University of Edinburgh and IOHK and National and Kapodistrian University of Athens and IOHK

Towards a general framework for software updates in decentralized ledger systems

Anonymous Author(s)

Research partially supported by H2020 project PRIVILEDGE # 780477

Towards a general framework for software updates in decentralized ledger systems

Anonymous Author(s)

ABSTRACT

Software updates are a synonym to software evolution and thus are ubiquitous and inevitable to any blockchain platform. In this paper, we propose a general framework for software updates in distributed ledger systems. Our framework is primarily focused on permission-less blockchains and aims at providing a solid set of enhancements, covering the full spectrum of a blockchain system, in order to ensure a secure and decentralised update mechanism for a public ledger. Our main contribution is the proposal of a logical architecture that enables a smooth incorporation of updates into all components of a blockchain system. Moreover, we drill down to each component of the architecture and propose specific enhancements and changes. In addition, we describe in detail new components such as the governance module and the update-logic module. For the former, we propose a voting protocol, in order to enable a decentralized update consensus and for the latter, we show how we can implement different update policies for each type of update based on a rich set of update metadata. Finally, we implement our ideas in a prototype update system for the Cardano blockchain and discuss implementation issues and validation results.

CCS CONCEPTS

• **Security and privacy** → Use https://dl.acm.org/ccs.cfm to generate actual concepts section for your paper;

KEYWORDS

updates; sidechains; governance

1 INTRODUCTION

Software updates are everyware. The most vital aspect for the sustainability of any software system is its ability to effectively and swiftly adapt to changes (i.e., software updates). Therefore the adoption of changes is in the heart of the lifecycle of any system and blockchain systems are no exception. Software updates might be triggered by a plethora of different reasons.

Typically, the main driver for a change will be a change request, or a new feature request by the user community. These type of changes will be planned for inclusion in some future release and then implemented and properly tested prior to deployment into production. In addition, another major source of changes is the correction of bugs and errors of the system. The latter usually produce the so-called "hot-fixes", which are handled in a totally different manner than the former. Depending on the severity level of the problem, there are different levels of reaction and methods for the deployment of the correction patch into production.

More specifically for blockchain systems, a typical source of changes are enhancements at the consensus protocol level and/or at the consensus rules level. The former usually aim at reinforcing the protocol against a broader scope of adversary attacks, while the latter usually are triggered through the enhancement of the information and the semantics of this information embedded within each transaction and/or a block.

Finally, there are also more radical changes, which are usually caused by the introduction of new research ideas that try to solve significant problems (e.g., scalability issues, interoperability issues, etc.) and the advent of new technology, which becomes relevant. These type of changes usually introduce new concepts and are not just enhancements and thus trigger a major change to the system.

Context of this paper. In this paper our focus is on the update mechanism of permissionless stake-based blockchain systems. We try to overcome known shortcomings of today's update methods for blockchain systems and propose a logical architecture for an update mechanism for stake-based ledgers. Our architecture covers all the components of a typical blockchain system and demonstrates how can such a mechanism be smoothly incorporated into each layer of the blockchain system. We describe in detail the various enhancements and changes in the existing components and then proceed into describing new components such as the governernance component, or the update-logic component. Our aim is to cover all aspects of software updating, from update proposal submission to update deployment and activation. Finally, we implement our logical architecture into a prototype update system for the Cardano blockchain. We then discuss various implementation issues and the prototype architecture.

Problem Definition The traditional way of handling software updates is neither decentralized nor secure

- No standard way to propose updates
- Not a decentralized and democratic way to reach at a consensus on update priorities. Only "social consensus" is reached via social media. This is unstable and prone to chain splits.
- No essential auditing and verifiability of the agreement
- No standard way to record the immutable history of all these events (proposals,agreement,
- activation of updates)
- No standard method for security guarantees for the software installed
- No standard way to resolve conflicts and respect dependencies
- No standard Update metadata
- One-size-fits-all for all "types" of changes (bugs,CRs)

- Hard-forks are the norm

Limitations of Existing Solutions Good examples of governance:

- Treasury paper (https://eprint.iacr.org/2018/435)
- Dash Governance - https://www.dash.org/governance/
- Decred Governance - https://docs.decred.org/governance/consensus-rules-voting/

Bitcoin and Ethereum use a "Social Governance" scheme, which basically means that decisions on update proposals is reached through discussions on social media. This type of informal guidance is too unstable and prone to chain splits, or prone to becoming too de-facto centralized [https://vitalik.ca/general/2017/12/17/voting.html]

