3 as an MP model consists of five components (left side of the figure) and two data structures with which they interact (center and right side of the figure). The *phase 1 vote* component captures the *identify common blockchain* step of the

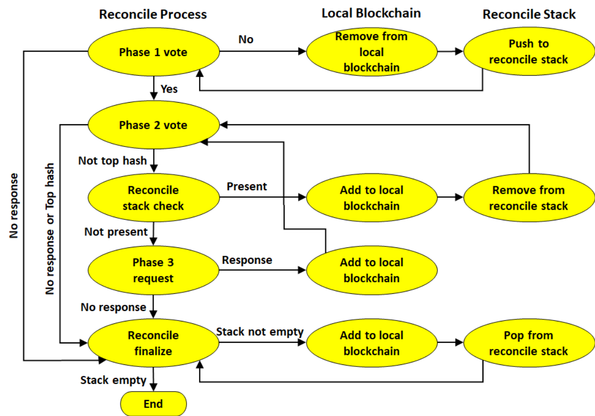

Figure 3. The UVSLP consensus process as an abstracted MP model.

blockchain reconcile process described in Section 3.3. Rather than implementing the voting process in the model, the MP exhaustive trace is used to account for all possible voting outcomes. In this case, the vote will indicate that the local blockchain's current high-order block is contained in a plurality of neighboring local blockchains (invoke the *phase 2 vote* component) or that it is not (move the block to the reconcile stack and reinvoke the *phase 1 vote* component).

The *phase 2 vote* component implements the first portion of the *extend common blockchain* step in which a block that can be added to the local blockchain is identified. The outcome of this vote will provide the block hash digest of a block to be added to the local blockchain (invoke the *reconcile stack check* component) or indicate that the local blockchain cannot be extended (invoke the *reconcile finalize* component).

If the local blockchain can be extended, the new block may already be available locally in the reconcile stack (i.e., if it had been previously added to and subsequently removed from the local blockchain). This check is accomplished by the *reconcile stack check* component. If the block is present in the reconcile stack, it is moved to the local blockchain, and the *phase 2 vote* component is invoked. If the block is not present, the *phase 3 request* component is invoked.

The *phase 3 request* component is used to obtain a missing block from a neighboring agent. One possible outcome of the request is that the block is obtained. In this case it is added to the local blockchain and the *phase 2 vote* component is invoked. It is also possible that no response is received (i.e., the agent from which it would be obtained no longer has communications with the requesting agent). In this case, the local blockchain cannot be extended further, and the *reconcile finalize* component is invoked.

The *reconcile finalize* component conducts the *commit reconcile stack* step of the *blockchain reconcile* process. It iteratively pops blocks from the reconcile stack and adds them to the local blockchain.

The MP model also includes three "universal behaviors" that describe how the model is to execute:

1. all pathways through the algorithm will terminate upon and only upon termination of the *reconcile finalize* component,
2. execution will not terminate unless the reconcile stack is empty, and
3. execution will always begin with invocation of the *phase 1 vote* component.

5.1.2. Consensus model property verification Exhaustive tracing of the consensus model by MP was used to verify UVSLP compliance with the `no block duplication`, `no block loss`, `block propagation`, and `uniform chain` properties. In keeping with the small scope hypothesis, the required number of iterations of each loop was limited to facilitate the analysis. An example corresponding to a trace in which the UV successfully reconciles is depicted in Figure 4. Yellow blocks in the diagram represent Figure 3 events while blue blocks indicate nondeterministic event outcomes. Diagrams for all traces are available in the appendix to [4].

To validate adherence to the `no block duplication` property, it must be established that no block was ever stored in the local blockchain more than once. Analysis of the data structures in which a block can be stored confirmed that this was the case. In the Figure 4 example, the initial *phase 1 vote* component outcome indicates that the top of the local blockchain had diverged, so a single block was removed from the local blockchain and pushed to the reconcile stack. The second *phase 1 vote* iteration indicated that the common blockchain had been reached. At this point, the manipulated block was being maintained in exactly one data structure (i.e., the reconcile stack).

