Development of a blockchain-based browser extension for the management of open-source repositories on GitHub.

Michael Frank
michael.frank@stud.hs-flensburg.de
Hochschule Flensburg
Flensburg, Schleswig-Holstein, Deutschland

Nico Hohm
nico.hohm@stud.hs-flensburg.de
Hochschule Flensburg
Flensburg, Schleswig-Holstein, Deutschland

ABSTRACT

With the development of new technologies such as blockchain, new possibilities are available for the organization of open-source projects. In this paper, we present a blockchain-based chrome browser extension for the governance of open-source projects on the GitHub platform. This browser extension has an underlying protocol that is intended to decentralize and democratize governance and also to provide a financial incentive for participation in open-source projects.

KEYWORDS

Open-source development, Open source, Software development, Development initiative, Browser extension, Chrome extension, GitHub, Blockchain, Ethereum, Smart Contracts

1 INTRODUCTION

1.1 Open-source development on GitHub

Open-source development is a type of software development in which a decentralised and collaborative community develops software publicly and transparently. Open-source development enables the creation of innovative and free software through the collaboration of many people and it also provides free access to the software for everyone [17, 18]. Some examples for such significant projects are the Linux Kernel [19] and the Mozilla Firefox browser [14], which are used by millions of people every day. A particular difficulty in open-source development is the coordination of the many developers who contribute to the development of the projects with their own ideas or improvements. Sometimes a change contains an error, therefore each change has to be tracked, traced, and if necessary reversed [18]. These problems occur not only in open source development, but also in normal software development. As in normal software development, version control is used to solve these problems [18, 20].

In particular, the version control protocol Git is ideal for open-source development, because with Git it is possible to have several distributed remotes that can access and manage the same source code [5, 20]. Projects that are coordinated via Git are called repositories. The developers have a local copy of the repository on their systems and can push their changes to or pull the current status from the main repository. These actions are coordinated via a so-called Git server, which has to be hosted somewhere so that the developers can work with it [5]. This server must be permanently accessible otherwise the mentioned actions will no longer work and the coordination of the developers will be interrupted [20]. Hosting services such as GitHub were created so that such problems can be prevented and not everyone has to set up their own Git server if they want to start an open-source project [6, 20]. Today GitHub is the largest Git hosting provider with over 56 million registered developers and over 100 million repositories. [6] The service is also used by large IT corporations such as Microsoft, Facebook or Google and hosts a large number of the largest and most important open-source repositories [8].

Normally, an open-source project on GitHub starts with a user creating a repository for it. The user who created the repository is the owner of it. This user has full control over the repository and can push changes, decide which changes are accepted (merged) and even delete the repository. In addition, he can add so-called collaborators to the repository, who have read and write privileges in the repository [11]. Normal users who do not have collaboration rights can contribute to the open-source project by creating a pull request with their change. Another user with the necessary rights can then decide whether the change is useful or not. Depending on his decision, he can merge (accept) or reject the changes. The described process is the typical approach to how the community develops for an open-source project.

1.2 Problem

The described workflow for managing repositories on GitHub has some significant disadvantages. Firstly this approach is not necessarily decentralised or democratic. Very few people usually have the necessary rights to merge pull requests, and they can decide over the head of the general community whether to merge or reject a pull request. So it doesn't matter what the general community thinks, as long as the administrators have a different opinion. Rejecting good or useful pull requests is bad, but not a direct threat to the project. The opposite is to merge a critical bug into the main repository, which can cause enormous damage, as in the example of the Heartbleed bug in the Open SSL repository [15]. This danger exists mainly because it only takes one person with the necessary rights to overlook the bug and decide to merge the flawed pull request. Less critical problems, which nevertheless complicate the work in the communities of open-source projects, are the lack of

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

initiative for developers to review pull requests or to create pull requests themselves. If an open-source repository is not financially supported by a company or a large community, they usually live on developers who work on these projects in their spare time, which means that further development sometimes takes a very long time.

