The Liquid Democracy Journal

on electronic participation, collective moderation, and voting systems

Issue 6

Berlin, 2018-07-27

The Liquid Democracy Journal is dedicated to the idea of Liquid Democracy, which is a democratic principle that uses transitive delegations to unite the best of direct and representative democracy.

But this journal is not just limited to Liquid Democracy; it also covers those topics coming up when implementing it: **electronic particatipation, collective moderation, and voting systems.**

The Liquid Democracy Journal
on electronic participation, collective moderation, and voting systems

Issue 6, Berlin 2018-07-27 (electronic version 2019-02-22)

Published by:	Interaktive Demokratie e. V., Berlin, Germany
Edited by:	Jan Behrens Axel Kistner Andreas Nitsche Björn Swierczek
Contact editors at:	editors@liquid-democracy-journal.org *For unsolicited sent-in works neither the editors nor the publisher take any responsibility.*
Subscribe at:	http://www.liquid-democracy-journal.org/
Archive available at:	http://www.liquid-democracy-journal.org/

ISSN-L:	2198–9532
ISSN print version:	2198–9532
ISSN electronic version:	2199–1758

Editorial

by the Editors, Berlin, July 27, 2018

In our last issue, we wrote about "The Origins of Liquid Democracy" mentioning Rob Lanphier's assessment that verification of identity is of such general importance for many internet applications that he expected a solution to emerge. But almost 25 years after this assessment, identification remains one of the biggest challenges for online decision making systems. For many, this fact is hard to accept, and some even suggest to allow non-verified voters to take part for the sake of inclusion while trying to limit the influence of such voters and encourage verification by assigning different voting weights.

In this issue we publish "A Mathematical View on the Sockpuppet Problem" which was written during the course of a scientific cooperation. It contains a (rather trivial) mathematical proof, explaining why allowing non-verified voters is a bad idea, which was to our surprise all but obvious to scientists of some disciplines.

Right after the publication of Issue #5 of this journal, the Association for Interactive Democracy joined a Digital Democracy Workshop of the Centre for Digital Culture at King's College London which was hosted by Newspeak House London. Multiple stakeholder engagement workshops in London, Milan, San Donà di Piave and Turin have been organized.

A pre-release version of LiquidFeedback 4.0 is part of the WeGovNow pilot platform in the City of Turin. In this platform, LiquidFeedback also serves as the authorization server for all other applications, which is also aiming at solving the general problem of voter identification on the net. Turin has been the first medium scale use case for the new unified user management with some 10,000 registrations within the first month. Some weeks later the City of San Donà di Piave started their WeGovNow platform using the new role accounts feature provided by LiquidFeedback's unified user management. This issue of the Liquid Democracy Journal will inform about the "Unified User Management with LiquidFeedback".

The principal theme of this issue is the "Roadmap to a Decentralized LiquidFeedback" using "The LiquidFeedback Blockchain". In this context, one article highlights the topic of "Decentalized Accreditation" and a difficult challenge regarding correctability of online voting is discussed in our article "Practical Consequences of Arrow's Theorem for Open Electronic Voting". Four articles and a working prototype of the LiquidFeedback Blockchain describe the motivation, the considerations, the solution, and the challenges.

The Editors

Corrigendum

In the first issue (Issue #1) of "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", in the article "The Evolution of Proportional Representation" [Evolution] it was wrongly claimed that the Harmonic Weighting algorithm would allow

»any group of size P · M / (1+N) to place M initiatives of a set S amongst the first N positions, if all members of the group support all initiatives in that set (S), and no member of the group supports an initiative outside the set which gains a display position amongst the first N positions.«

[Evolution, p.35]

For the proof, the article references to the book "The Principles of LiquidFeedback" [PLF]. In this book, a proof is given in the footnote on page 78. This proof, however, only proves a weaker statement covering

»those cases where parts of the minority support initiatives that are not supported by all members of the minority (i.e. initiatives that are not in S), as long as those other initiatives which are supported by parts of the minority do not gain a better display position than [any of] the best-ranked M initiatives of S.«

[PLF, p.78] [Errata]

Furthermore, [Aziz, Example 2] shows that the stronger statement wrongly used in our article "The Evolution of Proportional Representation" is indeed violated by Harmonic Weighting. We therefore must correct the statement that article. Instead of
»no member of the group supports an initiative outside the set which gains a display position amongst the first N positions«
read
»no member of the group supports an initiative outside the set which gains a better display position than any of the best-ranked M initiatives of S«
and skip the rest of the paragraph.

The claim in our book "The Principles of LiquidFeedback" (with the existing [Errata]) is correct. We would like to thank Markus Brill from the Technische Universität Berlin for his help and for kindly referring us to the counterexample (Example 2 in [Aziz]).

For sources, see right page

Read in this Issue:

[Aziz] Haris Azis: "A Note on Justified Representation Under the Reverse Sequential PAV rule", working paper, 2017. Retrieved on 2018-02-18 from: http://www.cse.unsw.edu.au/~haziz/invseqpav.pdf

[Errata] "Errata as of 2016-12-22" for 'The Principles of LiquidFeedback'. Available at: http://principles.liquidfeedback.org/errata/

[Evolution] Jan Behrens: "The Evolution of Proportional Representation in LiquidFeedback". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 1, March 20, 2014, pp. 32-41. ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/1/The_Liquid_Democracy_Journal-Issue001-04-The_evolution_of_proportional_representation_in_LiquidFeedback.html

[PLF] Behrens, Kistner, Nitsche, Swierczek: "The Principles of LiquidFeedback". ISBN 978-3-00-044795-2. Published January 2014 by Interaktive Demokratie e. V., available at http://principles.liquidfeedback.org/

Unified User Management with LiquidFeedback

by Jan Behrens and Björn Swierczek, Berlin, July 27, 2018

This article will present a technology based on the OAuth 2.0 standard to allow user identification on the internet: LiquidFeedback acts as an identity server, and, given a proper accreditation process for LiquidFeedback, other third-party software may use LiquidFeedback to verify identities when collecting quantified user input (single-sign-on with identity verification). This allows organizations or political administrations to build an extensible ecosystem of internet applications that are invulnerable to the sockpuppet problem where users create a number of (fake) internet identities to unfairly increase the impact of their postings or votes.

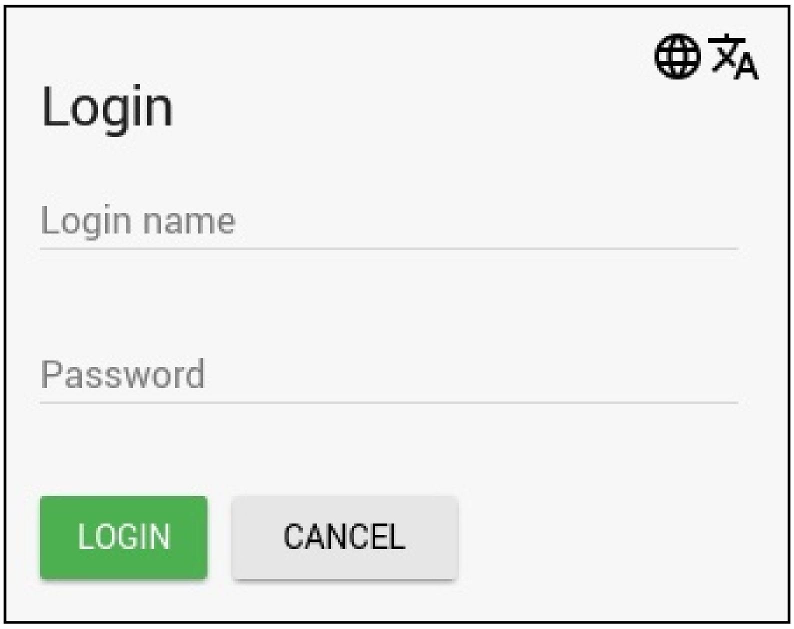


Figure 1: Login dialog of LiquidFeedback's unified user management

A necessary prerequisite to using this technology is a proper accreditation process of all participants who shall be eligible to take part in the deliberation process. This accreditation process is out of scope of this article and will differ according to the concrete application scenario.

For the most part of this article, we assume the reader is familiar with single-sign-on technologies as well as OAuth 2.0 as specified in RFC 6749.

This article is in large part based on the UWUM Work Report [UWUM]. LiquidFeedback's OAuth 2.0 server support as well as the UWUM Work Report have been contributed by FlexiGuided GmbH as part of the WeGovNow project which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 693514.