The initial *phase 2 vote* indicated that a block could be added to the blockchain, and the outcome of the *phase 3 request* component indicated that it was obtained from another UV after verification that it was not already present in the reconcile stack. At this point the block was added directly to the local blockchain. Again, the block was present in exactly one data structure (i.e., the blockchain).

Subsequent invocation of the *phase 2 vote* component indicated that the local blockchain could not be extended further, so the *reconcile finalize* component moved the previously removed block from the reconcile stack back to the blockchain.

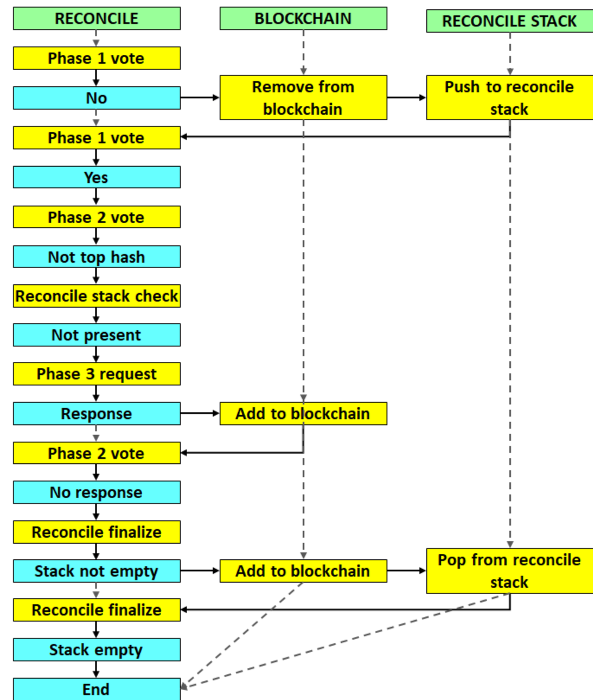

Figure 4. Example MP consensus model trace (events in yellow) of a successful reconcile.

Analyses of other consensus model execution traces yielded similar results and indicated that every event affecting a block did so in one of three ways: the block was moved from the blockchain to the reconcile stack, received from another vehicle and added to the blockchain, or moved from the reconcile stack to the blockchain. No result indicated that a block was ever present in more than one data structure at a time. Thus, the `no block duplication` property was satisfied in all traces and was evidently verified.

Similar analysis was conducted to verify the `no block loss` property. This property is verified by demonstrating that any block removed from the blockchain is eventually placed back into the blockchain and that any block received from another UV is eventually added to the blockchain. In the trace of Figure 4, a block was removed from the local blockchain following the first invocation of the *phase 1 vote* component. It was subsequently added back to the blockchain by the eventual invocation of the *reconcile finalize* component. A block was received from another UV following invocation of the *phase 3 request* component and immediately added to the local blockchain. Analysis of other execution traces provided

similar results indicating that the protocol consistently satisfied the `no block loss` property.

Use of the MP consensus model to verify the `block propagation` and `uniform blockchain` properties is predicated on an important assumption. That is, if it can be demonstrated that two network-adjacent UVs can be brought into unison, then the resulting blockchain can be inductively brought into unison with the rest of the agents in a connected network. With this in mind, verification of these properties reduces to verification for two UVs.

In that context, verification requires confirmation that a UV will eventually obtain all missing blocks from a neighboring UV's blockchain and that they will be added in the same location as in the neighbor's blockchain. This verification can be simplified by noting that it relies primarily on the traces corresponding to successful communication (i.e., the communications model implies that this trace will be realized eventually).

In the trace of Figure 4, the reconciling UV obtains a single block from a neighboring UV. The outcome of the second *phase 1 vote* invocation indicates that the high-order block of its local blockchain is contained in the neighboring UV's blockchain. Since the response to the *phase 3 request* component is the block adjacent to the local high-order block in the neighbor's blockchain, clearly it will be added to the reconciling UV's blockchain in the same location as the neighbor.

