Heterogeneous Paxos

Isaac Sheff
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany
https://IsaacSheff.com
isheff@mpi-sws.org

Xinwen Wang
Cornell University, Ithaca, NY, USA
https://www.cs.cornell.edu/~xinwen/
xinwen@cs.cornell.edu

Robbert van Renesse
Cornell University, Ithaca, NY, USA
https://www.cs.cornell.edu/home/rvr/
rvr@cs.cornell.edu

Andrew C. Myers
Cornell University, Ithaca, NY, USA
https://www.cs.cornell.edu/andru/
andru@cs.cornell.edu

Abstract

In distributed systems, a group of *learners* achieve *consensus* when, by observing the output of some *acceptors*, they all arrive at the same value. Consensus is crucial for ordering transactions in failure-tolerant systems. Traditional consensus algorithms are homogeneous in three ways:

- all learners are treated equally,
- all acceptors are treated equally, and
- all failures are treated equally.

These assumptions, however, are unsuitable for cross-domain applications, including blockchains, where not all acceptors are equally trustworthy, and not all learners have the same assumptions and priorities. We present the first consensus algorithm to be heterogeneous in all three respects. Learners set their own mixed failure tolerances over differently trusted sets of acceptors. We express these assumptions in a novel *Learner Graph*, and demonstrate sufficient conditions for consensus.

We present *Heterogeneous Paxos*, an extension of Byzantine Paxos. Heterogeneous Paxos achieves consensus for any viable Learner Graph in best-case three message sends, which is optimal. We present a proof-of-concept implementation and demonstrate how tailoring for heterogeneous scenarios can save resources and reduce latency.

2012 ACM Subject Classification Computer systems organization → Redundancy; Computer systems organization → Availability; Computer systems organization → Reliability; Computer systems organization → Peer-to-peer architectures; Theory of computation → Distributed algorithms; Information systems → Remote replication

Keywords and phrases Consensus, Trust, Heterogeneous Trust

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2020.5

Related Version Technical Report at https://arxiv.org/abs/2011.08253 [47].

Supplementary Material Implementation source: https://github.com/isheff/charlotte-public

1 Introduction

The rise of blockchain systems has renewed interest in the classic problem of consensus, but traditional consensus protocols are not designed for the highly decentralized, heterogeneous environment of blockchains. In a Consensus protocol, processes called *learners* try to decide on the same value, based on the outputs of some set of processes called *acceptors*, some of

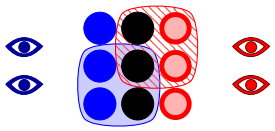

Figure 1 Illustration of the scenario in § 1.1. Blue learners are drawn as blue eyes, red learners as red, outlined eyes. Blue acceptors are drawn as blue circles, red acceptors as red, outlined circles, and third parties as black circles. The light solid blue region holds a quorum for the blue learners, and the striped red holds a quorum for the red learners.

whom may fail. (In our model, learners send no messages, and so they cannot fail.) Consensus is a vital part of any fault-tolerant system maintaining strongly consistent state, such as Datastores [14, 9], Blockchains [41, 20, 16], or indeed anything which orders transactions. Traditionally, consensus protocols have been *homogeneous* along three distinct dimensions:

- Homogeneous acceptors. Traditional systems tolerate some number f of failed acceptors, but acceptors are interchangeable. Prior work including "failure-prone sets" [38, 27] explores *heterogeneous* acceptors.
- Homogeneous failures. Systems are traditionally designed to tolerate either purely Byzantine or purely crash failures. There is no distinction between failure scenarios in which the same acceptors fail, but possibly in different ways. However, some projects have explored *heterogeneous*, or "mixed" failures [48, 13, 33].
- Homogeneous learners. All learners make the same assumptions, so system guarantees apply either to all learners, or to none. Systems with *heterogeneous* learners include Cobalt [36] and Stellar [39, 35, 21].

Blockchain systems can violate homogeneity on all three dimensions. Permissioned blockchain systems like Hyperledger [1], J.P. Morgan's Quorum [2], and R3's Corda [26] exist specifically to facilitate atomic transactions between mutually distrusting businesses. A crucial part of setting up any implementation has been settling on a set of equally trustworthy, failure-independent acceptors. These setups are complicated by the reality that different parties make different assumptions about whom to trust, and how.

Defining heterogeneous consensus poses challenges not covered by homogeneous definitions, particularly with respect to learners. How should learners express their failure tolerances? When different learners expect different possible failures, when do they need to agree? If a learner's failure assumptions are wrong, does it have any guarantees? No failure models developed for one or two dimensions of heterogeneity easily compose to describe all three.

Failure models developed for one or two dimensions of heterogeneity do not easily compose to describe all three, but our new trust model, the *Learner Graph* (§ 3), can express the precise trust assumptions of learners in terms of diverse acceptors and failures. Compared to trying to find a homogeneous setup agreeable to all learners, finding a learner graph for which consensus is possible is strictly more permissive. In fact, the learner graph is substantially more expressive than the models used in prior heterogeneous learner consensus work, including Stellar's *slices* [39] or Cobalt's *essential subsets* [36]. Building on our learner graph, we present the first fully *heterogeneous* consensus protocol. It generalizes Paxos to be heterogeneous along all three dimensions.

Heterogeneity allows acceptors to tailor a consensus protocol for the specific requirements of learners, rather than trying to force every learner to agree whenever any pair demand to agree. This increased flexibility can save time and resources, or even make consensus possible where it was not before, as we now show with an example.

1.1 Example

Suppose organizations Blue Org and Red Org want to agree on a value, such as the order of transactions involving both of their databases or blockchains. The people at Blue Org are *blue learners*: they want to decide on a value subject to *their* failure assumptions. Likewise, the people at Red Org are *red learners* with their own assumptions. While neither organization's learners believe their own organization's acceptors (machines) are Byzantine, they do not trust the other organization's acceptors at all. To help achieve consensus, they enlist three trustworthy third-party acceptors. Figure 1 illustrates this situation.

All learners want to agree so long as there are no Byzantine failures. However, no learner is willing to lose liveness (never decide on a value) if only one of its own acceptors has crashed, one third-party acceptor is Byzantine, and all the other organization's learners are Byzantine. Furthermore, learners within the same organization expect *never* to disagree, so long as none of their own organization's acceptors are Byzantine.