1.3 Solution

To solve these problems, this paper presents the development of a chrome browser extension that uses the GitHub API [7] and smart contracts [9] on the Ethereum blockchain [4] to enable decentralised management of pull requests in GitHub repositories with a financial incentive for the community. Using the Ethereum Blockchain, the community can vote on which pull requests should be merged or rejected. The aim is to merge good requests and reject bad ones by rewarding good decisions and punishing bad ones. In addition, the community can use a crowdfunding mechanism to pool Ether, the native currency of the Ethereum blockchain, to pay developers for solving problems or bugs. This creates a free market with Ether as its currency where supply and demand form an initiative for solving the tasks. The functionality of the protocol on which the browser extension is based is explained in detail in *3. Protocol*. Our goal with this browser extension and this paper is to solve the problems mentioned and to improve the way open-source development is done.

2 RELATED WORK

Related work falls into three areas: *repository governance*, *decentralised voting* and *developer initiative*. Previous work has already looked at decentralised approaches to manage GitHub repositories. One work pursues the idea that a repository always belongs to an ownerless protocol and that pull requests can only be merged or rejected by the community via this protocol. This should prevent the danger that users with writing rights misuse them [20].

Furthermore trustless and decentralised voting on the blockchain is a topic that has been addressed in several papers and which also plays a critical role in this paper [16, 20].

In addition, the lack of financial incentive to implement problem fixes or changes in a open-source repository is an issue that has been researched. The research looked at what is important to the developer who solves the bounty and what is important to the financial backer of the bounty. [21]

3 PROTOCOL

The described browser extension consists of two parts. A frontend that allows users to vote on pull requests or to back a bounty financially, and a protocol that runs locally in the browser extension but uses the GitHub API and the Ethereum blockchain to take over actions such as distributing stakes or merging and rejecting pull requests by the vote outcome. The protocol is the actual solution to how the control of the pull request management is decentralised, the frontend on the other hand provides the possibility to interact with the protocol. In the process of development and research, a total of two protocols were designed, with the second protocol being an extension of the first. Adding further functionalities that we subsequently deemed as important. This section introduces both protocols and their differences on a non-engineer level.

3.1 First protocol

The first protocol starts with a developer deciding to develop a change and creating a pull request so that it can be merged into the main branch of the repository. After the pull request has been created, any community member, i.e. anyone who follows the repository, can start the voting phase for the pull request. In this step, a smart contract is created which is used for the later voting and the distribution of the stakes. The community can then vote for or against the pull request for a certain period of time. When someone submits a vote, they must weight it with a stake. Ether is used as a stake and the more Ether is staked on a vote, the higher its weight is. After the voting period has expired, the protocol adds up the stakes for and against the pull request.

If the majority of the stakes, i.e. more than 50 percent, have staked for the pull request, it is merged otherwise it is rejected. It is important to note that the stake of the votes cannot be changed subsequently, once the stake has been sent to the smart contract, it is held until the voting phase is finished. After that, the majority stakers divide the stakes of the minority stakers among themselves. They get a percentage share of the minority stakes in relation to their stakes in the winning pool. For example, whose decision has won and who represents 50 percent of the majority stakes, receives half of the minority stakes. The complete process is graphically illustrated in *Figure 1*.

3.2 Second protocol

As already described, the second protocol is an extension of the first protocol. The decision why we have expanded the first protocol is explained in the *5. Discussion*. In general, however, it can be said that we consider the extension as an improvement to the first protocol. The protocol can be divided into four phases, which are as follows:

(1) Initialization and bounty funding
(2) Issue claiming and solution
(3) Pull request voting
(4) Evaluation and distribution

During the first phase, the smart contract is created after the initialization of the bounty process. With the help of this smart contract, all further processes such as crowdfunding the bounty, claiming the issue, voting on the pull request, and distributing the rewards and stakes are organized persistent, decentralised and trustless via the Ethereum blockchain. The details of the mentioned processes and the individual phases are explained in this section.