Additional analysis verified that this scenario was the only one in which a block received from another UV through the *blockchain reconcile* process was added to the local blockchain. Further, it should be noted that while the local blockchains of the two UVs from Figure 4 may not agree at the end of the trace (i.e., the second UV's blockchain might not contain blocks added by the *reconcile finalize* component), the disparity can be resolved by a subsequent *blockchain reconcile* process with the roles reversed. Thus, satisfaction of both the `block propagation` property and the `uniform blockchain` property is demonstrated by the MP consensus model.

5.2. The UVSLP state model

A state model was utilized to verify the `idle stop` property. This model was developed to capture the system states that start with the proposal of a new block in the *block generation and commit* process through the end of the *blockchain reconcile* process.

5.2.1. State model abstraction The UVSLP state model abstraction is depicted in Figure 5. As with Figure 3, each oval represents one or more UVSLP event handlers and arrows indicate possibly nondeterministic outcomes of those event handlers. Unlike the consensus model, the MP state model includes events associated with the agent being modeled and external events associated with other UVs.

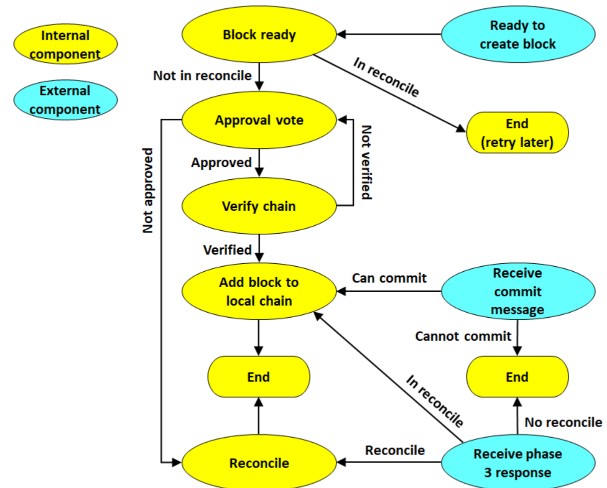

Figure 5. The MP UVSLP abstracted state model.

The MP implementation of Figure 5 consists of three external events (depicted in blue) that account for the ways in which new blocks can be received by a UVSLP agent and five internal components (depicted in yellow) that describe how received blocks are processed.

The *ready to create block* component represents the local generation of a block. When it is invoked, it subsequently invokes the *block ready* component to process the new block and propose it for addition.

The *block ready* component receives newly generated blocks and determines whether or not they can be processed. If the agent is not reconciling its local blockchain, the *approval vote* component is invoked. If, on the other hand, the agent is in the midst of a *blockchain reconcile*, the addition is not possible, and the trace ends without processing the new block. In this case, the protocol calls for the addition to be reattempted upon completion of the reconcile process. The abstract state model does not include this step, however, because it is logically identical to the *ready to create block* invocation.

The *approve vote* component represents the *block generation and commit* proposal and approval process. If the proposed addition is approved, the *verify chain* component is invoked. If the addition is rejected, the *reconcile component* is invoked.

The *verify chain* component accounts for a race condition arising from near-simultaneous proposal of block additions by multiple agents. If a proposing

vehicle processes a commit request from another agent before its own voting process is complete, its local blockchain will not match what had been approved. In this case, the *verify chain* will note the mismatch and reinvoke the *approval vote* component. If the local blockchain still matches what was proposed, the *add block to local chain* component is invoked.

The *add block to local chain* component finalizes the addition of a new block to the local blockchain and terminates the thread upon completion.

The external *receive commit message* component accounts for the receipt of a commit request from another agent (i.e., following completion of the sending agent's *block generation and commit* process). Upon initiation, the component determines if the commit can be completed based on the high-order block of the local blockchain. If the local blockchain matches the commit request, the *add block to local chain* component is invoked. If not, the received block is ignored (the request is not locally valid), and the thread ends.

The final external component, *receive phase 3 response*, accounts for receipt of new blocks from other agents as part of the *blockchain reconcile* process. If the local vehicle is in the midst of a reconcile and the block is received in response to its request, the *add block to local chain* component is invoked. If not, the *reconcile* component is invoked or the thread ends depending on whether or not the local blockchain can be improved through reconciliation with neighboring blockchains. Though not required, this feature improves protocol performance and was included in the abstraction.