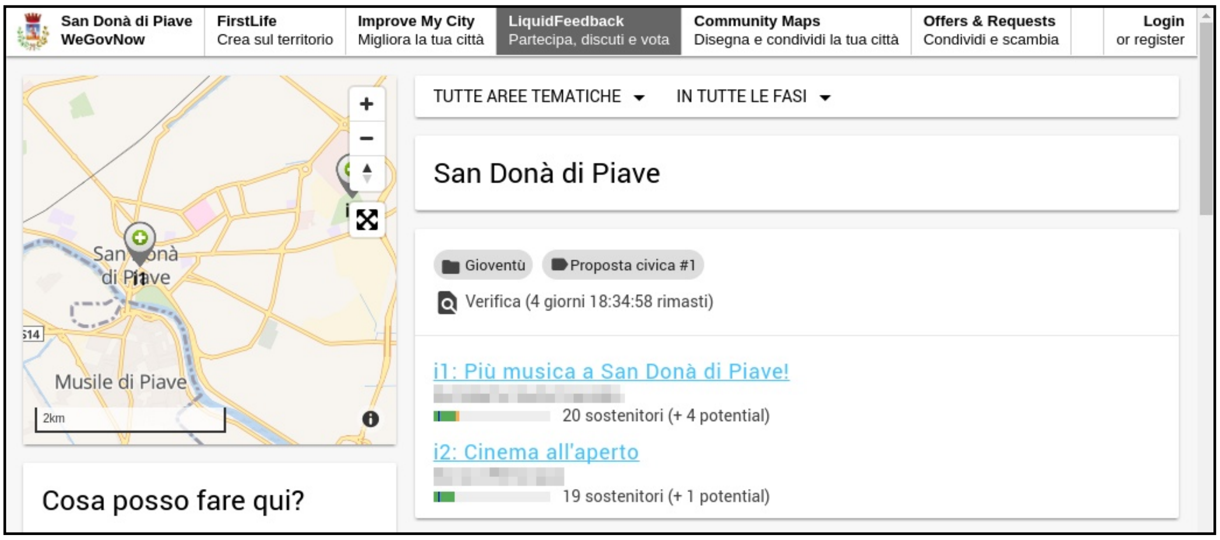
Motivation

In the previous issue of The Liquid Democracy Journal, we mentioned that a central problem of internet participation systems had already been stated in 1995 by Rob Lanphier: without a verification of identities, you cannot ensure the basic democratic principle of "one person – one vote". [Origins] Without that principle, even a single person may be able to manipulate the results of a vote. [Sockpuppet]

» *The main problem facing electronic voting on the Internet is verifying that one person gets one vote, and that all people are represented (even those without Internet access). Verification of identity is a problem that plagues many applications on the Internet (such as making purchases on the net, or filing taxes on the net), and so this one will likely be solved regardless of whether electronic voting makes it an issue.*«

Rob Lanphier, 1995
[Lanphier]

These findings cannot just be applied to electronic voting but are also valid for any other participation solution incorporating any form of opinion formation through counting of ratings (e.g. "likes", "+1" or "thumbs up"). Consequently, it is not possible to quantify people's opinions on the internet without verifying their identity.



San Donà di Piave

Gioventù · Proposta civica #1

Verifica (4 giorni 18:34:58 rimasti)

i1: Più musica a San Donà di Piave!
20 sostenitori (+ 4 potential)

i2: Cinema all'aperto
19 sostenitori (+ 1 potential)

Figure 2: Use case "WeGovNow – We Government Now!" serving different applications "under the same roof" using LiquidFeedback's Unified User Management

For the sake of inclusion, it has been proposed to allow people to participate anonymously with a reduced voting weight. Disregarding that this is already a violation of treating each participant equally, it can furthermore be proven mathematically that reducing the voting weight of anonymous participants to a fixed (but non-zero) value cannot solve the problem that creating a finite amount of so-called "sockpuppets" (fake identities) is sufficient to overrule the results of a vote. [Sockpuppet]

Lanphier was optimistic in regard to a solution of the identification problem on the internet, but until today (more than 20 years later), there is no standardized way to identify people. Many corporations are using their own solutions for identifying people, e.g. as customer or owner of a specific bank account, some governments have developed electronic identity systems for use with the internet, but there is no generally available solution which is practically feasible and intended to be used to authenticate and authorize participants for democratic participation processes using the internet.

When talking about solutions for verifying one's identity on the internet, we have to distinguish between:

- ***technologies*** *(e.g. open source software, hardware appliances, standardized protocols, etc.)*

and

- *their particular* ***application*** *(e.g. solutions created by particular organizations or states).*

In the remainder of this article, we present a technology based on the OAuth 2.0 standard [RFC 6749] that has been developed as part of the EU-funded WeGovNow project and which has been incorporated into the LiquidFeedback software version 4.0, which is to be published soon. Ensuring that each person has at most one account in LiquidFeedback will automatically enable other (third-party) software to use LiquidFeedback to identify their users and thus avoid any problems with sockpuppets.

Existing alternatives

There are existing standards for single-sign-on procedures on the internet, and these also allow user identification. As we will explain in the following paragraphs, none of these solutions have matched our needs.

The most notable standard for single-sign-on (SSO) is the "OpenID Connect" standard. [OpenID] Unfortunately OpenID Connect is a rather complex standard and requires extra implementation work compared to the underlying OAuth 2.0 standard (which is much simpler to adopt).

Another alternative in mind was pure OAuth 2.0, but OAuth 2.0 has been created with a different application scenario and is out-of-the-box not suitable to provide a secure user identification and authentication. Lodderstedt, McGloin, and Hunt state in [RFC 6819] ("OAuth 2.0 Threat Model and Security Considerations") that clients which wish to implement a user login should rather use "an appropriate protocol" such as OpenID or SAML instead of OAuth 2.0.

Nonetheless, using OAuth 2.0 with its "Authorization Code Flow" can still provide a lightweight way to create a secure authentication and identification service. Only a few (standard compliant) extensions are necessary to allow applications to query the user's identity. Similar approaches have been taken by platforms such as Facebook ("Facebook Login") but are often tied to a particular platform and cannot be installed in a separate realm with a secure and/or application-specific accreditation process.

Despite problems with existing solutions, it was our intention to stick to existing standards where appropriate. Even with necessary extensions, we managed to make LiquidFeedback's identification service to be fully compliant with the OAuth 2.0 specification [RFC 6749].

From OAuth 2.0 to a single-sign-on solution

In order to use OAuth 2.0 as a single-sign-on solution, LiquidFeedback takes the role of the "authorization server" according to [RFC 6749] while other (third-party) components can take the role of "clients" and "resource servers".

However, OAuth 2.0 provides different modes of operation (e.g. the "Authorization Code Flow" and the "Implicit Flow") and is by itself not sufficient for user authentication. Both the Authorization Code Flow and the Implicit Flow could be extended to provide user authentication and thus allow to implement a single-sign-on (SSO) system where third-party services on the internet can identify users. Because the Implicit Flow would require additional security mechanisms to be implemented at client side (where bad implementations result in security vulnerabilities), [RFC 6749, subsection 10.16] LiquidFeedback extends the Authorization Code Flow for user identification purposes.

In short, the Access Token Response of the OAuth 2.0 Authorization Code Flow gets extended by simply returning an additional field "member_id" which identifies the signed-in user. For details, refer to [UWUM].

(Dynamic) Client registration

The OAuth 2.0 standard [RFC 6749] requires a procedure called "client registration" without defining how this procedure is to be carried out. In the following, we explain about the problems that can come along with such a client registration and how we deal with them.

Client registration means that any component wishing to use the authorization server needs to be known to the authorization server beforehand. There are a couple of well-founded security reasons why this client registration is necessary. [UWUM, subsection 2.4.2] Nonetheless, this client registration is one of the biggest drawbacks of the OAuth 2.0 protocol: either clients can register without manual verification and approval by the operator of the authorization server, in which case the security concerns are often not fully addressed, or a manual verification is conducted in which case additional workload is imposed on the operator of the authorization server and/or certain clients might be excluded from using the authorization server. Furthermore, when using OAuth 2.0 for API authentication (which is its original field of application), manual client registration prohibits the creation of generic clients (e.g. alternative user interfaces) which may directly connect to any instance of a particular software installation (e.g. connect to LiquidFeedback instances in different municipalities or organizations).

The OAuth 2.0 specification [RFC 6749] does not care about these problems at all and simply states that client registration is out of scope of the standard and would "typically involve end-user interaction with an HTML registration form":

»*Before initiating the protocol, the client registers with the authorization server. The means through which the client registers with the authorization server are beyond the scope of this specification but typically involve end-user interaction with an HTML registration form.*«

[RFC 6749]

This form of client registration, however, is only suitable for a service-centered approach where a software provides only a single service (e.g. Facebook, Google, Twitter, etc). When considering an open source ecosystem, we expect more than one installation of a particular technological solution. It is therefore not sufficient to register a client at a single service provider if this client shall be usable for any authorization server using the same (open source) software.

One possible solution would be the creation of a central (i.e. world-wide) client registry. Such a central client registry, however, could be a single point of failure and would empower a central authority (e.g. the Public Software Group) to control usage of the protocol (e.g. it would be possible to block certain clients). We consider this approach contrary to the concepts of open source and open data.