Phase 1: Initialization and bounty funding
The protocol workflow first starts independently when a community member creates an issue, in the GitHub repository, related to a problem or a new feature. If the owner or an authorized user of the open-source repository thinks that the issue is reasonable, they can initiate a bounty process for the issue. The community can then

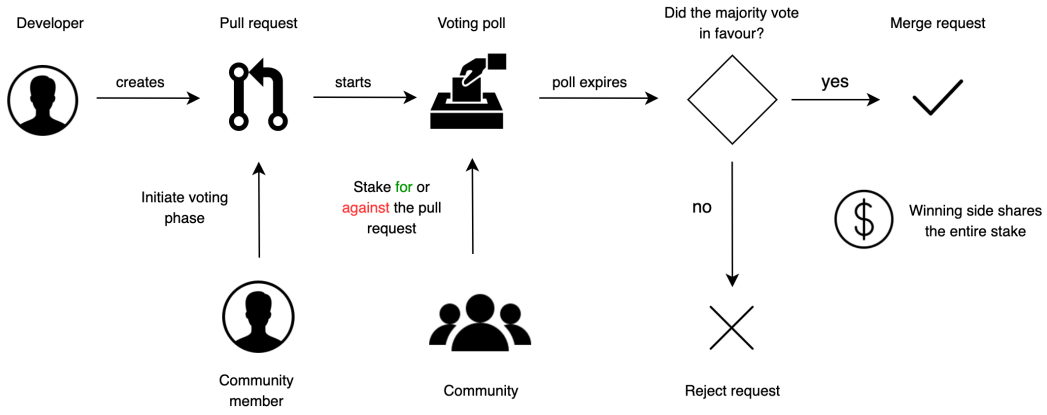

Figure 1: First version of the extension protocol

fund the bounty with their own Ether to motivate a developer to solve the issue. A developer can potentially receive this bounty as a reward if he solves the issue in a pull request and the community accepts it. The Ether paid into the bounty is held in a smart contract for a certain period of time so that the bounty cannot be negatively manipulated in the short term. If no developer wants to solve this issue and the period described above has expired, the community members that funded the bounty receive their shares back.

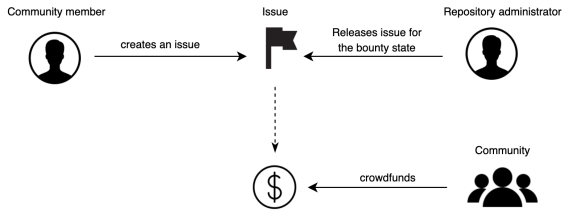

Figure 2: Workflow of the first phase

Phase 2: Issue claiming and solution
While the community collects the bounty, a developer can always decide for himself whether the bounty is high enough for him to solve the issue. As soon as the reward is high enough that a developer would solve the issue for this amount, he can claim the issue for him. But the following must be given. No other developer has already reserved the issue, an issue can only be processed by one developer. Furthermore, the developer has to pay a collateral to claim the issue, which he can lose if he either does not submit a solution within a given period of time or the solution gets rejected by the community. Once the issue is reserved by a developer, he has a certain period of time in which he has to program a solution and provide it as a pull request. If the developer does not provide a solution or it is rejected by the community, he loses his collateral, which is then sent to the bounty. The issue is then set back to the claiming phase.

Phase 3: Pull request voting
This process is the same as the first variant of the protocol, the only differences are that the voting process starts, when the developer submits his pull requests and that the staker must include a comment on his vote as to why he voted the way he did. This comment is then posted in the comment section of the pull request.

Phase 4: Evaluation and distribution
As with the first protocol, the minority stakes are transferred to the majority stakers after the voting phase. The difference here is that if the pull request is accepted by the community, the developer receives his bounty and collateral. However, if the pull request is rejected, the developer loses his collateral, which gets allocated to the bounty and the protocol puts the issue back into the claiming phase.

Algorithm 1: Stake distribution algorithm using the example of an accepted pull request

Input: All votes *votes*; Developer *dev*; Pro votes *provotes*; Bounty *bounty*; Collateral *collateral*

```
fullStake ← SumStakes(votes);
proStake ← SumStakes(provotes);
foreach vote ∈ provotes do
    share← (vote.weight ÷ proStake) · fullStake
    Transfer(vote.address, share);
end
Transfer(dev.address, bounty + collateral);
```

4 IMPLEMENTATION

The implementation of the chrome browser extension consists of a frontend and the in *3. Protocol* described protocol. In this section, the technical implementation of the two components will be explained. Furthermore, for information purposes, when the protocol is discussed, it is referring to the second version of the protocol. The mentioned chrome extension frontend has been created according to the current standard with HTML, CSS, JavaScript and the Chrome extension API [1]. In summary, the chrome extension

was developed as a kind of web application with a few specific limitations. On the other hand, the development and implementation of the protocol required much more thought. The protocol must connect the Ethereum blockchain with GitHub and mediate the data between the two endpoints.