Unfortunately, existing protocols cannot satisfy these learners. Stellar [39], for instance, has one of the most expressive heterogeneous models available, but it cannot express heterogeneous failures. It cannot express blue and red learners' desire to terminate if a third-party acceptor crashes, but not necessarily agree a third-party acceptor is Byzantine. Our work enables a heterogeneous consensus protocol that satisfies all learners.

1.2 Heterogeneous Paxos

Heterogeneous Paxos, our novel generalization of Byzantine Paxos achieves consensus in a fully heterogeneous setting (§ 5), with precisely defined conditions under which learners are guaranteed safety and liveness. Heterogeneous Paxos inherits Paxos' optimal 3-message-send best-case latency, making it especially good for latency-sensitive applications with geodistributed acceptors, including blockchains. We have implemented this protocol and used it to construct several permissioned blockchains [21]. We demonstrate the savings in latency and resources that arise from tailoring consensus to specific learners' constraints.

1.3 Contributions

- The **Learner Graph** offers a general way to express heterogeneous trust assumptions in all three dimensions (§ 3).
- We **formally generalize the traditional consensus properties** (Validity, Agreement, and Termination) for the fully heterogeneous setting (§ 4).
- **Heterogeneous Paxos** is the first consensus protocol with heterogeneous learners, heterogeneous acceptors, and heterogeneous failures (§ 5). It also inherits Paxos' optimal 3-message-send best-case latency.
- **Experimental results** from our implementation of Heterogeneous Paxos demonstrate its use to construct permissioned blockchains with previously unobtainable security and performance properties (§ 6).

2 System Model

We consider a *closed-world* (or *permissioned*) system consisting of a fixed set of *acceptors*, a fixed set of *proposers*, and a fixed set of *learners*. Proposers and acceptors can send messages to other acceptors and learners. Some predetermined, but unknown set of acceptors are *faulty* (we assume a non-adaptive adversary). Faults include crash failures, which are not *live* (they can stop at any time without detection), and Byzantine failures, which are neither *live* nor *safe* (they can behave arbitrarily).

▶ **Definition 1** (Live). *A live acceptor eventually sends every message required by the protocol.*

▶ **Definition 2** (Safe). *A safe acceptor will not send messages unless they are required by the protocol, and will send messages only in the order specified by the protocol.*

Learners set the conditions under which they expect to agree. They want to decide values, and to be guaranteed agreement under certain conditions. While learners can make bad assumptions, since they do not send messages, they cannot misbehave, and so there are no "faulty learners."

Network. Network communication is point-to-point and reliable: if a live acceptor sends a message to another live acceptor, or to a learner, the message arrives. We adopt a slight weakening of *partial synchrony* [18]: after some unknown global stabilization time (GST), all messages between live acceptors arrive within some unknown latency bound Δ. In Heterogeneous Paxos, live acceptors send all messages to all acceptors and learners, but Byzantine acceptors may equivocate, sending messages to different recipients in different orders, with unbounded delays. We assume that messages carry effectively unbreakable cryptographic signatures, and that acceptors are identified by public keys. We also assume messages can **reference** other messages *by collision-resistant hash*: if one message contains a hash of another, it uniquely identifies the message it is referencing [42].

Consensus. The purpose of consensus is for each learner to decide on exactly one value, and for all learners to decide on the same value. Here, *execution* refers to a specific instance of consensus: the actions of a specific set of acceptors during some time frame. A *protocol* refers to the instructions that safe acceptors follow during an execution.

An execution of consensus begins when *proposers propose* candidate values, in the form of a message received by a correct acceptor. (No consensus can make guarantees about proposed values only known to crashed or Byzantine acceptors.) Proposers might be clients sending requests into the system. We make no assumptions about proposer correctness for safety properties, but to guarantee liveness, we will assume that acceptors can act as proposers as well (i.e. proposers are a superset of acceptors). After receiving some messages from acceptors, each learner eventually *decides* on a single value.

Traditionally, consensus requires three properties [19]:

- *Validity*: if a learner decides p, then p was proposed.[1]
- *Agreement*: if learner a decides value v, and learner b decides value v', then $v = v'$.
- *Termination*: all learners eventually decide.

In § 4, we generalize these properties to account for heterogeneity.

3 The Learner Graph

We characterize learners' failure assumptions with a novel construct called a *learner graph*. The learner graph is a general way to characterize trust assumptions for heterogeneous consensus. It can encompass most existing formulations, including Stellar's "slices" [39] and Cobalt's "essential sets" [36]. We discuss other formulations in § 7.

▶ **Definition 3** (Learner Graph). *A learner graph is an undirected graph in which vertices are learners, each labeled with the conditions under which they must terminate (§ 4.3 formally defines termination). Each pair of learners is connected by an edge, labeled with the conditions under which those learners must agree (§ 4.2 formally defines agreement).*

[1] Correia, Neves, and Veríssimo list several popular *validity* conditions. Ours corresponds to MCV2 [15]

3.1 Quorums

A *quorum* is a set of acceptors sufficient to make a learner decide: even if everything else has crashed [32], if a quorum are behaving correctly, a learner will eventually decide. In a learner graph, each learner a is labeled with a set of quorums Q_a. The learner requires termination precisely when at least one quorum are all live.

Within a specific execution, we assume some (unknown) set of pre-determined acceptors are actually *live*. We call this set $\mathcal{L}$.

3.2 Safe Sets

To characterize the conditions under which two learners want to agree, we need to express all possible failures they anticipate. Surprisingly, crash failures cannot cause disagreement: any disagreement that occurs when some acceptor has crashed could also occur if the same acceptor were correct, but very slow, and did not act until after the learner decided. Therefore, for agreement purposes, each tolerable failure scenario is characterized by a *safe set* (usually written s), the set of acceptors who are *safe*, meaning they act only according to the protocol. Between any pair of learners a and b in the learner graph, we label the edge between them with a set of safe sets a–b: so long as one of the safe sets in a–b indeed comprises only safe acceptors, the learners demand agreement.

Within a specific execution, we assume some (unknown) set of pre-determined acceptors are actually *safe*. We call this set $\mathcal{S}$. We do not require it, but systems often assume that $\mathcal{S} \subseteq \mathcal{L}$, since a Byzantine acceptor [31] may choose not to send messages.