Therefore, we implemented a dynamic client registration protocol that keeps implementational complexity at a minimum while providing good security properties. Dynamic client registration is described in subsection 2.4.2 of [UWUM].

Further tweaks

LiquidFeedback's OAuth 2.0 server implementation can also be used for all other purposes that OAuth 2.0 may be used for. Further improvements have been made to extend the basic OAuth 2.0 workflow. For example, subsection 10.4 of [RFC 6749] suggests refresh token rotation to provide better security properties but does not specify how old refresh tokens are invalidated. Always revoking the old refresh token after transmission of a new refresh token can have a bad effect on system stability, considering that transmissions might not be processed properly by the receiver. Furthermore, multiple backends of a client could simultaneously access the OAuth 2.0 token endpoint. Such legit accesses by two legit backends of the same client would need to be distinguished from accesses by a legit client and a malicious third party who obtained a copy of a refresh token. Subsection 2.15 of [UWUM] explains the mechnanisms employed by LiquidFeedback to mitigate the risk of refresh token abuse while solving the problems of race conditions during refresh token rotation.

Another improvement, for example, is to allow downgrading access token scopes to allow meta-API providers as explained in subsection 2.14 of [UWUM].

Summary

Arguably, the biggest problem with the OAuth 2.0 standard is that it only provides a framework for authorization while keeping cruicial details undefined such as client registration. Those details, however, must be well-defined in order to provide a secure yet flexible authentication service.

Integrating an OAuth 2.0 server with appropriate extensions into LiquidFeedback automatically empowers all organizations or political administrations which use LiquidFeedback to provide identity services to third-party applications, even those unknown to the organization at time of deployment.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 693514.

[Lanphier] Rob Lanphier: "A Model for Electronic Democracy?" Published at http://robla.net/1996/steward/ (accessed 2016-04-19) and also archived on October 27, 2005, available at http://web.archive.org/web/20051027051842/http://robla.net/1996/steward/ (accessed 2016-04-19)

[OpenID] The OpenID Connect 1.0 specification, available at: http://openid.net/connect/ (accessed 2018-02-10)

[Origins] Jan Behrens: "The Origins of Liquid Democracy". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 5, May 11, 2017, pp. 7-17. ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/5/The_Liquid_Democracy_Journal-Issue005-02-The_Origins_of_Liquid_Democracy.html

[RFC 6749] D. Hardt (Ed.): "The OAuth 2.0 Authorization Framework", October 2012. Available at: https://tools.ietf.org/html/rfc6749

[RFC 6819] Lodderstedt (Ed.), McGloin, Hunt: "OAuth 2.0 Threat Model and Security Considerations", January 2013. Available at: https://tools.ietf.org/html/rfc6819

[Sockpuppet] Jan Behrens: "A mathematical view on the sockpuppet problem". April 12, 2016, published July 27, 2018 as Appendix A to "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 6, July 27, 2018. ISSN 2198-9532. Published by Interaktive Demokratie e. V.

[UWUM] Behrens, Kistner, Nitsche, Swierczek: "Work report on Unified WeGovNow User Management (UWUM) development", December 12, 2016. Published as Appendix 2.1.2 (pages 125-151) to the WeGovNow Deliverable D3.1 ("Consolidated System Architecture") available at: https://ec.europa.eu/futurium/sites/futurium/files/d3.1_693514_consolidated_system_architecture_.pdf (also included as Appendix B to "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 6, July 27, 2018, pp. 39-50, ISSN 2198-9532, published by Interaktive Demokratie e. V.)

Roadmap to a Decentralized LiquidFeedback

by Andreas Nitsche, Berlin, July 27, 2018

With LiquidFeedback published in late 2009 and the book “The Principles of LiquidFeedback” published in January 2014, we, the authors of LiquidFeedback, presented specific rules of procedure for a democratic process, providing every participant with truly equal rights to the maximum known extent while maintaining feasibility and effectiveness also in cases when the number of participants is huge. [PLF, Postface] Even though recorded votes ensure that any manipulation of the outcome can be detected by the participants, LiquidFeedback (as of now) still depends on a central authority (e.g. an organization or state administation) to host the system. With this article, we present a roadmap for a future LiquidFeedback that doesn't depend on a central authority but can be operated decentralized by any group of people who decide to do so.

The overall idea

The already existent LiquidFeedback proposition development and decision making process allows a (previously defined) group of people to enter a fair proposition development and decision making process where each participant gains equal rights to the maximum possible extent. The process is suitable for a broad range of applications such as policy making, technical standardization, (democratic) product development, or collaborative publications.

Currently, the correct execution of LiquidFeedback's rules of procedure is carried out by a centralized server infrastructure to be operated by an organization or political administration which wants to provide the system to the participants. The new idea is to empower the participants to not only *engage* in the process but to also be able to technically *execute* the process in a joint effort.

Not just changes to the rules of procedure or any document jointly created with such a future LiquidFeedback system would depend on a majority of people supporting these changes, but the actual execution of these changes would be carried out by a distributed system, hosted by eligible voters, that could interpret properly formed proposals automatically (e.g. enabling a new software component or changing a paragraph of a policy automatically).

Requirements

When creating such a system, there are a couple of requirements to meet. First of all, we need to ensure that decentralization doesn't come at the cost of sacrificing the fairness of the existent LiquidFeedback process. It should also be ensured that the process is transparent for all participants. Only transparency ensures that compromized hard- or software (e.g. due to hacking attempts or technical failures) can be detected; i.e. the results of the system shall be verifiable by the participants. In case of errors or manipulations, it should be possible to correct the results.

Despite transparency and correctability, which allow us to handle errors of the system, we still expect such a system to be as secure as possible, i.e. it should be as difficult as possible to manipulate the system even if it cannot be completely outruled. Furthermore, such a system should be scalable and allow millions of people to take part in democratic decisions at the same time, if desired.

Challenges

In order to fulfill the previously stated requirements, proper algorithms have to be selected and/or created where necessary. A data model as well as algorithms for data distribution and finding a consensus have to be defined. In some cases we can rely on existing technologies, in other cases we need to modify existing technologies or invent new algorithms, especially where existing solutions are not suitable for distributed application or are environmentally hazardous (e.g. blockchain with proof-of-work). The system needs to be fault tolerant and (as previously mentioned) correctable, e.g. it must be possible to recover from a compromized end user device due hacker intrusion (key revocation and recovery).

A couple of theoretical findings as well as technological insights must be kept in mind when designing a decision making process, even more so when abstaining from relying on a central trusted authority.

Most notably Kenneth Arrow's impossibility theorem published in 1951 affects the design of decision making processes. From this theorem follows that when there are more than two voting options, tactical voting cannot be outruled completely. [Gibbard] [PLF, section 4.14] On the other hand, breaking down complex decisions into simple binary (i.e. yes/no) questions suffers the problem that the order of the questions has a massive impact on the overall outcome of the process. [GoD] Consequently, if we want to treat all participants of a democratic process equally, we need to allow more than two voting options on a ballot for a given issue. In turn, tactical voting has to be taken into consideration. To treat everyone equally, it is important to temporarily hide cast ballots. [PLF, section 4.14] [GoD] Doing this without a central (trusted) authority is a rather complex issue because we want no participant to be able to change his or her ballot based on the other ballots. Even worse, in some cases even withdrawing a ballot from a vote (dependent on the other voters' ballots) may lead to tactical advantages. [Moulin] The last article in this Issue #6 of "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems" will elaborate the serious consequences of these findings for all forms of electronic decision making systems. [Practical]

Another aspect to keep in mind is the overall complexity of the system. The above mentioned challenges will require sophisticated algorithms, so another important aspect to keep in mind is making sure the overall complexity of the system is as low as possible in order to allow easy maintenance and continuing revolution.

Existing technologies

In addition to new algorithms, a future decentralized LiquidFeedback will also be based on a couple of existing technologies. The currently existing version of LiquidFeedback already provides a process for proposition development and decision making that treats every participant equally and scales (in theory) for groups of unlimited size. [Infinite] [Limiter] However, current LiquidFeedback versions don't allow an automatic interpretation/application of proposals that were agreed upon. For example, the members of a political party may use LiquidFeedback to decide on changing their party program, but the actual change will need to be executed by a person designated to maintain the document. This limitation has partially been lifted by combining LiquidFeedback with revision control systems, which is a first step to automate the application of approved proposals in the decision making process. [RevisionControl]

Peer-to-peer networks for file sharing have been existent since the millenium change and are an important advance in distributed computing. While these networks allowed the distribution of data, they were unable to solve the problem of commonly agreeing on a particular state of data. Blockchains solved this problem by ensuring that a large portion of the participants work with and refer to the same working state.

