Distributed Verifiable Ledgers: A Framework

Preamble

When discussing distributed ledgers, the spectrum of technology is often thought of as being along a permissionless <-> permissioned axis (some introduce the idea of doubly permissioned/permissionless). I have always found this terminology confusing - I hope this way of looking at it is less confusing.

This framework says nothing about how the desired properties are delivered technologically, but hopefully thinking about your problem in these terms will make it easier to make a technology choice. With that in mind, I look at how some existing technologies fit into the framework.

Overview

In this framework, there are four axes on which to frame a problem, namely admission control, consensus, verification and enforcement.

The first is about what can be accepted into the ledger, the second about who the keepers of the ledger are, and how they work together, the third about how the ledger is cryptographically verified, and the fourth about ensuring correct behaviour within the system (i.e. enforcing the “verifiable” part of the distributed ledgers).

Note that these four components are not *entirely* independent - for example, to create a fully verifiable ledger, there must be enough information available about admission control and consensus to allow enforcement to take place.

Admission Control

Clearly there must be some framing of what is permitted in the ledger, or it can be rendered unusable by filling it with rubbish.

Bitcoin, for example, only admits “blocks” which have an appropriate hash (i.e. which chains from some previous block and which has a hash of appropriate work factor). The blocks themselves have a finite size and are accepted at a (more-or-less) constant rate, so this sets an upper rate limit to what can be included in the chain, regardless of content.

Certificate Transparency only admits TLS certificates with a well-formed signature chain from a known CA. CAs only issue certificates for legitimate websites, and so there is an upper bound to the amount of content in a CT log.

Some designs, for example CT, require that entries appear in multiple ledgers, and I include this requirement under the broad heading of “admission control”.

So, admission control consists of the rules under which entries are admitted to logs, and can be anything verifiable, but will typically include one or more of:

- Correct formatting
- Correct provenance
- Redundancy policy

Admission control is one of the dimensions of "permissioning" in traditional distributed ledger discourse.

Consensus

Once entries appear in individual ledgers, there is still the need to decide on the consensus view. In the case of CT, for example, the consensus is the union of all trusted logs. Currently, that means "trusted by Chrome" but will likely evolve as CT is used in more browsers and other contexts.

In Bitcoin, consensus is the longest chain[1].

Consensus can potentially be in the eye of the beholder. For example, imagine a future CT where different browsers have different trusted logs. A user will only care what the browser currently in use trusts, not what other browsers trust (this is because in CT, enforcement is done by browsers, and hence the browser view is the authoritative one).

There are many mechanisms to allow consensus, for example

- Proof of work in an unknown group of ledgers (i.e. "most work wins")
- Majority rule in a known group of ledgers
- The union of all ledgers

Consensus rules will likely imply some kind of admission control for ledgers - for example, in CT, ledgers are admitted by the browsers. In Bitcoin, they are admitted by demonstrating work.

Verification

None of this is useful unless there is some mechanism by which correct behaviour can be verified. Exactly how this is done is highly dependent on the content (i.e. admission control) and consensus mechanism, but in general, the ideal is that any statement about correct behaviour can be verified by examining or interacting with the ledgers themselves. Rather than try to nail down all possible avenues for verification, I give some examples below.

[1] Note that this means you never really know what is in the consensus, but I have already written about that elsewhere, and also, see the section on enforcement.

In CT, the ledger's correct behaviour can largely be verified by examining the ledgers themselves - for example, the append-only and non-equivocation properties can be verified from the ledgers themselves[2]. If they are violated, then cryptographic evidence can be provided.

There are, however, a couple of properties of CT that require a different approach. For example, ledgers are required to incorporate entries within a certain time. This is established in a statistical way by the authority (the browsers in this case) submitting entries and verifying they are added to the ledger quickly enough.

Likewise, if it is claimed a ledger is refusing to accept a valid entry, then the authority can verify that by attempting to submit it itself.

In Bitcoin, the ledger itself provides its own audit trail which can be used to verify cryptographic correctness, but (as far as I know) there is no mechanism to verify that, for example, that ledgers don't refuse entries from some particular party, or containing some particular information.

Enforcement

Verification is pointless without some mechanism for enforcement.

In CT, rules are enforced in various ways - for example, ledgers failing to meet their requirements results in those ledgers becoming untrusted. CAs behaving incorrectly are detected by CT (for at least some types of misbehaviour) but the enforcement comes from outside the CT system, generally in the form of action by the CA/Browser forum, or the individual browsers.

Bitcoin enforces some aspects of correctness in the Bitcoin software itself - for example, correct formatting and chaining of entries. It also protects against the perpetual uncertainty of what the "longest chain" actually is by adding checkpoints to the ledger to the software itself[3].

In general, enforcement of the rules, both those of the ledger and those of the ecosystem it is part of, has to somehow be built into the system, but exactly how is highly dependent on the system itself. It is interesting to note that some aspects of enforcement for CT, in particular analysis of the logs and of the behaviour of CAs revealed by them, emerged spontaneously after CT was deployed.

[2] Note that non-equivocation verification requires some form of gossip. I assume here the as-yet-unimplemented gossipless gossip proposal is in use (in which each ledger adds copies of its own tree heads to the other ledgers, thus requiring all ledgers to equivocate to allow any to do so).
[3] The software does not allow a new "longer chain" that originates from a block before the most recent checkpoint.