3.2.1 Subset of Tolerable Failures

We generally assume that a subset of tolerable failures is always tolerated:

▶ **Assumption 4.** *Subset of failures properties:*
$$\begin{array}{l} \forall \ell \;.\; q_a \in Q_a \;\Rightarrow\; \ell \cup q_a \in Q_a \\ \forall x \;.\; s \in a\text{–}b \;\Rightarrow\; x \cup s \in a\text{–}b \end{array}$$

One might imagine, for example, two learners who demand agreement if two acceptors fail, but not if only one acceptor fails. However, we have no guarantee on time: if two acceptors are indeed faulty, one might act normally for an indefinite time, so the system would act as though only one has failed, and we will have to guarantee agreement.

3.2.2 Generalized Learner Graph Labels

It is possible to generalize the labels of learners and learner graph edges, and characterize quorums (conditions under which a learner must terminate) and safe sets (conditions under which pairs of learners must agree) as more detailed formal models (e.g., modeling network synchrony failures). All consensus failure models of which we are aware can be formalized using learner graphs with generalized labels. Heterogeneous Paxos works with any model of labels, so long as each label can be mapped (not necessarily uniquely) to a set of quorums for each learner, and a set of safe sets for each edge. For simplicity, in this work, we define labels as a set of quorums for each learner, and a set of safe sets for each edge.

3.3 Example

Consider our example from § 1.1 and Figure 1. All learners want to agree when all acceptors are safe. However, each learner demands termination (it must eventually decide on a value) even when one of its own acceptors has crashed, and one third part as well as all the

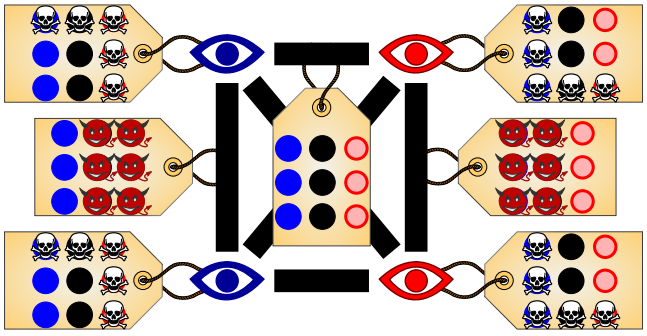

Figure 2 Learner Graph from § 3.3: Learners are eyes, with darker blue learners on the left, and outlined red learners on the right. Edge labels display *one* safe set for which the learners want to agree (unsafe acceptors are marked with a devil). The center label represents all edges between red and blue learners. Learner labels display *one* quorum for which the learner wants to terminate (crashed acceptors are marked with a skull). In each label, blue acceptors are blue circles, red acceptors are red, outlined circles, and third-party acceptors are black circles.

other organization's acceptors have failed as well. Furthermore, learners within the same organization expect *never* to disagree, so long as none of their own organization's acceptors are Byzantine: neither organization tolerates the other, or third-party acceptors, creating internal disagreement. In Figure 2, we diagram the learner graph. For space reasons, we draw each label with only *one* quorum or *one* safe set.

3.4 Agreement is Transitive and Symmetric

Agreement (formally defined in § 4.2) is symmetric, so learner graphs are undirected (a–$b = b$–a). Agreement is also transitive: if a agrees with b and b agrees with c, then a agrees with c. As a result, a and c must agree whenever both the conditions a–b and b–c are met. When learners' requirements reflect this assumption, we call the resulting learner graph *condensed*. We describe how to condense a learner graph in § 3.5 of [47].

▶ **Definition 5** (Condensed Learner Graph (CLG)). *A learner graph G is condensed iff:* $\forall a, b, c.\ (a\text{–}b \cap b\text{–}c) \subseteq a\text{–}c$

Self-Edges. A CLG describes when a learner a agrees with itself (i.e., if it decides twice, both decisions must have the same value): a–a.

▶ **Lemma 6** (Self-agreement). *A learner must agree with itself in order to agree with anyone:* $a\text{–}b \subseteq a\text{–}a$

Proof. Follows from Definition 5, and the fact that the CLG is undirected (§ 3.4) ◀

3.5 Liveness Bounds from Safety

Given the conditions under which learners want to agree, we can derive a (sufficient) bound on the quorums they require to terminate. In other words, given labels for the edges in the learners graph, we can bound the labels for the vertices.

As we will cover in more detail in § 5.1, each of a learner's quorums must intersect its neighbors quorums at a safe acceptor. As a result, we can construct a sufficient set of quorums for each learner in a CLG as follows: for each edge of the learner, each quorum includes a majority of acceptors from a each of the safety sets.

3.6 Safety Bounds from Liveness

Given the conditions under which learners want to terminate, we can derive a (necessary) bound on the safe sets they can require on each of their edges. As we will cover in more detail in § 5.1, each of a learner's quorums must intersect its neighbors quorums at a safe acceptor. As a result, safe sets can be assembled for each edge in a CLG as follows: each set includes one acceptor from the intersection of each pair of quorums (one from each learner).

4 Heterogeneous Consensus

We now define our novel heterogeneous generalization of traditional consensus properties.

4.1 Validity

Intuitively, a consensus protocol shouldn't allow learners to always decide some predetermined value. Validity is the same in heterogeneous and homogeneous settings.

▶ **Definition 7** (Heterogeneous Validity).

- *A consensus execution is valid if all values learners decide were proposed in that execution.*
- *A consensus protocol is valid if all possible executions are valid.*

4.2 Agreement

Our generalization of Agreement from the homogeneous setting to a heterogeneous one is the key insight that makes our conception of heterogeneous consensus possible. It generalizes not only the traditional homogeneous approach, but also the "intact nodes" concept from Stellar [39], and "linked nodes" from Cobalt [36].

▶ **Definition 8** (Entangled). *In an execution, two learners are entangled if their failure assumptions matched the failures that actually happen:* $\mathit{Entangled}(a, b) \triangleq \mathcal{S} \in a\text{–}b$

In the example (§ 1.1), if one third-party acceptor were Byzantine, the blue learners would be entangled with each other, and similarly with the red learners, but no blue learners would be entangled with red learners. It is possible for failures to divide the learners into separate groups, which may then decide different values even if they agree among themselves.