To ensure that (in the long term) only one state exists within a decentralized network, blockchain systems usually rely on proof-of-work methods where energy (and therefore money) has to be invested on a well-defined mathematic challenge in order to create a new state within the network. Due to the bad environmental side effects and because of the risk that a malicious party could gain a majority of processing resources, an alternative to the proof-of-work scheme has been proposed, called proof-of-stake. However, applying these proof-of-stake algorithms (that do not rely on wasted energy) to traditional blockchains can cause problems because participants in the network have nothing to lose by creating conflicting blocks, thus effectively preventing a consensus within the network (which was the goal in the first place). Methods have been proposed to fix this problem by introducing a punishment for such behavior [EthereumWiki], but in the next article, we will show how a consensus can be achieved in a different way (that is, in our opinion, much easier). [LFB]

Proof-of-stake and trusting the majority

In order to be independent of a central authority, we need to make all operational decisions of the system in a decentralized way. In this context, each eligible voter shall have the same influence on all decisions being made within the system, including operational decisions such as a consensus on who cast a vote and who was late.

It should be noted that if we completely replace the central authority with an automatic, decentralized system where all participants hold the same stake (proof-of-stake with same stake for each eligible person), then it's up to the majority to properly execute the agreed upon rules. For the remainder of our considerations, we need to assume that the majority of people are willing to ensure the overall fairness of the process. This is not a major restriction because in traditional decision making processes, the proper execution of the rules of procedure usually depends on elected people (or persons appointed by elected people). If the majority of the electorate are malicious, every democratic system would fail. We therefore assume benevolence of a majority of eligible voters.

Accreditation

A necessary prerequisite for any democratic decision making process is the accreditation of the participants. There needs to be clarity who is eligible to vote and how to identify these people during the voting process. There needs to be a process allowing new people to gain eligibility to vote as well as a process to remove people's accounts (e.g. in case of death or leaving a political party). While the design of such an accreditation process in detail is out of scope of this article, we assert that it is made public (at least among the participants) who enters and who leaves the set of eligible voters.

Under the previously stated prerequisite that a majority of eligible voters are benevolent and further assuming that the accreditation process is designed in such a way that the electorate can verify the list of persons gaining or losing eligibility to vote, we can conduct a vote on each person to be added or removed from the list of eligible voters. In many cases, such a vote will not prompt the electorate to express their own opinion on accepting a person to enter or banning a person from the group of eligible voters, but may rather be a mere confirmation of the outcome of a previously defined process that has been carried out in the real world (e.g. appearing in front of an audience and paying the membership fee for an organization).

The article “Decentralized Accreditation” [Accreditation] in this Issue #6 will have a closer look at different choices on how to implement a decentralized accreditation process for determining, identifying, and authorizing the eligible voters.

Prototype

A working prototype to demonstrate the basic principles has been created and will be included in the electronic version of this Issue #6 as well as published on the website of the journal in advance. The next article will explain the prototype as well as a new method for merging conflicting blocks in a blockchain which helps to implement a proof-of-stake system that is environmentally friendly.

[Accreditation] Jan Behrens, Andreas Nitsche, Björn Swierczek: "Decentralized Accreditation". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 6, July 27, 2018, pp. 30-33. ISSN 2198-9532. Published by Interaktive Demokratie e. V.

[EthereumWiki] "Proof of Stake FAQ" on Ethereum Wiki on GitHub. Accessed on 2018-06-10 at https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ/ea47c31b36b00415fe5b54db36ddad43c9bf650e

[Gibbard] Allan Gibbard: "Manipulation of Voting Schemes: A General Result". In "Econometrica", Vol. 41, No. 4, July 1973, pp. 587–601. Published by the Econometric Society (Wiley-Blackwell).

[GoD] Jan Behrens: "Game of Democracy". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 2, October 7, 2014, pp. 11-22. ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/2/The_Liquid_Democracy_Journal-Issue002-02-Game_of_Democracy.html

[Infinite] Jan Behrens, Andreas Nitsche, Björn Swierczek: "A Finite Discourse Space for an Infinite Number Of Participants". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 4, July 28, 2015, pp. 42-52. ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/4/The_Liquid_Democracy_Journal-Issue004-02-A_Finite_Discourse_Space_for_an_Infinite_Number_of_ Participants.html

[LFB] Jan Behrens, Axel Kistner, Andreas Nitsche, Björn Swierczek: "The LiquidFeedback Blockchain". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 6, July 27, 2018, pp. 18-29. ISSN 2198-9532. Published by Interaktive Demokratie e. V.

[Limiter] Jan Behrens, Andreas Nitsche, Björn Swierczek: "LiquidFeedback's Issue Limiter". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 5, May 11, 2017, pp. 32-35. ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/5/The_Liquid_Democracy_Journal-Issue005-04-LiquidFeedbacks_Issue_Limiter.html

[Moulin] Hervé Moulin: "Condorcet's principle implies the no show paradox". In "Journal of Economic Theory", Vol. 45, Issue 1, June 1988, pp. 53-64. Published by Cornell University, Department of Economics.

[PLF] Behrens, Kistner, Nitsche, Swierczek: "The Principles of LiquidFeedback". ISBN 978-3-00-044795-2. Published January 2014 by Interaktive Demokratie e. V., available at http://principles.liquidfeedback.org/

[Practical] Jan Behrens: "Practical Consequences of Arrow's Theorem for Open Electronic Voting". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 6, July 27, 2018, pp. 34-37. ISSN 2198-9532. Published by Interaktive Demokratie e. V.

[RevisionControl] Björn Swierczek: "Democratic File Revision Control with LiquidFeedback". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 4, July 28, 2015, pp. 8-41. ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/4/The_Liquid_Democracy_Journal-Issue004-01-Democratic_Revision_Control_with_LiquidFeedback.html

The LiquidFeedback Blockchain

by Jan Behrens, Axel Kistner, Andreas Nitsche and Björn Swierczek, Berlin, July 27, 2018

In this article, we present the LiquidFeedback Blockchain, an alternative approach to the blockchain. Unlike proof-of-work schemes, which require mining, or many proof-of-stake schemes, which require special consideration of the nothing-at-stake problem where nodes might take advantage from working against a consensus, our scheme does not require energy-costly mining and, at the same time, does not just cope with multiple heads on the blockchain but even utilizes them to find a consensus. Our consensus-finding algorithm is inspired by swarm behavior commonly found in the animal kingdom.

Our approach isn't just all theory, but along with the publication of this article, we present a working prototype that is implementing the most important aspects described here. Where appropriate, this article will describe further possible enhancements. Our prototype isn't limited to the bare handling of the blockchain but provides a module framework which allows different applications to be run decentralized. One such demo application demonstrates the principle of a decentralized accreditation, and another demo application is a simple voting app to complete our proof-of-concept.*

For a closer look regarding the motivation, refer also to [Roadmap].

Players on the field

In the remainder of this article and within the source code of the prototype implementation, we distinguish between nodes and persons.

(See Figure 1)

A node is an instance of the LiquidFeedback Blockchain client software that runs on a computer connected to the internet (or any other network), while a person is a human who wants to use the LiquidFeedback Blockchain and trusts a particular node.

** This application doesn't incorporate the LiquidFeedback proposition development and decision making process yet but simply allows voting on binary decisions. It is thus only suitable for demonstration purposes.*