MP state model execution begins with invocation of any of the three external components. A single trace can include asynchronous invocation of multiple external components effectively resulting in multiple execution threads. Each thread executes independently and ends upon completion of the behaviors associated with the *add block to local chain* or *reconcile* component or upon specific outcomes associated with the *block ready*, *receive commit message*, and *receive phase 3 response* components.

5.2.2. State model property verification

Exhaustive tracing of the MP state model was used to verify UVSLP satisfaction of the `idle stop` property. As with the consensus model, the execution scope of the state model traces was limited to facilitate MP execution and results analysis. Figure 6 depicts a trace in which the external *ready to create block*, *receive commit message*, and *receive phase 3 response* events were each invoked. The first two execution threads end with the individual invocations of the *add block to local chain* component, and the third ends upon completion of the *reconcile* component.

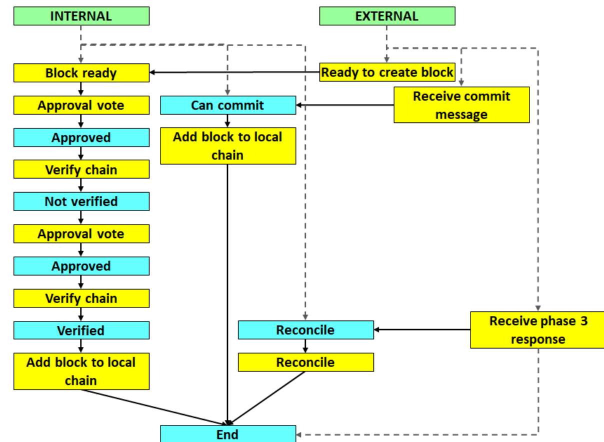

Figure 6. Example MP state model trace (events in yellow) with three external component invocations.

To demonstrate satisfaction of the `idle stop` property, it must be established that any blocks received by the UV are added to the local blockchain or discarded before the program terminates. In the Figure 6 example, three blocks were received over the course of the trace. The second was immediately added to the local blockchain, and the first was eventually added after invoking the *approval vote* component a second time. The third was presumably received as a byproduct of another agent's *blockchain reconcile* process and led to the invocation of the *reconcile* component. Completion of the reconcile process would include the addition of the newly received block to the local blockchain (per the `no block loss` property). Had the *receive phase 3 response* component determined that a reconcile was not required, the block would have been discarded.

Analysis of the other state model trace diagrams as described in [4] did not identify any `idle stop` violations. When addition of a new block to the local blockchain was appropriate, it was added. When addition of the new block to the local blockchain was not possible, that is when the local blockchain did not match the blockchain to which it was to be added or when it had not been requested, it was either discarded or accounted for by invocation of the *reconcile* component. Thus, UVSLP satisfaction of the `idle stop` property is demonstrated by the MP state model.

6. Conclusion

This paper provided an overview of the UVSLP, a distributed blockchain protocol for maintenance of system-wide logs by swarm systems operating

in unreliable or constrained communications environments. We examined the use of two executable MP UVSLP models to verify the satisfaction of mathematical properties presented as protocol requirements.

The MP models provided assurances that the UVSLP functioned correctly and that there were no apparent violations of the required properties. Although exhaustive MP traces do not amount to a formal proof, its use in verifying the UVSLP functionality provides a basis for confidence that the protocol is correct. Further, it provides an MP use case example that is generalizable to a large set of additional complex, parallel processes.

Acknowledgement This material was supported in part by the National Science Foundation under Agreement No 1565443. Any opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect the views of the National Science Foundation or the U.S. Government.

References

[1] J. Arquilla and D. F. Ronfeldt, *Swarming & the Future of Conflict*. Santa Monica, CA: Rand Corporation National Defense Research Institute (U.S.), 2000.

[2] X. Chen, J. Tang, and S. Lao, "Review of unmanned aerial vehicle swarm communication architectures and routing protocols," *Applied Sciences*, vol. 10, no. 10, p. 3661, 2020.