The collaboration with the Ethereum blockchain was done using the JavaScript library web3js [13]. This library enables interactions with an Ethereum node and thus with the blockchain using HTTP. This allows the chrome extension to retrieve general information from the blockchain, but also to write data and even create smart contracts. Querying blockchain data does not require any special settings, as it is a read-only request. All write actions on the blockchain, such as voting on a pull request or generating a smart contract, require a wallet in the extension, as you have to pay transaction fees for the actions.

No special library is needed for the GitHub connection, this is done via the API provided by GitHub [7]. This can also be accessed using HTTP, which is needed to fetch information about repositories, to create voting comments or to either merge or reject a pull request. In contrast to working with the Ethereum blockchain and smart contracts, many of the requests against the GitHub API require an authentication token, which can be fetched via the GitHub OAuth API [10]. The GitHub OAuth is used as a login in the extension, which means that the extension does not need its own login and can therefore fetch the authentication token. However, the token alone is not sufficient as authentication for every request; for merging and rejecting pull requests, a developer token must also be stored in the extension, which can be obtained from the GitHub account settings.

Probably the most difficult aspect of the implementation of the chrome extension is the conclusion of the voting phase, i.e. *Phase 4: Evaluation and distribution*. Contrary to the other phases and actions, this process must happen automatically as soon as the voting phase ends. However, since we only have a client-side program, we can only detect this when one of the developed chrome extensions is running. For this reason, after the GitHub OAuth login, the extension checks if there are completed pull request votes and if so, the extension triggers the described workflow *Phase 4: Evaluation and distribution* for the pull request in the background. Then the necessary requests are sent to the smart contract and the GitHub API.

5 DISCUSSION

Protocol update
In the course of implementing the first protocol, we found some major flaws in it, which is why we have modified and extended it as described in *3. Protocol*. In a conversation with several software developers, the question came up how to protect the protocol from malicious users creating pull requests and releasing them for voting. For this reason, it was decided that the release for voting cannot take place through the general community, as otherwise, the danger of such attacks is too great. In the second iteration of the protocol, pull requests are released by users with administrator rights by opening the bounty for them. At this point, decentralisation must be reduced in order to provide more security. In addition, the idea came up to use the bounty system to create a further incentive for development and maintenance in open-source projects, which was not given in the first protocol. This adaptation should lead to a free market for the development of features or the solution of issues. This also aims to improve the quality of pull requests, as developers have to put effort into their development in order for the pull request to be accepted and for them to receive both their collateral and the bounty.

Advantages and disadvantages of ethereum
At the time of this research paper, Ethereum is one of the largest blockchain platforms with over one million daily transactions [3]. While Ethereum is constantly being developed and improved due to the size of the community, the main blockchain is currently not suitable for this application. Currently, the Ethereum fees for a transaction are around 23.70$ [2]. This means that you have to pay this fee for every action, whether voting or contributing to an issue. Unfortunately, this is not sustainable, although future developments of the Ethereum protocol may change this. Currently, this problem is being avoided by working on the Ethereum testnet Sokol [12]. However, this cannot be used in live operations because the Ether on this chain has no value.

Synchronisation of voting results
The current browser extension approach has the advantage that everything is started from the extension and you can react to misbehaviour on the part of the web3js library or GitHub API. However, this approach has difficulties especially when completing voting polls. Because there is no central server that carries out these changes, there is always the danger that two clients simultaneously initiate this process in the background. Although no direct damage can be done, unnecessary transaction fees may incur. In addition, it is problematic that if a voting phase expires but no one starts the extension for a longer period of time, the associated pull request is not merged or rejected and the stakes and bounties are not sent. Also, this implementation requires that community members are willing to have their own wallets on Ethereum and pay transaction fees for their actions.