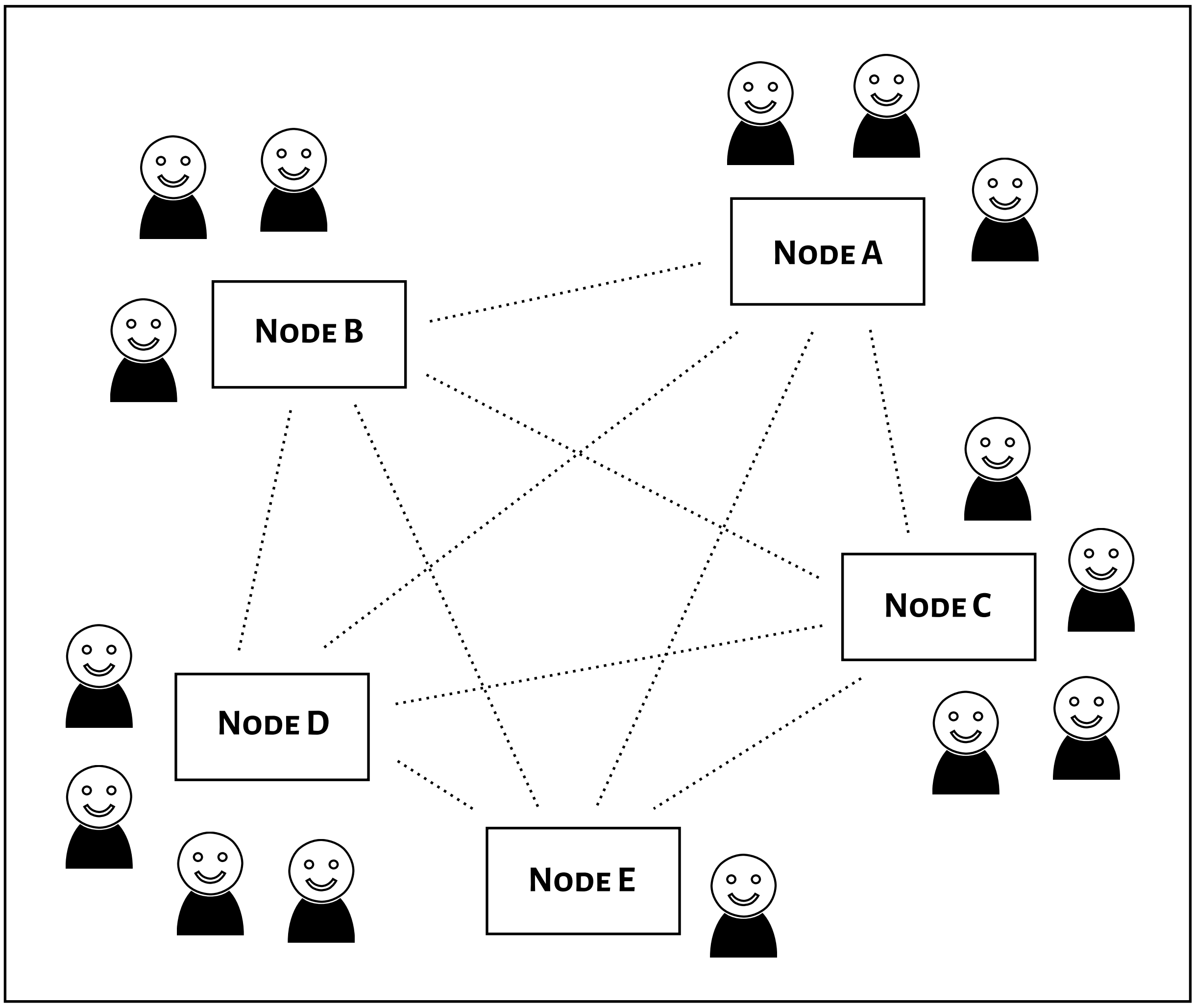


Figure 1: Nodes and persons

Nodes are responsible for engaging in a peer-to-peer communication to exchange new information and agree on a common state. Accredited persons are necessary to empower peers to have their point of view taken into consideration by the other nodes. The more persons trust a node, the higher the influence is of that node within the network. Unlike traditional peer-to-peer systems, the presented prototype supports:

- ***centralized application*** *(a single node used by all persons),*
- *completely* ***decentralized application*** *(each person runs his or her own node),*
- *or* ***any combinations in between****, e.g. some persons have their own node, while other persons share nodes.*

In the current implementation, nodes as well as persons are technically identified through their public key. A node that handles a person's account is currently required to retrieve that person's private key. Other (potentially more secure) schemes of empowering a node are thinkable.

A deterministic blockchain

The main purpose of our blockchain is to find a common understanding which transactions (e.g. cast votes) have been submitted to the system at a certain time or time interval. Each block in the blockchain will represent a quantized amount of time. For periods without transactions, each block contains an integer for storing the time interval such that it is possible to skip a big number of time intervals without enlarging the blockchain.

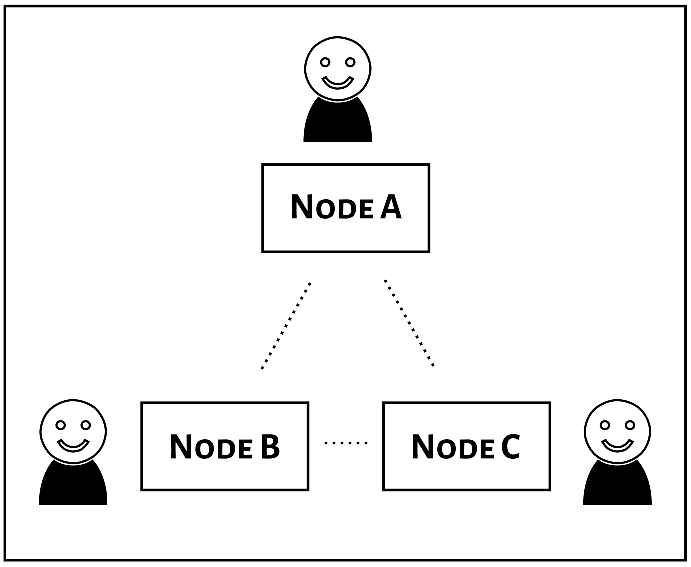


Figure 3: Complete decentralized application

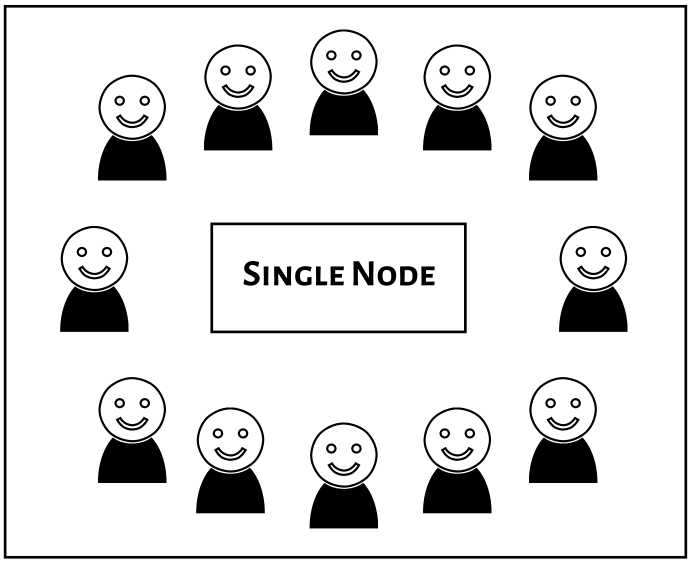


Figure 2: Fully centralized application

As usual, the hash of each block is computed by hashing a concatenation of the hash of the previous block with the payload of the current block in order to ensure integrity over all previous blocks with a single hash value at the end of the chain. Our approach, however, differs from usual blockchains as follows:

- *Each block contains an unordered non-empty set of transactions. This is implemented through a deterministic order (lexicographical sorting of transaction hashes) of all transactions within a block.*

- *The calculation of the hash of the block is deterministic and there is deliberately no nonce used. Thus equal sets of transactions (at the same time interval) create the same hash, as long as the preceding chain is the same.*

```
{
  "index": 3284,
  "interval": 600,
  "previous_hash": "33b55d628a62d086084894dbc43f3a1368ce90227c15f99adb6bde66a88dd0bc",
  "timeindex": 25480150,
  "transactions": {
    "1bd0e00bc2605e193cd939186e10dfcaeae3756dc59331e96642523dc9bd0057": {
      "data": {
        "decision": "yes",
        "identification": "Jon Doe",
        "intent": "accredit",
        "person_id": "10a5d0f2b71fe83db8021c4472532e047fce84d43193fa7a0082637be9f2c506"
      },
      "module": "person",
      "person_id": "10a5d0f2b71fe83db8021c4472532e047fce84d43193fa7a0082637be9f2c506",
      "signature": "kqM/FlUHGfddRwczbzq4xYgQItA2CE/XyKqcGxruU1utsYg18HeQ0R9t2mDrNSRy\n
                   yojZ0zqNiz/xOw2pw+69YXXNkbnrV1SCXfgMs09nSHcWDtWlziffvf4Lje8i6Br/\nI
                   tZGHjm6vKT1v5KNrAxOSaHisku3GP0eJno1YvYccqfWWKzugBI0YpjKCHbpMQL+\noO
                   x571/7SBmYNxqcRJPttN75JoRQK7AdTs82LLLyWwHCJG2+vg0FPVG+aiDiDGsZ\nQ5N
                   7CWeaW1AKPiQ2FfD8H1ACNoClLe4ZNIXp88XCEdwFGOEp0CKK4R93VV7/Fnlh\nRREB
                   Mqgk0c/UisHTJ41aow==\n"
    }
   "33b55d628a62d086084894dbc43f3a1368ce90227c15f99adb6bde66a88dd0bc": {
      "data": {
        "decision": "yes",
        "initiative_id":  "f33ddf23044ed5c0dcbc858015645ae25196a1f1b55397bf2b610b07451
                          b1964",
        "intent": "vote"
      },
      "module": "demo01",
      "person_id": "10a5d0f2b71fe83db8021c4472532e047fce84d43193fa7a0082637be9f2c506",
      "signature": "IdO7BXd95twQwnb/BWthIhEMFD4PEk/doMY8d8e3mZ3bj+B41bT+LJH0DcqljSbU\n
                   gXAn1q3iVpFI19eCVhenT7inDJTMZBgmjY0TOkz0zyh235HBk/9J/Z2GsK7T9ctk\nK
                   Gp9BVDayFJy/Z9diW/VaWl6GoDL40mQ1m27MEd92iFnr4CVc2QCdcNS/ml1ohPT\nBa
                   IswWXP16yVrnQSXjrUpQtCmpkuU1iGs1NASHK5dCiQ7/VAAkYhW4Ur89xaRapb\nwji
                   uFVA/MsFShCkiDZBl0PDzdc9BbK9jGSUL0U0NUpwzP+0PRwumzPZaF8NmxwLA\nwixd
                   XaB8ZFUVZfHrW+nVQw==\n"
    }
  }
}
```

Figure 4: Example of a block in the LiquidFeedback Blockchain

Gossiping

The exchange of data between all nodes is implemented through a so-called "gossip" protocol, where nodes randomly connect to each other and exchange data. Eventually every node will have received all information.