▶ **Definition 9** (Heterogeneous Agreement).

- *Within an execution, two learners have agreement if all decisions for either learner have the same value.*
- *A heterogeneous consensus protocol has agreement if, for all possible executions of that protocol, all entangled pairs of learners have agreement.*

In Heterogeneous Paxos, as in many other protocols, learners decide on a value whenever certain conditions are met for that value: learners can even decide multiple times. If there aren't too many failures, a learner is guaranteed to decide the same value every time. Because learners send no messages, they cannot *fail*, but they can make incorrect assumptions. Within the context of an execution, entanglement neatly defines when a learner is *accurate*, meaning it cannot decide different values.

▶ **Definition 10** (Accurate Learner). *is entangled with itself:* $\mathit{Accurate}(a) \triangleq \mathit{Entangled}(a, a)$

In the example (§ 1.1), if one third-party acceptor were Byzantine, then the blue and red learners would be accurate, but if a blue acceptor were also Byzantine, the blue learners would not be accurate (although the red learners would still be accurate).

```
acceptor_initial_state:
 known_messages = {}
 recently_received = {}

acceptor_on_receipt(m):
 for r ∈ m.refs:
  while r ∉ known_messages:
   wait()
 atomic:
  if m ∉ known_messages:
   forward m to all acceptors and learners
   recently_received ∪= {m}
   known_messages ∪= {m}
   if m has type 1a:
     z = new 1b(refs = recently_received)
     recently_received = {}
     on_receipt(z)
   if m has type 1b and b(m) == max_{x∈known_messages} b(x)
     for learner ∈ learners:
       z = new 2a(refs = recently_received, lrn = learner)
       if WellFormed(z):
         recently_received = {}
         on_receipt(z)
```

```
learner_initial_state:
  known_messages = {}

learner_on_receipt(m):
 for r ∈ m.refs:
  while r ∉ known_messages:
   wait()
 known_messages ∪= {m}
 for S ⊆ known_messages:
  if Decision_self(S ∪ {m}):
    decide(V(m))
```

Figure 3 Pseudo-code for Acceptor (left) and Learner (right). § 5 defines message structure (§ 5.2), *WellFormed* (Assumption 26), b() (Definition 19), V() (Definition 20), and Decision() (Definition 21).

4.3 Termination

Termination has no well agreed-upon definition for the heterogeneous setting, as it does not generalize easily from the homogeneous one. A heterogeneous consensus protocol is specified in terms of the (possibly differing) conditions under which each learner is guaranteed termination (§ 3). For example, in our prior work on Heterogeneous Fast Consensus, we distinguish between "gurus," learners with accurate failure assumptions, and "chumps," who hold inaccurate assumptions [45]; Stellar calls them "intact" and "befouled" [39]. When discussing termination properties, we use the following terminology:

▶ **Definition 11** (Termination).

- *Within an execution, a learner has termination if it eventually decides.*
- *A heterogeneous consensus protocol has termination if, for all possible executions of that protocol, all learners with a safe and live quorum have termination.*

Protocols can only guarantee termination under specific network assumptions, and varying notions of "eventually" [19, 29, 40]. Following in the footsteps of Dwork et al. [18], Heterogeneous Paxos guarantees Validity and Agreement in a fully asynchronous network, and termination in a partially synchronous network (Assumption 31). Furthermore, as in all other consensus protocols, if there are too many acceptor failures, some learners may not terminate. Specifically, a learner will decide (terminate) if at least one of its quorums is live.

▶ **Definition 12** (Terminating Learner). *has a live, safe quorum:* $Terminating(a) \triangleq \mathcal{L} \cup \mathcal{S} \in Q_a$

5 Heterogeneous Paxos

Heterogeneous Paxos is a consensus protocol (§ 2) based on Byzantine Paxos, Lamport's Byzantine-fault-tolerant [31] variant of Paxos [28, 29] using a simulated leader [30]. This protocol is conceptually simpler than *Practical Byzantine Fault Tolerance* [10]. When all learners have the same failure assumptions, Heterogeneous Paxos is *exactly* Byzantine Paxos.

Byzantine Paxos was originally written as a sequence of changes from crash-tolerant Paxos [30, 28]. We were able to construct a complete version of Byzantine Paxos in such a way that we could describe Heterogeneous Paxos with only a few additions, highlighted in pale blue. To our knowledge, without the portions highlighted in pale blue this is also the most direct description of the Byzantine Paxos via Simulated Leader protocol in the literature. Figure 3 presents pseudocode for Heterogeneous Paxos acceptors and learners.

Informally, Heterogeneous Paxos proceeds as a series of (possibly overlapping) *phases* corresponding to three types of messages, traditionally called *1a*, *1b*, and *2a*:

- Proposers send *1a* messages, each carrying a value and unique *ballot number* (stage identifier), to acceptors.
- Acceptors send *1b* messages to each other to communicate that they've received a *1a* (line 15 of Figure 3).
- When an acceptor receives a *1b* message for the highest ballot number it has seen from a learner a's *quorum* of acceptors, it sends a *2a* message labeled with a and that ballot number (line 20 of Figure 3). There is one exception ($WellFormed$ in Figure 3): once a safe acceptor sends a *2a* message m for a learner a, it never sends a *2a* message with a different value for a learner b, unless:
  - It knows that a quorum of acceptors has seen *2a* messages with learner a and ballot number higher than m.
  - Or it has seen Byzantine behavior that proves a and b do not have to agree.
- A learner a *decides* when it receives *2a* messages with the same ballot number from one of its quorums of acceptors (line 11 on the right of Figure 3).

Proposers can restart the protocol at any time, with a new ballot number. Acceptor and Learner behavior in Heterogeneous Paxos is described in Figure 3. We now describe their sub-functions, including message construction (§ 5.2), $WellFormed$ (Assumption 26), b() (Definition 19), V() (Definition 20), and Decision() (Definition 21).

Key Insight. Intuitively, Heterogeneous Paxos operates much like Byzantine Paxos, except that all acceptors execute the final phase separately for each learner. The shared phases allow learners to agree when possible, while the replicated final phase allows different learners to decide under different conditions. § 8 of [47] describes several heterogeneous consensus scenarios, as well as quorums for each learner.