[3] N. Carter, P. Pommer, D. T. Davis, and C. E. Irvine, "Increasing log availability in unmanned vehicle systems," in *National Cyber Summit*, pp. 93–109, Springer, 2021.

[4] N. Carter, "Design and informal verification of a distributed ledger protocol for distributed autonomous systems using Monterey Phoenix," MS thesis, Naval Postgraduate School, Monterey, CA, December 2020.

[5] P. Pommer, "Design and implementation of a distributed ledger to support data survivability in an unmanned multi-vehicle system," MS thesis, Naval Postgraduate School, Monterey, CA, June 2021.

[6] M. Auguston, "Monterey Phoenix, or how to make software architecture executable," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pp. 1031–1040, 2009.

[7] D. T. Davis, T. H. Chung, M. R. Clement, and M. A. Day, "Consensus-based data sharing for large-scale aerial swarm coordination in lossy communications environments," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3801–3808, IEEE, 2016.

[8] M. Hancock and E. Vaizey, *Distributed Ledger Technology: Beyond Block Chain*. UK Government Chief Scientific Adviser, 2016.

[9] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain technology overview," *arXiv preprint arXiv:1906.11078*, 2019.

[10] T. Alladi, V. Chamola, N. Sahu, and M. Guizani, "Applications of blockchain in unmanned aerial vehicles: A review," *Vehicular Communications*, vol. 23, p. 100249, 2020.

[11] J. A. Tran, G. S. Ramachandran, P. M. Shah, C. B. Danilov, R. A. Santiago, and B. Krishnamachari, "Swarmdag: A partition tolerant distributed ledger protocol for swarm robotics," *Ledger*, vol. 4, no. Supp 1, pp. 25–31, 2019.

[12] D. Dolev, C. Dwork, and L. Stockmeyer, "On the minimal synchronism needed for distributed consensus," *Journal of the ACM (JACM)*, vol. 34, pp. 77–97, January 1987.

[13] L. Lamport, "Paxos made simple," *ACM SIGACT News (Distributed Computing Column)*, vol. 32, pp. 51–58, December 2001.

[14] T. H. Chung, M. R. Clement, M. A. Day, K. D. Jones, D. Davis, and M. Jones, "Live-fly, large-scale field experimentation for large numbers of fixed-wing uavs," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1255–1262, IEEE, 2016.

[15] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, no. 3.2 in 3, p. 5, Kobe, Japan, 2009.

[16] M. A. Day, M. R. Clement, J. D. Russo, D. Davis, and T. H. Chung, "Multi-uav software systems and simulation architecture," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 426–435, IEEE, 2015.

[17] K. Giammarco and M. Auguston, "Monterey Phoenix—behavior modeling approach for the early verification and validation of system of systems emergent behaviors," in *Engineering Emergence*, pp. 357–388, CRC Press, 2018.

[18] K. Giammarco, K. Giles, and C. A. Whitcomb, "Comprehensive use case scenario generation: An approach for modeling system of systems behaviors," in *2017 12th System of Systems Engineering Conference (SoSE)*, pp. 1–6, IEEE, 2017.

[19] K. Giammarco and M. Auguston, "Well, you didn't say not to! A formal systems engineering approach to teaching an unruly architecture good behavior," *Procedia Computer Science*, vol. 20, pp. 277–282, 2013.

[20] D. Jackson, *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[21] M. Auguston, "Monterey Phoenix system and behavior modeling language (version 4.0) user manual." `https://wiki.nps.edu/download/attachments/604667916/MP2-syntax-v4.pdf`, March 2020. [Accessed May 2022].

[22] M. Auguston, K. Giammarco, W. C. Baldwin, M. Farah-Stapleton, *et al.*, "Modeling and verifying business processes with Monterey Phoenix," *Procedia Computer Science*, vol. 44, pp. 345–353, 2015.

[23] J. Bryant, "Using Monterey Phoenix to analyze an alternative process for administering naloxone," *Capstone Research Project, Science and Math Academy, Aberdeen, MD*, 2016.

[24] M. Revill, "UAV swarm behavior modeling for early exposure of failure modes," MS thesis, Naval Postgraduate School, Monterey, CA, September 2016.