Anoma Research Topics | TECHNICAL REPORT

# Heterogeneous Paxos 2.0: the Specs

**Aleksandr Karbyshev**[a] **and Isaac Sheff**[a]

[a]Heliax AG

* *E-Mail: aleksandr@heliax.dev, isaac@heliax.dev*

## Abstract

We introduce *Heterogeneous Paxos 2.0*, a modified version of Heterogeneous Paxos. Similar to the original, our assumptions and guarantees remain consistent; however, our new version simplifies the algorithm logic, reduces bandwidth usage, and enables a more efficient implementation.

**Keywords:** heterogeneous Paxos ; distributed algorithm ; consensus ;

(Received Jun 17, 2024; Version: Jun 27, 2024)

## Contents

## 1. Introduction

We present a modified version of Heterogeneous Paxos protocol [SWvRM20]. Our assumptions and guarantees are similar to those of the original, but our new version simplifies algorithm logic, reduces communication complexity, and allows for a more efficient implementation. This new version, *Heterogeneous Paxos 2.0*, is formally specified in TLA$^+$ [Lam02] in the Typhon repository [Ano], but here we attempt to explain the protocol in more reader-friendly terms.

Our new protocol results from several distinct improvements to the original Heterogeneous Paxos. Applied together, they are Heterogeneous Paxos 2.0. The remainder of this introduction describes the main improvements.

## 1.1. Broadcast Primitive

We assume a *broadcast* primitive, which sends each message to each acceptor and learner. For liveness, this primitive must provide the guarantee that each message received by one honest (safe and live) acceptor is eventually received by all acceptors. To guarantee this, the original Heterogeneous Paxos had each acceptor echo well-formed messages to all other acceptors.

Instead, we leave the exact implementation of *broadcast* out of the Heterogeneous Paxos 2.0, allowing multiple possible implementation options. While echoing all messages would work, so would (for example) explicitly requesting unknown *refs* from a received message, which would (likely) require much less bandwidth.

## 1.2. 2a Messages

Instead of sending a *2a* message (Section 2.3) for each learner [SWvRM20], we send a single *2a* with a set of learners. Conceptually, this is similar to sending a set of *2a* messages, but in practice, it is more efficient, both to send and to track with *refs* (Section 2.3).

## 1.3. One Message In, at Most One Message Out

With the broadcast primitive and the *2a* messages, we can substantially simplify our protocol: In fact, we can remove all recursion and broadcast at most one message for each message received. This entails another minor change: instead of each actor receiving its own message in the same atomic action that it sends it (messages are broadcast, so actors receive their own messages), they receive them in some future action, just like any other message. This change, in turn, may increase implementation efficiency by breaking up atomic actions into smaller schedulable pieces.

## 1.4. Byzantine Behavior Detection

Each message specifies the previous message from the same sender (Section 2.3). This makes it much easier to detect certain kinds of Byzantine behavior: two messages referencing the same parent form a proof of misbehaviour. In contrast, the original protocol called for comparing transitive history sets [SWvRM20]. This change makes detecting Byzantine behaviour much easier to implement without sacrificing any guarantees.

## 2. Specification

### 2.1. Network Model

We assume the *closed-world* distributed system consisting of a fixed finite set of *acceptors* $\mathbb{A}$, a fixed finite set of proposers $\mathbb{P}$ and a fixed finite set of *learners* $\mathbb{L}$. The learners are uniquely identified by their learner IDs. We denote by $\mathcal{S} \subseteq \mathbb{A}$ a set of *safe*, non-Byzantine, acceptors that follow the protocol.

We assume that message broadcast is *reliable*, i.e., every message sent or received by a correct (safe and live) actor is eventually received by all live actors. The delivery order of sent messages does not have to be preserved.

### 2.2. Learner Graph

We use the notion of *learner graph* introduced in [SWvRM20]. We recap its formal definition below.

Let $\mathbb{L}$ be a fixed finite set of learners, and $\mathbb{A}$ a fixed finite set of acceptors.

- Nodes of learner graph are elements $\alpha \in \mathbb{L}$.
- Each learner $\alpha$ is labelled with a set $Q_\alpha$. The elements of $Q_\alpha$ are quorums $q \subseteq \mathbb{A}$. We assume that each $Q_\alpha$ is closed upwards, i.e., for any $q' \supseteq q \in Q_\alpha$, we have $q' \in Q_\alpha$.
- For any pair $\alpha, \beta \in \mathbb{L}$, there is a graph edge labelled with a set of quorums, $\alpha$–$\beta$, called a *safe set* of $\alpha$ and $\beta$. We assume that any safe set $\alpha$–$\beta$ is closed upwards, i.e., for any $q' \supseteq q \in \alpha$–$\beta$, we have $q' \in \alpha$–$\beta$.
- The graph is undirected, i.e., $\alpha$–$\beta = \beta$–$\alpha$.