6 CONCLUSION AND FUTURE WORK

The presented implementation of the chrome browser extension and the underlying protocol forms a solid basis for a new kind of open-source development via GitHub. With the help of new technologies such as the blockchain, new possibilities are now available for setting up centralised software in a more decentralised and thus also more democratic way. As described above, we see some problems with the current implementation and there is also the question of how open the open-source community is to this approach.

For these reasons, we see two aspects that need to be explored in future work. The first aspect would be a qualitative study on how open-source developers and community members feel about this decentralised development approach and what they think about such things as blockchain and transaction fees. The second aspect would be to research whether a server-client architecture is more

suitable for this application, as was done in the work *Development of a blockchain based access control protocol for GitHub repositories as open-source project* [20]. One possibility would be to adapt the server to the requirements of the browser extension and connect the extension to it. This could be researched to see if this approach leads to better reliability and solves some of our described weaknesses.

REFERENCES

[1] [n.d.]. Chrome API Reference. https://developer.chrome.com/docs/extensions/reference/ [Online; Retrieved 20.02.2021].

[2] [n.d.]. Ethereum Avg. Transaction Fee historical chart. https://bitinfocharts.com/de/comparison/ethereum-transactionfees.html [Online; Retrieved 20.02.2021].

[3] [n.d.]. Ethereum Daily Transactions Chart. https://etherscan.io/chart/tx [Online; Retrieved 20.02.2021].

[4] [n.d.]. Ethereum developer resources. https://ethereum.org/en/developers/ [Online; Retrieved 20.02.2021].

[5] [n.d.]. Git scm. https://git-scm.com/ [Online; Retrieved 20.02.2021].

[6] [n.d.]. GitHub about. https://github.com [Online; Retrieved 20.02.2021].

[7] [n.d.]. GitHub REST API. https://docs.github.com/en/rest [Online; Retrieved 20.02.2021].

[8] [n.d.]. Gitstar Ranking. https://gitstar-ranking.com/ [Online; Retrieved 20.02.2021].

[9] [n.d.]. Introduction to smart contracts. https://ethereum.org/en/developers/docs/smart-contracts/ [Online; Retrieved 20.02.2021].

[10] [n.d.]. Managing OAuth Apps. https://docs.github.com/en/developers/apps/managing-oauth-apps [Online; Retrieved 20.02.2021].

[11] [n.d.]. Permission levels for a user account repository. https://docs.github.com/en/github/setting-up-and-managing-your-github-user-account/permission-levels-for-a-user-account-repository [Online; Retrieved 20.02.2021].

[12] [n.d.]. Sokol blockexplorer. https://blockscout.com/poa/sokol/ [Online; Retrieved 22.02.2021].

[13] [n.d.]. web3.js - Ethereum JavaScript API. https://web3js.readthedocs.io/en/v1.3.0/ [Online; Retrieved 20.02.2021].

[14] MDN Web Docs. [n.d.]. Contributing to the Mozilla code base. https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Introduction [Online; Retrieved 20.02.2021].

[15] Melissa Iori. [n.d.]. Heartbleed Keeps Flowing-Open Source Security. ([n. d.]).

[16] David Khoury, Elie F Kfoury, Ali Kassem, and Hamza Harb. 2018. Decentralized voting platform based on ethereum blockchain. In *2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET)*. IEEE, 1–6.

[17] Redhat. [n.d.]. What is open source? https://www.redhat.com/en/topics/open-source/what-is-open-source [Online; Retrieved 20.02.2021].

[18] Maha Shaikh and Ola Henfridsson. 2017. Governing open source software through coordination processes. *Information and Organization* 27, 2 (2017), 116–135.

[19] torvalds. [n.d.]. GitHub repository of the linux kernel. https://github.com/torvalds/linux [Online; Retrieved 20.02.2021].

[20] Torben Ulrich. 2020. Development of a blockchain based access control protocol for GitHub repositories as open-source project. (2020).

[21] Jiayuan Zhou, Shaowei Wang, Cor-Paul Bezemer, Ying Zou, and Ahmed E Hassan. 2019. Bounties in open source development on github: A case study of bountysource bounties. *arXiv preprint arXiv:1904.02724* (2019).