In our case, however, the selection of nodes for exchanging new parts of the blockchain must not be arbitrary. Each node that wants to pull information from other nodes randomly selects a number of accredited persons. The corresponding nodes are contacted for a pull-only exchange of their current merged state of the blockchain. (Merging will be described in the next section.) In order to determine the portion of the blockchain that needs to be retrieved from each node, a binary search is conducted to determine a common trunk, after

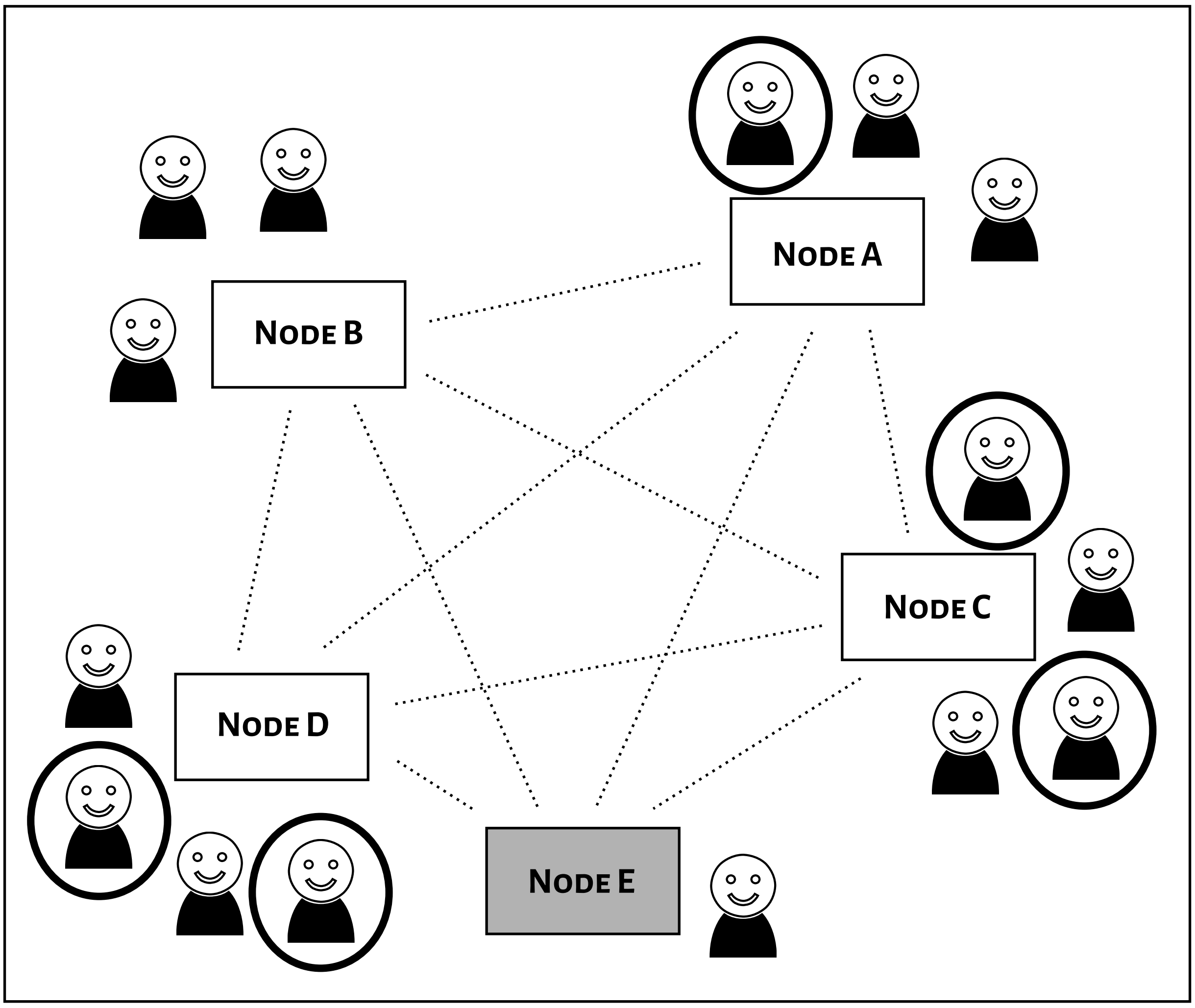


Figure 5: Node E picks random persons

which the remaining blocks are transferred from the corresponding node.

By performing multiple pull-only transfers from a number of nodes (by randomly selecting accredited persons), a node collects a set of different blockchains which can be represented as a tree by combining the common parts. If multiple persons were selected who are associated with the same node, a corresponding weight greater than one is stored for these retrieved blockchains. For authorization, the hash of the head of the exchanged blockchain is signed by the sending peer.

In addition to these pull-only transfers, any two nodes which connect to each other additionally perform a full push and pull exchange of the following data:

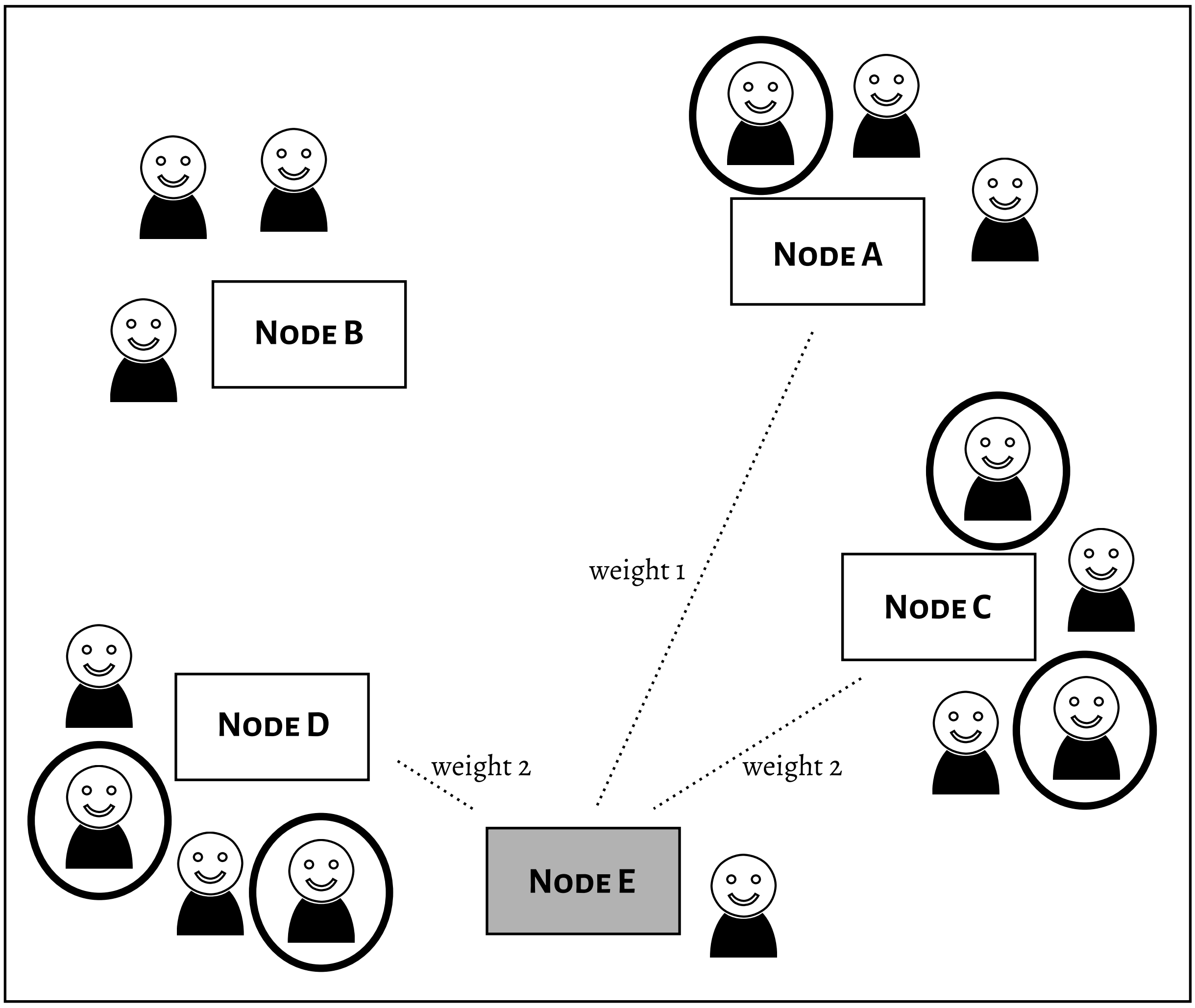


Figure 6: Node E pulls blockchains from other nodes

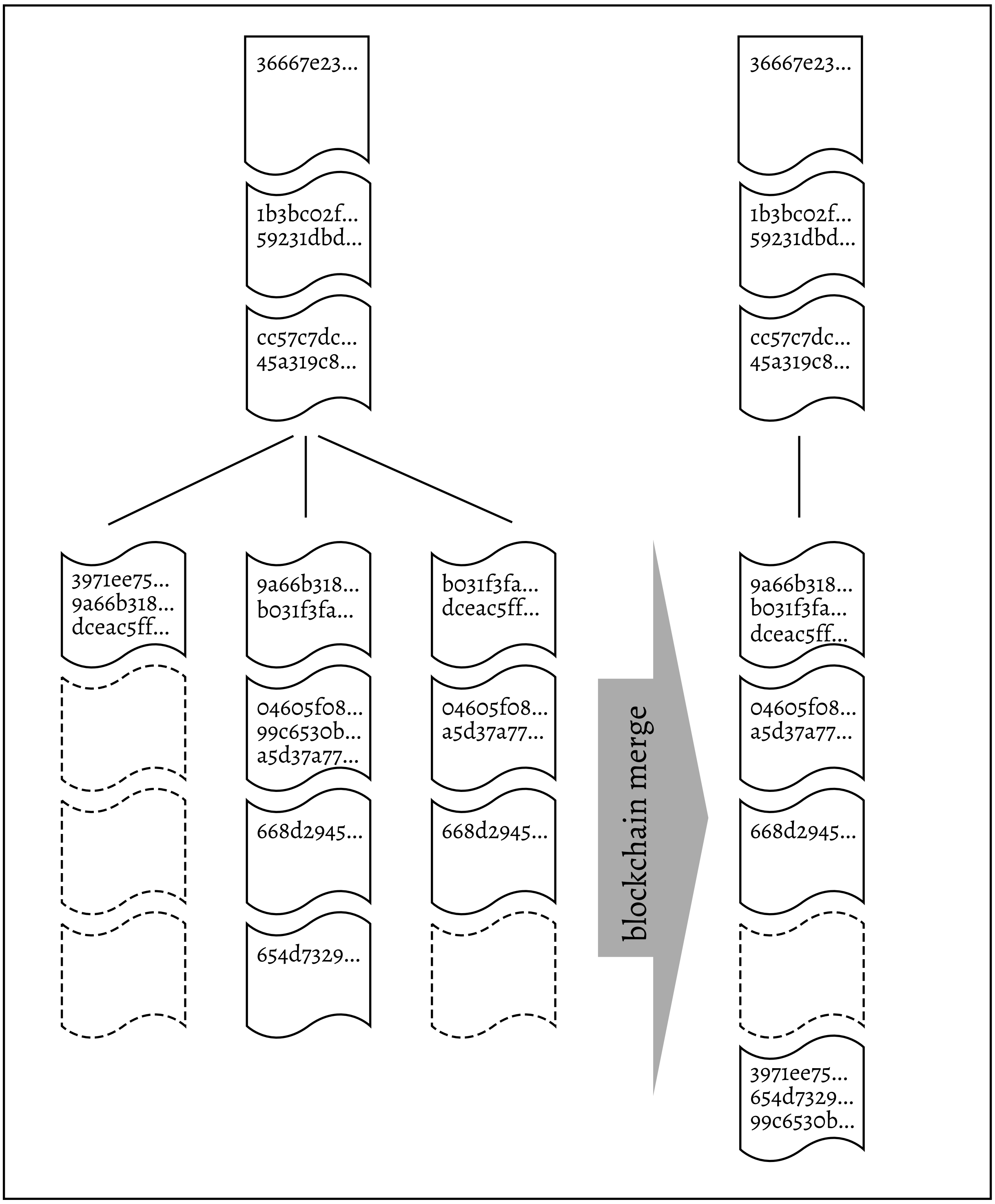


Figure 7: Merging blockchains based on majorities while preserving all transactions

- *host/port announcements from nodes (to inform other nodes how to contact that node),*
- *node trust announcements from persons (i.e. statements which node is the trusted node of a person).*

These additional announcements are secured by a signature of corresponding node or person, respectively, and contain a serial number to allow overriding previous announcements. An announcement with a lower serial is always superseded by an announcement with a higher serial (assuming a valid signature), in which case the older announcement is discarded and does no longer propagate through the network.

Merging

After a configurable number of persons have been randomly selected and their corresponding merged blockchains have been pulled, a merge algorithm is executed. This is where the LiquidFeedback Blockchain differs from other blockchains: alternative branches of the chain aren't cut off (discarded), but due to the deterministic calculation of hashes (that only depend on the sets of transactions per time interval), it is possible to merge a set of branches into a new blockchain that has a single head.

In order to mitigate attacks when a number of nodes is offline (e.g. due to denial-of-service attacks), there needs to be an adaptive "freezing threshold" on how many time intervals may be changed in a node's own blockchain due to merging other blockchain branches. The adaptive threshold will be described later in this article. All other blocks are allowed to change during the merge process.

Blocks for all time intervals after the freezing threshold are reconstructed sequentially. For each time interval after the frozen part, it is determined which remaining transactions (that are not contained in frozen blocks) are contained in a majority of the pulled (yet unmerged) branches at the given time interval (while taking the weight into account that may be greater than one if multiple persons with the same node have been selected during the random selection process).* Those transactions with a majority greater than 50% (considering all pulled branches since the last merge as 100%) are accepted for the current time interval. A block with all accepted transactions is appended and a new hash is calculated.

The process described above is repeated until reaching the current time interval based on the system clock of the node. All transactions which have not been incorporated into the merged blockchain earlier end up in the block for the current time interval.

Swarm behavior

The mechanisms described above mimic a behavior similar to swarms: regarding whether to consider a transaction to have been executed

* *A better approach would be to also take those branches into account where the transaction is contained in the current time interval* or *any previous time interval, but this hasn't been implemented in the prototype yet.*

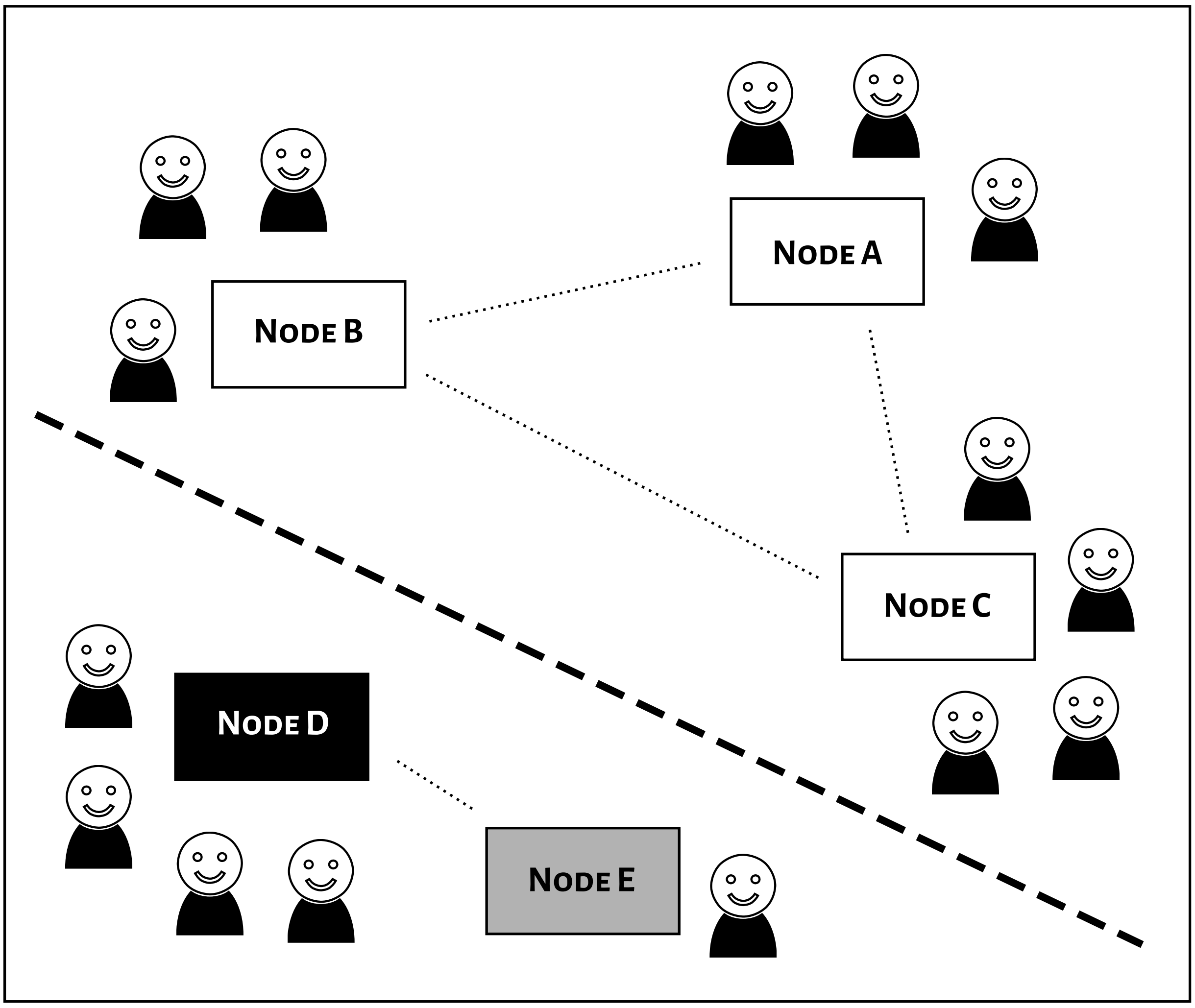


Figure 8: Network split attack step 1
Blocking a majority of nodes (A to C) to convince node E of node D's point of view