**Definition 1** (Condensation). *We say that the learner graph is* condensed *if for all $\alpha, \beta, \gamma \in \mathbb{L}$*

$$\alpha\text{–}\beta \cup \beta\text{–}\gamma \subseteq \alpha\text{–}\gamma$$

**Definition 2** (Validity). *We say that the learner graph is* valid *if for any $s \in \alpha$–$\beta$, $q \in Q_\alpha$, and $r \in Q_\beta$ we have $s \cap q \cap r \neq \emptyset$.*

**Definition 3** (Entanglement). *We say that learners $\alpha$ and $\beta$ are* entangled *if the set of safe acceptors $\mathcal{S}$ belongs to the learners' safe set.*

$$Entangled(\alpha, \beta) \overset{\text{def}}{=} \mathcal{S} \in \alpha\text{–}\beta$$

## 2.3. Protocol Message Structure

- Each message is assigned one of the types: *1a*, *1b* or *2a*. We write $x : t$ to denote that message $x$ has type $t$.
- Each message $x$ contains a cryptographic signature that uniquely identifies the message signer, denoted by $Sig(x)$.
- Each message $x$, except for *1a*-messages, carries a finite list of references to some other messages, $x.refs$. In practice, each reference is represented by a hash of the referenced message. Acceptors remember previously received messages (*known_messages*), allowing them to resolve references. *1a*-messages do not reference any other messages.
- Each message $x$, except *1a*-messages, carries a distinguished reference, $x.prev$, to a previous message signed by the same signer, identified by $Sig(x)$. If $x$ is the first message sent by the sender, $x.prev$ contains a special non-reference value $\perp$.
- Each *2a*-message $x$ contains a list of learner IDs, $x.lrns$.
- Each *1a*-message $x$ has two specific fields: $x.val$ contains a proposed value, and $x.bal$ is a natural number specific to this proposal—its *ballot* number. We assume that ballots are value-specific: $x.bal = y.bal$ implies $x.val = y.val$. One way to implement this is to include the hash of the value in the (least significant bits of the) ballot.

## 2.4. Definitions

We extend $Sig()$ over sets of messages to mean the set of signers of those messages:

**Definition 4** (Message set signers)**.** *For any set of messages* $M$

$$Sig(M) \overset{\text{def}}{=} \{Sig(m) \mid m \in M\}$$

**Definition 5** (Transitive references)**.**

$$Tran(x) \overset{\text{def}}{=} \{x\} \cup \bigcup_{m \in x.refs} Tran(m)$$

**Definition 6** (Transitive previous)**.**

$$PrevTran(x) \overset{\text{def}}{=} \{x\} \cup \begin{cases} PrevTran(x.prev) & \text{if } x.prev \neq \perp \\ \emptyset & \text{otherwise} \end{cases}$$

**Definition 7** (Message ***1a***).

$$Get1a(x) \overset{\text{def}}{=} \underset{m:1a \in Tran(x)}{\operatorname{argmax}} \; m.bal$$

**Definition 8** (Ballot).

$$B(x) \overset{\text{def}}{=} Get1a(x).bal$$

**Definition 9** (Value).

$$V(x) \overset{\text{def}}{=} Get1a(x).val$$

**Definition 10** (Caught acceptors).

$$Caught(x) \overset{\text{def}}{=} Sig\left(\left\{ m \in Tran(x) \;\middle|\; \begin{array}{l} \exists m' \in Tran(x). \\ \quad Sig(m) = Sig(m') \\ \wedge\, m \notin PrevTran(m') \\ \wedge\, m' \notin PrevTran(m) \end{array} \right\}\right)$$

**Definition 11** (Connected learners). *For any learner $\alpha$ and message $x$*

$$Con_\alpha(x) \overset{\text{def}}{=} \{\beta \in \mathbb{L} \mid \exists s \in \alpha\text{–}\beta.\, s \cap Caught(x) = \emptyset\}$$

**Definition 12** (Buried ***2a***-messages). *For any learner $\alpha$, 2a-message $x$ and message $y$*

$$Buried_\alpha(x:2a, y) \overset{\text{def}}{=}$$
$$Sig\left(\left\{ m \in Tran(y) \;\middle|\; \begin{array}{l} \exists z \in Tran(m). \\ \quad z:2a \\ \wedge\, \alpha \in x.lrns \wedge \alpha \in z.lrns \\ \wedge\, B(z) > B(x) \\ \wedge\, V(z) \neq V(x) \end{array} \right\}\right) \in Q_\alpha$$

