tldr; use social consensus & end up migrating everything!

Summary

This proposal suggests a simple approach to Aztec's governance and upgrade mechanism - focusing on making everything non-upgradable, and therefore dubbed "non-governance" or "anarchy" (perhaps we can come up with a better name, or just call it social consensus?).

Instead of having a system of governance that could potentially be co-opted by a few key stakeholders, such as the largest token holders or infrastructure providers, we propose a system where changes are triggered solely by community activity and acceptance, ala social consensus

. The idea is somewhat to avoid the traditional concept of a traditional software upgrade altogether and let the network evolve organically with the community's direct participation and manually opting-in to all changes, while preserving the immutability and accessibility of all previous versions.

Comparisons

Unlike most networks that utilize a governance mechanism involving voting rights, staking, or delegated power, non-govnernance eliminates the traditional hierarchy and goes back to what we saw with Bitcoin and Ethereum via social consensus.

In contrast to the Republic, or other types of "balanced governance" which attempt to achieve trade-offs between user experience and immutability, there is no multi-sig, no token voting, actually nothing really beyond code and social consensus.

[

3368×753 92.2 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/f7c00452dcb377289bf0e7045a300e996ac7258b.jpeg)

Credit to @LasseAztec for the diagram

This proposal has some explicit shortcomings, and does not meet the requirement of user-friendly migration whatsoever. A majority of users will need to manually migrate their assets/accounts to support an upgrade. This specific consideration is especially unique to Layer-2 networks in a sense, as social consensus on Ethereum or other Layer-1 networks do not have similar notion of (or perhaps importance placed on) "bridged assets" or assets that were initially created on other ledgers to worry about (for further reading, consider Jon's blog "Rollups aren't real").

As part of the user experience tradeoff, this proposal does nicely give users, developers, and infrastructure providers, the strongest possible guarantees that the network, applications, and services they're using today will not change tomorrow and "rug" via it's own governance. I believe that the added decentralization, credible neutrality, and inability to co-opt make this trade-off a worth-while consideration - and perhaps more importantly, helps paint what the ungovernable end of the spectrum looks like.

[

3483×4676 847 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/7800070df64f13c8790f14bd4762db1f6fd8d0fc.jpeg)

Credit to @LasseAztec for the diagram

Details

Non-governance operates under a very simple idea: if an upgrade becomes necessary, a new, separate version of the protocol is created, while keeping the original intact as long as someone, perhaps the user themselves, is willing to run infrastructure for the previous versions. This means that users, developers, and infrastructure providers then must voluntarily, and most often manually, migrate to the new protocol if they wish, based on the perceived benefits or improvements. The new protocol version would state with no state or from a genesis block, and be dependent on community migrations to achieve adoption or usage. This also means that they can also choose to stay with the old protocol, as long as there are a sufficient number of service providers continuing to support those pervious versions. It is entirely up to social consensus to determine which deployments are canonical or relevant to the community.

User stories

Here’s how it would address &/or impact the user personas:

Users:

Users are free to support or reject a network upgrade. To support an upgrade, they need to migrate their assets to the new version of the network. If they choose to reject the upgrade, they can simply continue using the original version.

The user experience of active migrations is surely one of the largest downsides or tradeoffs of this approach (solutions & suggestions welcome). On one hand there are potential concerns about legacy software not being supported, requiring users to potentially do expensive migrations often enough to be annoying… however, sufficiently sophisticated and financially capable users should retain the option to run any necessary infrastructure perpetually.

Developers:

Developers have the freedom to support the older version or the new one. To adapt to the new version, they will need to adjust their application interfaces and infrastructure. Additionally there will likely need to be some easy / potentially standardized way to migrate state. To reject new versions, they do nothing. There are a variety of examples of project’s doing this well by distributing their frontends over IPFS, or otherwise, to ensure they will always be accessible.

Infrastructure Providers:

Infrastructure providers have the freedom to support the older version or the new one. To adapt to the new version, they will need to update their infrastructure. To reject new versions, they do nothing. There may be some financial motiviaton and community demand to support a variety of older versions or widely adopted forks, which could become burdensome, or perhaps profitable.

Requirements

Censorship Resistance:

Since the proposal suggests social consensus, it is extremely resistant to upgrade specific censorship. If a user is being censorsed with respect to an upgrade, or an outdated version of the software, they would typically fallback to running their own node(s). At that point, concerns would primarily be within service providers (e.g. cloud environments) and code distribution platforms (e.g. github) required to run their own node(s).

Assuming that users have access to the code, knowledge of how to use it, and a sufficiently capable machine to run the software on, there are no other points of concern unique to upgrade mechanisms.

Grieving attacks or Malicious Upgrades:

This approach virtually eliminates the risk of malicious upgrades, as there is no centralized decision-making process that could be exploited or taken control of. If a version is deployed without buy-in from the community, it is unlikely to be accepted as canonical, and users would be unlikely to migrate their software or assets to it.