5.1 Valid Learner Graph

Naturally, there are bounds on the learner graphs for which Heterogeneous Paxos can provide guarantees. Unlike traditional consensus, in a Heterogeneous Consensus learner graph, each learner a has its own set of quorums Q_a. These describe the learner's termination constraints: it may not terminate if all of its quorums contain a non-live acceptor (Definition 12). The notion of a *valid* learner graph generalizes the homogeneous assumption that every pair of quorums have a safe acceptor in their intersection.

Homogeneous Byzantine Paxos guarantees agreement (§ 4.2) when all pairs of quorums have ≥ 1 safe acceptor in their intersection. The heterogeneous case has a similar requirement:

▶ **Definition 13** (Valid Learner Graph). *A learner graph is valid iff for each pair of learners a and b, whenever they must agree, all of their quorums feature at least one safe acceptor in their intersection:* $s \in a\text{–}b \wedge q_a \in Q_a \wedge q_b \in Q_b \Rightarrow q_a \cap q_b \cap s \neq \emptyset$

5.2 Messaging

Acceptors send messages to each other. Live acceptors echo all messages sent and received to all other acceptors and learners, so if one live acceptor receives a message, all acceptors eventually receive it. When safe acceptors receive a message, they process and send resulting messages specified by the protocol atomically: they do not receive messages between sending results to other acceptors. Safe acceptors also receive any messages they send to themselves immediately: they receive no other messages between sending and receiving.

Each message x contains a cryptographic signature allowing anyone to identify the signer:

▶ **Definition 14** (Message Signer). $Sig(x : message) \triangleq$ *the acceptor or proposer that signed* x

We can define Sig() over sets of messages, to mean the set of signers of those messages:

▶ **Definition 15** (Message Set Signers). $Sig(x : set) \triangleq \{\ Sig(m) \mid m \in x\ \}$

Furthermore, each message x carries references to 0 or more other messages, $x.refs$. These references are by hash, ensuring both the absence of cycles in the reference graph and that it is possible to know exactly when one message references another [42]. In each message, safe acceptors reference each message they received since the last message they sent. Since all messages sent are sent to all acceptors, and safe acceptors receive messages sent to themselves immediately, each message a safe acceptor sends *transitively* references all messages it has ever sent or received. Safe acceptors delay receipt of any message until they have received all messages it references. This ensures they receive, for example, a $1a$ for a given ballot before receiving any $1b$s for that ballot.

Each message has a unique ID and an identifiable type: $1a, 1b$, or $2a$. A $2a$ message x has one type-specific field: $x.lrn$ specifies a learner. A $1a$ message y has two type-specific fields: $y.value$ is a proposed value, and $y.ballot$ is a natural number specific to this proposal.

We assume that each $1a$ has a unique ballot number, which could be accomplished by including signature information in the least significant bits of the ballot number:

▶ **Assumption 16** (Unique ballot assumption). $z : 1a \wedge y : 1a \wedge z.ballot = y.ballot \Rightarrow z = y$

5.3 Machinery

To describe Heterogeneous Paxos, we require some mathematical machinery.

Transitive References. We define $\text{Tran}(x)$ to be the transitive closure of message x's references. Intuitively, these are all the messages in the "causal past" of x.

▶ **Definition 17.** $Tran(x) \triangleq \{x\} \cup \bigcup_{m \in x.refs} Tran(m)$

Get1a: It is useful to refer to the $1a$ that started the ballot of a message: the highest ballot number $1a$ in its transitive references.

▶ **Definition 18.** $Get1a(x) \triangleq \underset{m:\, 1a \in Tran(x)}{\text{argmax}}\ m.ballot$

Ballot Numbers. The ballot number of a $1a$ is part of the message, and the ballot number of anything else is the highest ballot number among the $1a$s it (transitively) references.

▶ **Definition 19.** $b(x) \triangleq Get1a(x).ballot$

Value. The value of a $1a$ is part of the message, and the value of anything else is the value of the highest ballot $1a$ among the messages it (transitively) references.

▶ **Definition 20.** $V(x) \triangleq Get1a(x).value$

Decisions. A learner decides when it has observed a set of $2a$ messages with the same ballot, sent by a quorum of acceptors. We call such a set a *decision*:

▶ **Definition 21.** $Decision_a(q_a) \triangleq Sig(q_a) \in Q_a \wedge \forall\{x,y\} \subseteq q_a.\ b(x) = b(y) \wedge x.lrn = a \wedge x : 2a$

Messages in a decision share a ballot (and therefore a value), so we extend our value function to include decisions: $\text{Decision}_a(q_a) \Rightarrow \text{V}(q_a) = \text{V}(m) \mid m \in q_a$

Although decisions are not messages, applications might send decisions in other messages as a kind of "proof of consensus." This is how the Heterogeneous Paxos integrity attestations work in our prototype blockchains (§ 6).

Caught. Some behavior can create proof that an acceptor is Byzantine. Unlike Byzantine Paxos, our acceptors and learners must adapt to Byzantine behavior. We say that an acceptor p is *Caught* in a message x if the transitive references of the messages include evidence such as two messages, m and m', both signed by p, in which neither is featured in the other's transitive references (safe acceptors transitively reference all prior messages).

▶ **Definition 22.** $$Caught(x) \triangleq \left\{ Sig(m) \;\middle|\; \begin{array}{r} \{m, m'\} \subseteq Tran(x) \ \wedge\ Sig(m) = Sig(m') \\ \wedge\quad m \notin Tran(m') \ \wedge\ m' \notin Tran(m) \end{array} \right\}$$

Connected. When some acceptors are proved Byzantine, clearly some learners need not agree, meaning that $\mathcal{S}$ isn't in the edge between them in the CLG: at least one acceptor in each safe set in the edge is proven Byzantine. Homogeneous learners are always connected unless there are so many failures no consensus is required.

▶ **Definition 23.** $Con_a(x) \triangleq \{\ b \mid s \in a\text{–}b \in CLG \ \wedge\ s \cap Caught(x) = \emptyset\ \}$

It is clear that disconnected learners may not agree, and so each $2a$ message x will have some implications only for learners still connected to its specified learner: $\text{Con}_{x.lrn}(x)$.