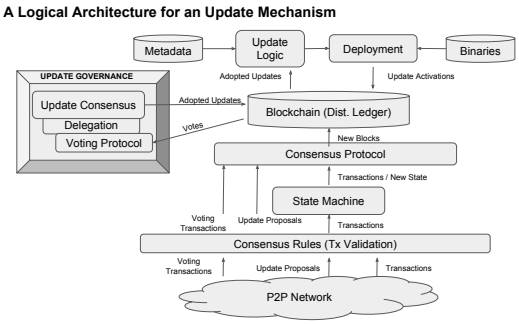
Goal of the paper

Outline of the paper

2 A LOGICAL ARCHITECTURE FOR AN UPDATE MECHANISM

The proposed architecture must define how the various new events (Update Proposal, Votes, Delegation, Update Activations etc.) are incorporated into the blockchain system and describe the interactions between the components.

Figure 1: A Logical Architecture for an Update Mechanism



3 LEDGER ENHANCEMENTS

Recording the History of Updates

We use the ledger as the âĂIJSingle version of the truthâĂİ for updates. We need to propose how the ledger is enhanced in order to accommodate the new events:

- Update Proposals
- Votes
- Delegation
- Update Activation
- Other?

Is only the SL layer affected? What about the CL layer (smart contracts)?

Update Proposals

- New transaction type.
- Issued Updates.
- Revealed Updates.

Update Proposal Lifecycle

State diagram of the various states in the life of an update proposal. What are the valid states?

4 UPDATE GOVERNANCE

Voting Protocol

- A voting protocol must be proposed that will enable an update consensus on. I.e., a community agreement on the priorities of the Update Proposals.
- The protocol must be secure and provide an incentivization scheme that will guarantee safety for the voting results, in an open and truly decentralized setting.
- Need to define the vote transaction

Delegation Protocol

- A voting delegation protocol must be proposed to ensure voting for proposals even when there exist stakeholders that are not at a position to vote for themselves and need to delegate their voting right to some other user.
- Also we need to describe if there is any connection of this delegation with the mining delegation.
- Also, what happens when a stakeholder is not live during update proposal voting

5 CONSENSUS

Integration with Proof-of-Stake Protocols

- How is the voting protocol integrated with a PoS protocol?
- How is the change of stake distribution affects the update governance?
- Can we integrate also with PoW protocols? Is our protocol âĂIJconsensus agnosticâĂİ?

Integration with the Ouroboros Family of Protocols

6 UPDATE LOGIC

Update Metadata

We need to define the set of update metadata that we need to maintain, in order to implement various *update policies*, achieve conflict resolution, respect update dependencies and in general support the full scope of the Updating Process. These metadata will be included in the Update Proposal (see the Update Manifest below)

Update Logic Application

Conflict Resolution. How do we ensure that multiple concurrent requests for updates are handled simultaneously in a way that

- conflicts are resolved and the adopted updates are consistent,

- so that no contradictory updates are to be deployed at the same time
- Community splits over controversial updates are avoided

Dependencies. How do we impose respect for update dependencies, so that the system reaches a consistent state?

Update Policies per type of updates.

- How do we discriminate between different types of updates (e.g., software vs. protocol, bug-fix vs. change request)
- We need to provide a different deployment path for each Type of Update. For example, critical hot-fixes might need to bypass some of the governance steps.
- We need to incorporate into our update logic the notions of bug Severity and âĂIJRequired Speed of DeploymentâĂİ

Decentralized Storage

Update code and metadata must be stored in a decentralized manner and cannot be stored in the ledger. Storing updates in the cloud in centrally-owned servers undermines openness and decentralization. We need a secure P2P database/file-system solution that is smoothly integrated with the ledger.

Update Security Guarantees

How do we ensure that the downloaded software is secure and the same with the Update Proposal that has been voted?

7 DEPLOYMENT / ACTIVATION

Update Activation

What is an activation event and how do we handle it. When should it be triggered, how is it applied and how do we record it?

How do we ensure that Participants who have been inactive when some update was deployed are prevented from getting locked out of the system?

Rollbacks

How can we smoothly rollback an update, in the case of a problem?

Hard/Soft/Velvet forks and Sidechains

- When is a soft fork acceptable?
- When is a hard fork acceptable?
- When should a velvet fork be used?
- When should the sidechain mechanism be used?
- Basic Research results on sidechains for PoS ledgers
- Sidechains and governance in the presence of multiple updates. Can we have multiple âĂIJupdate sidechainsâĂİ running in parallel?

8 PROTOTYPE IMPLEMENTATION

The Prototype Architecture

Validation of the Prototype

9 CONCLUSION

APPENDIX

A APPENDIX