**Definition 13** (Connected ***2a***-messages). *For any learner $\alpha$ and message $x$*

$$Con2as_\alpha(x) \overset{\text{def}}{=} \left\{ m \in Tran(x) \;\middle|\; \begin{array}{l} \quad m:2a \\ \wedge\, Sig(m) = Sig(x) \\ \wedge\, \exists \beta \in Con_\alpha(x).\, \neg Buried_\beta(m, x) \end{array} \right\}$$

**Definition 14** (Freshness). *For any learner $\alpha$ and 1b-message $x$*

$$fresh_\alpha(x:1b) \overset{\text{def}}{=} \forall m \in Con2as_\alpha(x).\, V(m) = V(x)$$

**Definition 15** (Quorum of message). *For any learner $\alpha$, 2a-message $x$*

$$q_\alpha(x:2a) \overset{\text{def}}{=} Sig\big(\{m \in Tran(x) \mid m:1b \wedge fresh_\alpha(m) \wedge B(m) = B(x)\}\big)$$

**Definition 16** (Chain property)**.**

$$ChainRef(x) \overset{\text{def}}{=} x.prev \neq \bot \longrightarrow x.prev \in x.refs \land Sig(x.prev) = Sig(x)$$

**Definition 17** (Decision)**.** *For any learner $\alpha$ and set of messages $S$*

$$Decision_{\alpha}(S) \overset{\text{def}}{=} Sig(S) \in Q_{\alpha} \land \forall x, y \in S.\, x : 2a \land \alpha \in x.lrns \land B(x) = B(y)$$

*Any set of messages $S$ such that $Sig(S) \in Q_{\alpha}$ is called a* message quorum*. If $Decision_{\alpha}(S)$ holds, we shall write $V(S)$ to denote $V(x)$ for some message $x \in S$.*

**Definition 18** (Well-formedness)**.** *For any message $x$*

$$WellFormed1b(x) \overset{\text{def}}{=} \forall y \in Tran(x).\, x \neq y \land y \neq Get1a(x) \longrightarrow B(y) \neq B(x)$$
$$WellFormed2a(x) \overset{\text{def}}{=} x.lrns = \{\alpha \in \mathbb{L} \mid q_{\alpha}(x) \in Q_{\alpha}\}$$
$$\begin{aligned} WellFormed(x) \overset{\text{def}}{=}& \\ & ChainRef(x) \\ & \land (x : 1b \longrightarrow (\exists z \in x.refs.\, z : 1a) \land WellFormed1b(x)) \\ & \land (x : 2a \longrightarrow x.refs \neq \emptyset \land WellFormed2a(x)) \end{aligned}$$

## 2.5. Protocol

The formal specification of the protocol is formulated in PlusCal [Lam09] in the Typhon repository [Ano]. For better readability, we present the pseudocode of learner and acceptor algorithms in Figures 1 and 2, respectively. Like TLA$^{+}$ and PlusCal, we specify exactly which actions are *safe* for each actor. This does not rule out unnecessary or repetitive actions. Formally, each action is a *predicate* over state changes: the action is only deemed safe if every part of the predicate holds true. Actions can include other actions.

To ensure liveness, we require *weak fairness* [Lam02]: in any execution trace, if an action is safe for an infinite sequence, it eventually occurs.

Next, we describe the semantics of some particular instructions.

**Instruction Semantics.**

**`broadcast`(*z*)** When called, it sends message $z$ to every learner and acceptor, including the caller. Formally, this would mean that the defined state change is only allowed if the new state includes $z$ in the network.

**`assume`** ***P*** is a synonym of PlusCal's "`when`" instruction [Lam24]. The instruction, defined for Boolean state predicate $P$, restricts possible executions and should be considered as a guard à la Dijkstra [Dij75]: execution of the function containing "`assume` $P$" is only possible if the predicate evaluates to true during such a putative execution.

```
Acceptor::init():
  known_messages = {}
  recent_messages = {}
  prev_message = ⊥

Acceptor::process_1a($m$):
  assume $m.type$ = "1a"
  with $z$ = 1b(prev = prev_message, refs = recent_messages ∪ {$m$}):
    if $WellFormed1b(z)$:
      recent_messages = {$z$}
      prev_message = $z$
      broadcast($z$)

Acceptor::process_1b($m$):
  assume $m.type$ = "1b"
  with $\Lambda \subseteq \mathbb{L}$:
  with $z$ = 2a(prev = prev_message, refs = recent_messages ∪ {$m$}, lrns = $\Lambda$):
    assume $WellFormed2a(z)$
    recent_messages = {$z$}
    prev_message = $z$
    broadcast($z$)

Acceptor::process_2a($m$):
  assume $m.type$ = "2a"
  recent_messages ∪= {$m$}

Acceptor::receive($m$):
  assume $m \notin$ known_messages
  assume $\forall r \in m.refs.\ r \in$ known_messages
  known_messages ∪= {$m$}

Acceptor::process_message($m$):
  assume $WellFormed(m)$
  receive($m$)
  process_1a($m$) || process_1b($m$) || process_2a($m$)
```