Quorums in Messages. $2a$ messages reference *quorums of messages* with the same value and ballot. A $2a$'s quorums are formed from fresh $1b$ messages with the same ballot and value (we define *fresh* in Definition 28).

▶ **Definition 24.** $$q(x : 2a) \triangleq \left\{ m \;\middle|\; m : 1b \ \wedge\ fresh_{x.lrn}(m) \ \wedge\ m \in Tran(x) \ \wedge\ b(m) = b(x) \right\}$$

Buried messages. A $2a$ message can become irrelevant if, after a time, an entire quorum of acceptors has seen $2a$s with different values, the same learner, and higher ballot numbers. We call such a $2a$ *buried* (in the context of some later message y):

▶ **Definition 25.**

$$Buried(x : 2a, y) \triangleq \left\{ Sig(m) \;\middle|\; \begin{array}{l} m \in Tran(y) \ \wedge\ z : 2a \ \wedge\ \{x, z\} \subseteq Tran(m) \\ \wedge\ V(z) \neq V(x) \ \wedge\ b(z) > b(x) \ \wedge\ z.lrn = x.lrn \end{array} \right\} \in Q_{x.lrn}$$

Well-Formedness. In addition to the basic message layout, *2a* and *1b* messages must be *well-formed*. No *2a* should have an invalid quorum upon creation, and no acceptor should create a *2a* unless it sent one of the *1b* messages in the *2a*. Similarly, no *1b* should reference any message with the same ballot number besides a *1a* (safe acceptors make *1b*s as soon as they receive a *1a*). Acceptors and learners should ignore messages that are not well-formed.

▶ **Assumption 26** (Well-Formedness Assumption)**.**

$$x : 1b \ \wedge \ y \in Tran(x) \ \wedge \ x \neq y \ \wedge \ y \neq Get1a(x) \ \Rightarrow \ b(y) \neq b(x)$$
$$z : 2a \Rightarrow \ q(z) \in Q_{z.lrn} \ \wedge \ Sig(z) \in Sig(q(z))$$

Connected *2a* messages. Entangled learners must agree, but learners that are not connected are not entangled, so they need not agree. Intuitively, a *1b* message references a *2a* message to demonstrate that some learner may have decided some value. For learner a, it can be useful to find the set of *2a* messages from the same sender as a message x (and sent earlier) which are still unburied, and for learners connected to a. The *1b* cannot be used to make any new *2a* messages for learner a that have values different from these *2a* messages.

▶ **Definition 27.** $$Con2as_a(x) \triangleq \left\{ m \;\middle|\; \begin{array}{l} m : 2a \ \wedge \ m \in Tran(x) \ \wedge \ Sig(m) = Sig(x) \\ \wedge \ \neg Buried(m, x) \ \wedge \ m.lrn \in Con_a(x) \end{array} \right\}$$

Fresh *1b* messages. Acceptors send a *1b* message whenever they receive a *1a* message with a ballot number higher than they have yet seen. However, this does not mean that the *1b*'s value (which is the same as the *1a*'s) agrees with that of *2a* messages the acceptor has already sent. We call a *1b* message *fresh* (with respect to a learner) when its value agrees with that of unburied *2a* messages the acceptor has sent.

▶ **Definition 28.** $$fresh_a(x : 1b) \triangleq \forall m \in Con2as_a(x). \ V(x) = V(m)$$

5.4 Ballots

Heterogeneous Paxos can be thought of as taking place in *stages* identified by natural numbers called ballots. § 5.6.3 of [47] describes one way to construct unique ballot numbers.

Multiple Ballots. Proposers construct new 1a messages (with a value and a unique ballot number), and send them to all acceptors. Just like in Homogeneous Byzantine Consensus, it is possible for a ballot to *fail*: after some number of ballots, it may be the case that all messages have arrived, the protocol in Figure 3 doesn't require any acceptor to send any further messages, and yet no learner has decided. For this reason, it is necessary to start a new ballot when an old one is failing.

One way to handle this is to leave the responsibility at the proposers: if a proposer proposes a ballot, and learners don't decide for a while, then the proposer should propose again. Randomized exponential backoff can be used to allow clients to adapt to the unknown delay in a partially synchronous [18] network without flooding the system.

Another way is to have acceptors propose after a ballot has failed: when sufficiently many *1b* messages for a given ballot are collected, but none are fresh, an acceptor could send a new *1a*. There are subtleties to ensuring liveness, which we discuss in § 6.4.1 of [47].

5.5 Safety

Under our assumptions (§ 5.2 of [47]), Heterogeneous Paxos has the safety properties of Validity and Agreement (proofs in § 6.2 of [47] and § 6.3 of [47]):

▶ **Theorem 29** (Validity). *Heterogeneous Paxos is Valid (Definition 7):* $Decision_a(q_a) \Rightarrow \exists x : 1a.\, V(x) = V(q_a)$

▶ **Theorem 30** (Agreement). *Heterogeneous Paxos has Agreement (Definition 9):* $Entangled(a, b) \wedge Decision_a(q_a) \wedge Decision_b(q_b) \Rightarrow V(q_a) = V(q_b)$

5.6 Liveness

Heterogeneous Paxos, and indeed Byzantine Paxos, rely on a weak network assumption to guarantee termination. The assumption is complex precisely because it is weak; a simpler but stronger assumption, such as a partially synchronous network, would suffice.

▶ **Assumption 31** (Network Assumption). *To guarantee that a learner a decides, we assume that for some quorum $q_a \in Q_a$ of safe and live acceptors:*

- *Eventually, there will be 13 consecutive periods of any duration, with no time in between, numbered 0 through 12, such that any message sent to a or an acceptor in q_a before one period begins is delivered before it ends.*
- *If an acceptor in q_a sends a message in between receiving two messages m and m' (and it receives no other messages in between), and m is delivered in some period n, then the message is sent in period n.*
- *No 1a message except x, y, and z is delivered to any acceptor in q_a during any period.*
- *x is delivered to an acceptor in q_a in period 0, y is delivered to an acceptor in q_a in period 4, and z is delivered to an acceptor in q_a in period 9.*
- *$V(y) = V(z)$ is the value of the highest ballot 2a known to any acceptor in q_a at the end of period 3.*
- *$b(x)$ is greater than any ballot number of any message delivered to any acceptor in q_a before period 0, and $b(x) < b(y) < b(z)$.*

This assumption is *only necessary* for termination, not any safety property. We prove our termination theorem in § 6.4.1 of [47].