Well-Known Mechanism:

The proposal is extremely transparent, as users have direct control and can decide independently which version of the network to support. It is likely to result in a longer deliberation process, and periods in which a new version has been deployed but has not yet been accepted as canonical by social consensus - leading to some potential confusion.

While the mechanism itself is well known it may not easy to understand what the latest version is, or even reason about all the possible versions that could potentially exist - this complexity could be real and considerable if a large number of forks or previous versions become widely utilized.

Public decision making:

By using social consensus to drive the direction of the network, we can ensure that decisions are not dominated by a small group, or really any group. You, and everyone else, must convince people it’s worth it to migrate.

Questions:

1. What ways could this proposal be improved?
2. There are a number of small tweaks or expansions to this idea of social consensus ruling everything, such as introducing an informal version registry of sorts (perhaps multiple) that could be valuable in addressing the user experience shortfalls unqiue to L2’s that aren’t relevant to Bitcoin or Ethereum (being the most well understood uses of social consensus). There are also secuity considerations that are somewhat explored in question #3

but are worth more direct exporation.

1. There are a number of small tweaks or expansions to this idea of social consensus ruling everything, such as

introducing an informal version registry of sorts (perhaps multiple) that could be valuable in addressing the user experience shortfalls unqiue to L2's that aren't relevant to Bitcoin or Ethereum (being the most well understood uses of social consensus). There are also secuity considerations that are somewhat explored in question #3

but are worth more direct exporation.

1. How do we handle a scenario where the community is split across multiple versions?
2. Probably nothing, it's designed for these types of scenarios to exist/thrive independent to other versions or instances of the network (we all remember ethereum versus ethereum classic…right? …right??)

3. Probably nothing, it's designed for these types of scenarios to exist/thrive independent to other versions or instances of the network (we all remember ethereum versus ethereum classic…right? …right??)

4. What happens in the event of a vulnerability?
5. In general, client &/or contract implementators would work on a potential fix to the issue while users would attempt to recover and force exit their funds. There would be no way to "freeze" or stop the protocol/network from operating, in this specific proposal, and so another way of thinking about this candidly is that it would be a race to withdraw funds between users and attackers. This is potentially not be an acceptable security assumption for the network, particularly in the early days. For the sake of the proposal process and comparisons, I will leave others to explore a design space that consider's security councils, multisig's that could potentially pause/freeze/patch the network, or otherwise.
6. It is worth exploring the "training wheels" roadmap outlined by Vitalik, where there are multiple implementations of the state transitioner, and potentially a "fallback security council" that can step in to adjudicate issues between the two (or more) implementations, or perhaps if there is not a rollup produced within X timeframe (e.g. 7 days).

7. It is worth exploring the "training wheels" roadmap outlined by Vitalik, where there are multiple implementations of the state transitioner, and potentially a "fallback security council" that can step in to adjudicate issues between the two (or more) implementations, or perhaps if there is not a rollup produced within X timeframe (e.g. 7 days).

8. In general, client &/or contract implementators would work on a potential fix to the issue while users would attempt to recover and force exit their funds. There would be no way to "freeze" or stop the protocol/network from operating, in this specific proposal, and so another way of thinking about this candidly is that it would be a race to withdraw funds between users and attackers. This is potentially not be an acceptable security assumption for the network, particularly in the early days. For the sake of the proposal process and comparisons, I will leave others to explore a design space that consider's security councils, multisig's that could potentially pause/freeze/patch the network, or otherwise.

9. It is worth exploring the "training wheels" roadmap outlined by Vitalik, where there are multiple implementations of the state transitioner, and potentially a "fallback security council" that can step in to adjudicate issues between the two (or more) implementations, or perhaps if there is not a rollup produced within X timeframe (e.g. 7 days).

10. It is worth exploring the "training wheels" roadmap outlined by Vitalik, where there are multiple implementations of the state transitioner, and potentially a "fallback security council" that can step in to adjudicate issues between the two (or more) implementations, or perhaps if there is not a rollup produced within X timeframe (e.g. 7 days).

11. How does a hypothetical network token get minted across potentially different versions of the L1 rollup contract, if the rollup contract itself is non-upgradeable and requires migrations?
12. This is a tough question which I don't have a great answer for. Perhaps there are different tokens, each themselves versioned, and those with sufficient social consensus can be swapped 1:1 with previous versions. In general, it'd seems to result in a migration and some marketplaces/ecosystems needing to support previous versioned tokens. I'm not sure, and without an elegant solution this is considerably a large downside for this proposal. Suggestions welcome!

13. This is a tough question which I don't have a great answer for. Perhaps there are different tokens, each themselves versioned, and those with sufficient social consensus can be swapped 1:1 with previous versions. In general, it'd seems to result in a migration and some marketplaces/ecosystems needing to support previous versioned tokens. I'm not sure, and without an elegant solution this is considerably a large downside for this proposal. Suggestions welcome!