**Figure 1.** Heterogeneous Paxos 2.0 acceptor specification.

```
Learner::init():
  known_messages = {}
  decision = ⊥

Learner::receive($m$):
  assume $m \notin$ known_messages
  assume $\forall r \in m.refs.\ r \in$ known_messages
  known_messages ∪= {$m$}

Learner::process_message($m$):
  assume $WellFormed(m)$
  receive($m$)

Learner::decide():
  with $s \subseteq$ known_messages:
    assume $Decision_{\mathtt{self}}(s)$
    decision = $V(s)$
```

**Figure 2.** Heterogeneous Paxos 2.0 learner specification.

**with** $x \in S$: $C$ is adopted from the PlusCal counterpart [Lam24]: an element $s$ is non-deterministically chosen from the given set $S$ and $C$ is executed with $x$ being locally defined to be equal to $s$. The instruction "with $x = v$: $C$" is equivalent to "with $x \in \{v\}$: $C$".

$C_1 \ || \ \ldots \ || \ C_k$ means a parallel execution of $C_i$, $i = 1, \ldots, k$.

## 2.6. Protocol Properties

**Theorem 19** (Validity)**.** *For any learner $\alpha$, and set of messages $s$*

$$Decision_\alpha(s) \implies \exists x.\, x : 1a \wedge V(s) = V(x)$$

*Proof.* Directly follows from the definitions of $B()$, $V()$ and $Decision()$. □

**Theorem 20** (Safety)**.** *Let the learner graph of the network be valid and condensed. Let $\alpha, \beta \in \mathbb{L}$ be learners such that Entangled$(\alpha, \beta)$. For any protocol execution and reachable network state, if $Decision_\alpha(s_\alpha)$ and $Decision_\beta(s_\beta)$ hold in that state, for some message quorums $s_\alpha$ and $s_\beta$, then $V(s_\alpha) = V(s_\beta)$.*

*Proof.* See TLA$^+$ protocol formalization in [Ano]. □

## 2.7. Mailbox Layer

The predicates `Acceptor::receive()` and `Learner::receive()` (Figures 1 and 2) define *causal message delivery*, i.e., the message can be received only once and only if all its direct references have already been received. In practice, this logic can be separated out into a special *mailbox* component. This component can be used by both local acceptor and learner, avoiding code duplication and allowing for more efficient message processing.

A mailbox implementation could be integrated into the broadcast implementation (Section 1.1): one mailbox might request missing messages from another when it receives a message $m$, but has not yet received $m$'s causal dependencies.

## 3. Future Work

Due to the recursive structure of protocol messages, a naïve implementation of the presented algorithm would lead to $\mathcal{O}(l \cdot n^5)$ complexity of message processing, where $l$ is the number of learners in the network, i.e., given number $n$ of messages received so far, the time required to process the following message is proportional in the worst case to $n^5$. In the ongoing work, we are developing an effective implementation of this algorithm using additional data structures that has a linear time complexity of message processing, $\mathcal{O}(l \cdot n)$. We are planning to formally prove that this implementation correctly realizes the proposed algorithm.

## References

Ano. Anoma. Typhon project github repository. https://github.com/anoma/typhon/tree/2ff6266f27986a6bec3d49fccc8f06697a9d163e. (cit. on pp. 2, 6, and 8.)

Dij75. Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18(8):453–457, 1975. (cit. on p. 6.)

Lam02. Leslie Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley, 2002. (cit. on pp. 2 and 6.)

Lam09. Leslie Lamport. The pluscal algorithm language. In Martin Leucker and Carroll Morgan, editors, *Theoretical Aspects of Computing - ICTAC 2009, 6th International Colloquium, Kuala Lumpur, Malaysia, August 16-20, 2009. Proceedings*, volume 5684 of *Lecture Notes in Computer Science*, pages 36–60. Springer, 2009. (cit. on p. 6.)

Lam24. Leslie Lamport. *A PlusCal User's Manual. C-Syntax Version 1.8*, 2024. (cit. on pp. 6 and 8.)

SWvRM20. Isaac Sheff, Xinwen Wang, Robbert van Renesse, and Andrew C. Myers. Heterogeneous paxos: Technical report, 2020. (cit. on pp. 1, 2, and 3.)