▶ **Theorem 32** (Termination). *If Assumption 31 holds for learner a, then a has Termination (Definition 11). Specifically, after period 12:* $Terminating(a) \Rightarrow \exists q_a.Decision_a(q_a)$ *If Assumption 31 holds for all terminating learners, then Heterogeneous Paxos has Termination.*

A *partially synchronous* network is one in which, after some point in time, there exists some (possibly unknown) constant latency Δ such that all sent messages arrive within Δ [18]. We explain elsewhere how to add artificial message receipt delays to Heterogeneous Paxos in order to guarantee Assumption 31 in a partially synchronous network (§ 6.4.2 of [47]).

6 Implementation

Since Heterogeneous Paxos is designed for cross-domain applications where different parties have different trust assumptions, it is well-suited for blockchains. We constructed a variety of example blockchains using the Charlotte framework [46], which allows for pluggable integrity (consensus) mechanisms. Our servers are implemented in 1,704 lines of open-source Java. Charlotte uses 256-bit SHA3 hashes, P256 elliptic curve signatures, protobufs [43] for marshaling, and gRPC [24] for transmitting messages over TLS 1.3 channels.

To explore the performance of Heterogeneous Paxos, we created several blockchains with different CLGs (§ 3). The results (§ 9.3 of [47]) show that heterogeneous configurations save resources and latency compared with homogeneous configurations tolerating the same failures. For instance, in our example configuration § 1.1, a Homogeneous configuration tolerating similar failures would cost an extra 7 unnecessary acceptors, increasing latency overhead by 51% relative to Heterogeneous Paxos. *2a* messages include a quorum of 256-bit message hashes, so they expand linearly with quorum size, as does the cost of unmarshaling and verifying the signatures of the messages referenced. In all experiments, however, computational overhead was dominated by the theoretical minimum (simulated) geodistributed network latency.

7 Related Work

Heterogeneous Acceptors and Failures. Heterogeneous Paxos is based on Leslie Lamport's Byzantine-fault-tolerant variant [30] of Paxos [28]. Byzantine Paxos supports heterogeneous acceptors because it uses quorums: not all acceptors need be of equal worth, but all quorums are. Although Lamport does not describe it explicitly, Byzantine Paxos can have heterogeneous, or *mixed* [48], failures, so long as quorum intersections have a safe acceptor and at least one quorum is safe and live.

Many papers have investigated hybrid failure models [48, 13, 7, 33] in which different consensus protocol acceptors can have different failure modes, including crash failures and Byzantine failures (heterogeneous failures). These papers typically investigate how many failures in each class can be tolerated. Other papers have looked at system models in which different acceptors may be more or less likely to fail [22, 38], or where failures are dependent (heterogeneous acceptors) [27, 17, 25].

Further generalizations are possible. Our Learner Graph uses only *safe* and *live* acceptors, but its labels might be generalized to support other failure types such as rational failures [3]. We have only considered learners that all make the same (weak) synchrony assumption, but others have studied learners with heterogeneous network assumptions [5, 37].

Heterogeneous Learners. Unlike ours, most related work conflates learners and acceptors. Early related work on "Consensus with Unknown Participants" [11, 23, 4] defines protocols in which each participant knows only a subset of other participants, inducing a "who-knows-whom" digraph; this work identifies properties of this graph that must hold to achieve consensus. Not every participant knows all participants, but trust assumptions are homogeneous: participants have the same beliefs about trustworthiness of other participants.

Our prior work describes [45] a heterogeneous failure model in which different participants may have different failure assumptions about other participants. We distinguished learners whose failure assumptions are accurate from those whose failure assumptions are inaccurate and we specified a heterogeneous consensus protocol in terms of the possibly different conditions under which each learner is guaranteed agreement. The paper constructs a heterogeneous consensus protocol that meets the requirements of all learners using lattice-based information flow to analyze and prove protocol properties.

Heterogeneous learners became of interest to blockchain implementations based on voting protocols where open membership was desirable. Ripple (XRP) [44] was the earliest blockchain to attempt support for heterogeneous learners. Originally, each learner had its own Unique Node List (UNL), the set of acceptors that it partially trusts and uses for making decisions. An acceptor in more UNLs is implicitly more influential. The protocol was updated because of correctness issues [12], and support for diverse UNLs was all but eliminated. Ripple has

proposed a protocol called Cobalt [36], in which each learner specifies a set of acceptors they partially trust, and it works if those sets intersect "enough." Cobalt does not account for heterogeneous failures, and only limited acceptor heterogeneity.

The Stellar Consensus [39, 34, 35] blockchain protocol supports both heterogeneous learners and acceptors, although it does not distinguish the two; each learner specifies a set of "quorum slices." Like Cobalt, Stellar does not account for heterogeneous failures. Neither Stellar nor Cobalt match Heterogeneous Paxos' best-case latency. Heterogeneous Paxos inherits Byzantine Paxos' 3-message-send best case latency, which is optimal for a consensus tolerating $\lceil \frac{n}{3} \rceil - 1$ failures in the homogeneous Byzantine case or $\lceil \frac{n}{2} \rceil - 1$ failures in the homogeneous crash case [6]. However, both Cobalt and Stellar are designed for an "open-world" model, where not all acceptors and learners are known in advance. We have not yet adapted Heterogeneous Paxos to an open-world setting.

The heterogeneous learner models of Cobalt and Stellar have been studied in detail by García-Pérez and Gotsman [21]. Cachin and Tackmann examine Stellar-style asymmetric trust models, including in shared-memory environments [8]. However, neither paper separates learners from acceptors, attempts to solve consensus, or considers heterogeneous failures; the Learner Graph is more general.

Like our work, *Flexible BFT* [37] distinguishes learners from acceptors and accounts for both heterogeneous learners and heterogeneous failures. It does not allow heterogeneous acceptors: they are interchangeable, and quorums are specified by size. Flexible BFT also has optimal best-case latency. It does not support crash failures, but introduces a new failure type called *alive-but-corrupt* for acceptors interested in violating safety but not liveness.

8 Conclusion

Heterogeneous Paxos is the first consensus protocol with heterogeneous acceptors, failures, and learners. It is based on the Learner Graph, a new and expressive way to capture learners' diverse failure-tolerance assumptions. Heterogeneous consensus facilitates a more nuanced approach that can save time and resources, or even make previously unachievable consensus possible. Heterogeneous Paxos is proven correct against our new generalization of consensus for heterogeneous settings. This approach is well-suited to systems spanning heterogeneous trust domains; for example, we demonstrate working blockchains with heterogeneous trust.

Future work may expand learner graphs to represent even more types of failures. Heterogeneous Paxos may be extended to allow for changing configurations, or improved efficiency in terms of bandwidth and computational overhead. New protocols can also make use of our definition of heterogeneous consensus, perhaps adding new guarantees such as probabilistic termination in asynchronous networks.

References

1 An introduction to Hyperledger, 2018.
2 Quorum whitepaper, 2018.
3 A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J. Martin, and C. Porth. BAR fault tolerance for cooperative services. In *SOSP*, pages 45–58, 2005.
4 E. A. Alchieri, A. N. Bessani, J. Silva Fraga, and F. Greve. Byzantine consensus with unknown participants. In *OPODIS*, pages 22–40, 2008.
5 E. Blum, J. Katz, and J. Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In *Theory of Cryptography*, pages 131–150, 2019.
6 G. Bracha and S. Toueg. Resilient consensus protocols. In *PODC*, pages 12–26, 1983.

7 C. Cachin and M. Backes. Reliable broadcast in a computational hybrid model with byzantine faults, crashes, and recoveries. In *DSN*, 2003.
8 C. Cachin and B. Tackmann. Asymmetric distributed trust. In *OPODIS*, 2019.
9 B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, and H. Simitci et al. Windows Azure Storage: a highly available cloud storage service with strong consistency. In *SOSP*, 2011.
10 M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *OSDI*, 1999.
11 D. Cavin, Y. Sasson, and A. Schiper. Consensus with unknown participants or fundamental self-organization. In *ADHOC-NOW*, 2004.
12 B. Chase and E. MacBrough. Analysis of the XRP ledger consensus protocol. *CoRR*, abs/1802.07242, 2018.
13 A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche. Upright cluster services. In *SOSP*, pages 277–290, 2009.
14 J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, and P. Hochschild et al. Spanner: Google's globally distributed database. *TOCS*, 31(3):8, 2013.
15 M. Correia, N. Neves, and P. Veríssimo. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. *Comput. J.*, 49:82–96, January 2006.
16 K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer. On scaling decentralized blockchains. In *Financial Cryptography and Data Security*, 2016.
17 C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and A. Tielmann. The disagreement power of an adversary. *Distributed Computing*, 24:137–147, November 2011.
18 C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.
19 M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
20 Ethereum Foundation. Ethereum white paper. Technical report, Ethereum Foundation, 2018.
21 Á. García-Pérez and A. Gotsman. Federated byzantine quorum systems. In *OPODIS*, pages 17:1–17:16, 2018.
22 D. K. Gifford. Weighted voting for replicated data. In *SOSP*, 1979.
23 F. Greve and S. Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *DSN*, pages 82–91, 2007.
24 grpc: A high performance, open-source universal RPC framework. `https://grpc.io`, 2018.
25 R. Guerraoui and M. Vukolić. Refined quorum systems. In *PODC*, 2007.
26 M. Hearn and R. G. Brown. Corda: A distributed ledger. Technical report, r3, 2019.
27 F. Junqueira and K. Marzullo. Designing algorithms for dependent process failures. In *Workshop on Future Directions in Distributed Computing*, pages 24–28, 2003.
28 L. Lamport. The Part-time Parliament. *TOCS*, 16(2):133–169, May 1998.
29 L. Lamport. Paxos made simple. Technical report, Microsoft Research, December 2001.
30 L. Lamport. Byzantizing Paxos by refinement. In *DISC*, pages 211–224, 2011.
31 L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. on Programming Languages and Systems*, 4(3):382–401, July 1982.
32 B. W. Lampson and H. E. Sturgis. Crash recovery in a distributed data storage system. Technical report, Xerox Palo Alto Research Center, Palo Alto, CA, 1979.
33 S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolic. XFT: Practical fault tolerance beyond crashes. In *OSDI*, 2016.
34 M. Lokhava, G. Losa, D. Mazières, G. Hoare, N. P. E. Barry, E. Gafni, J. Jové, R. Malinowsky, and J. M. McCaleb. Fast and secure global payments with Stellar. In *SOSP*, 2019.
35 G. Losa, E. Gafni, and D. Mazières. Stellar consensus by instantiation. In *DISC*, 2019.
36 E. MacBrough. Cobalt: BFT governance in open networks. *CoRR*, abs/1802.07240, 2018.
37 D. Malkhi, K. Nayak, and L. Ren. Flexible byzantine fault tolerance. In *CCS*, 2019.

38 D. Malkhi and M. Reiter. Byzantine quorum systems. In *STOC*, 1997.
39 D. Mazières. The Stellar consensus protocol: A federated model for internet-level consensus. `https://www.stellar.org`, April 2015.
40 A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The Honey Badger of BFT protocols. In *CCS*, pages 31–42, 2016.
41 S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
42 B. Preneel. Collision resistance. In *Encyclopedia of Cryptography and Security*, 2011.
43 Protocol buffers. `https://developers.google.com/protocol-buffers/`, 2018.
44 D. Schwartz, N. Youngs, and A. Britto. The Ripple protocol consensus algorithm. Technical report, Ripple Labs Inc, 2014.
45 I. Sheff, R. van Renesse, and A. C. Myers. Distributed protocols and heterogeneous trust. *CoRR*, abs/1412.3136(arXiv:1412.3136), December 2014.
46 I. Sheff, X. Wang, H. Ni, R. van Renesse, and A. C. Myers. Charlotte: Composable authenticated distributed data structures, technical report, 2019.
47 I. Sheff, X. Wang, R. van Renesse, and A. C. Myers. Heterogeneous Paxos: Technical report, 2020.
48 H. Siu, Y. Chin, and W. Yang. Byzantine agreement in the presence of mixed faults on processors and links. *Parallel and Distributed Systems*, 9(4), April 1998.