during a particular time interval, the node contacts its neighbors (in this case random nodes using a person's trust) and asks for their point of view. Each node will follow a majority point of view of its peers. This leads to an equillibrium where eventually all (or most) nodes will have the same point of view whether a transaction belongs to a particular block or not.

Freezing (and unfreezing) history

In order to mitigate certain kinds of attacks on the network, further security mechanisms have to be implemented. If a majority of nodes is offline, it is easier to maliciously create a wrong majority point of view among the remaining nodes. This could be abused in such way to first

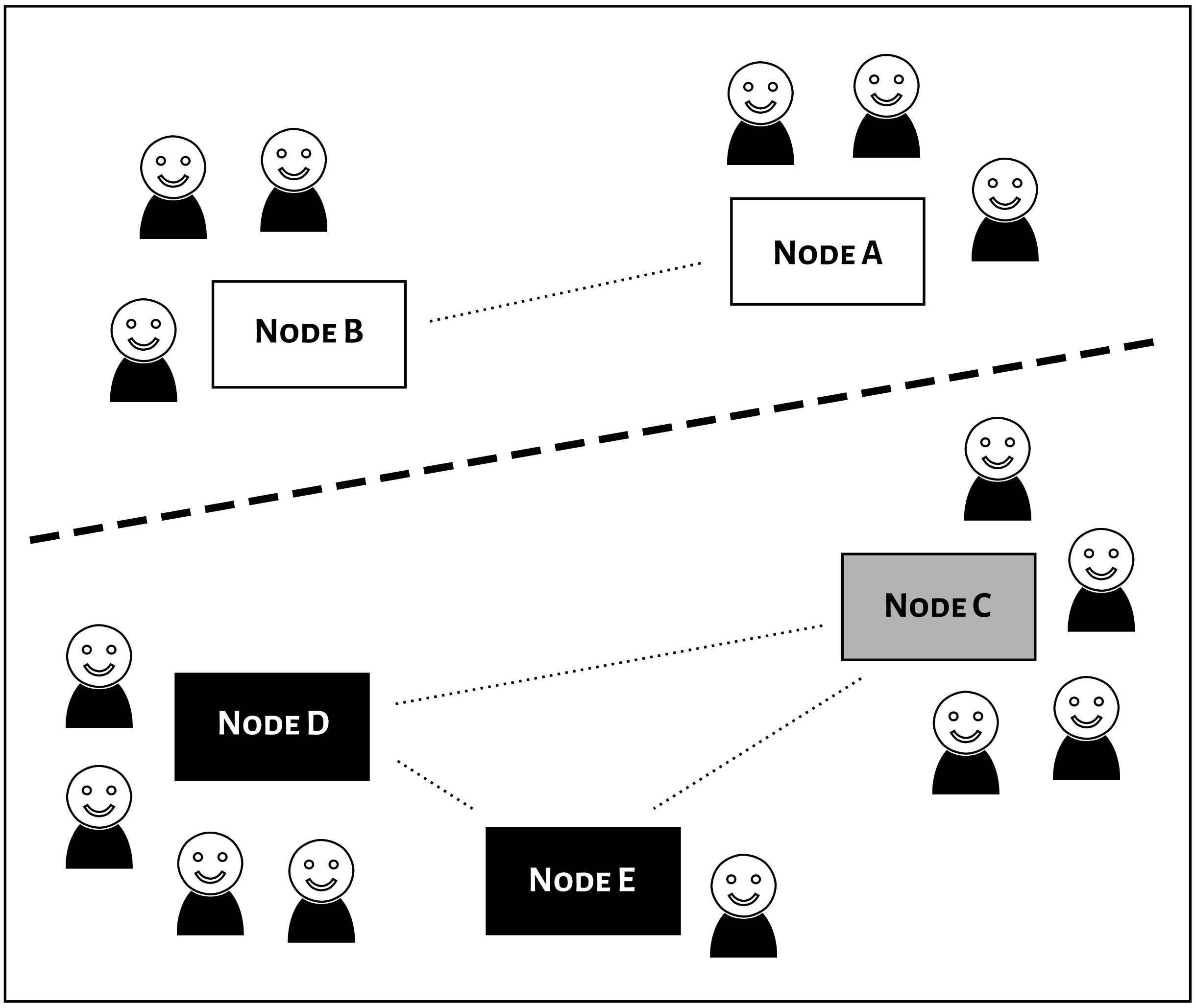


Figure 9: Network split attack step 2
Using majority of node D and E to convince node C (and thus gaining a majority within the whole network)

isolate some nodes (e.g. through a denial-of-service attack), convince them of an alternative state, and then incrementally allow connections of further nodes to these convinced nodes (e.g. by lifting the denial-of-service attack successively). This way, it may be possible to rewrite history without having to control more than half of the nodes (other than blocking them).

To mitigate this attack, changing history should be limited by freezing the past. Each node would have a threshold index which indicates which blocks are allowed to be modified during the merge process. Any blocks with a time index lower than the threshold index must not be modified during merging, and all blocks with higher time indices must not in-

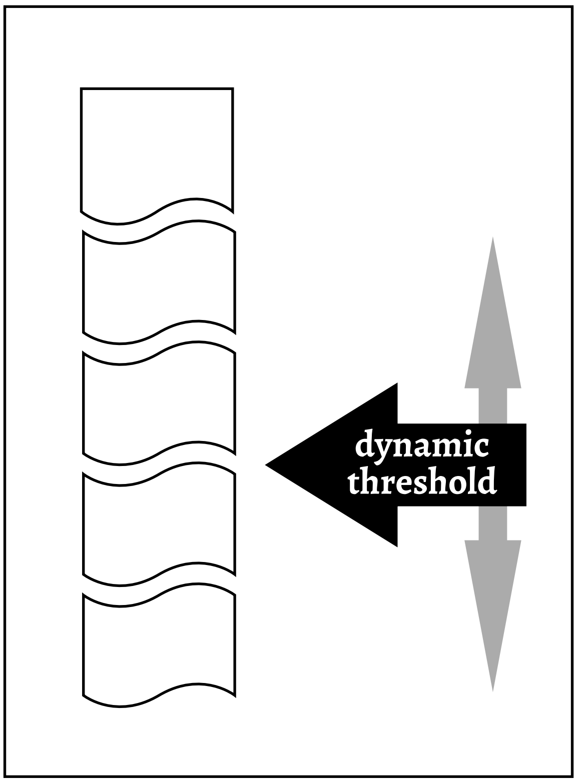


Figure 10: Blocks above the dynamic freezing threshold are not changed during merging

corporate transactions contained already in the frozen part of history during merging. The threshold time index can adaptively change. For example, if during a certain time 'X' the merging algorithm would have changed history if it wasn't frozen, the threshold can be slowly adjusted towards the past, eventually allowing a change of the frozen past. The older the blocks are, the more stable they become. If during the time 'X' no part of the frozen history would have been changed if it wasn't frozen, the threshold can be adjusted towards the future (e.g. twice as fast as moving it to the past otherwise). This way, an equilibrium would be reached that is usually close to the present, and denial-of-service attacks would need to be long-lasting in order to change a larger portion of history.

Flood protection

Another possible attack vector is to flood the system with transactions. To mitigate these kinds of attacks, a quota is necessary that limits each person from injecting more than a certain amount of data per time into the system. Any transactions that would violate this limit could easily be removed from the system during merge.

The application layer

The prototype created along with this article comes with a module framework which allows different applications to be run decentralized. Each application runs in a sandbox and provides hooks to process exchanged transactions and modify an application specific state. The framework assists by rolling back the state accordingly after a merge with history change.

A meta-application is thinkable, which would allow to vote on application code. Applications which receive a majority of votes could be automatically incorporated into the system or updated, respectively.

Adopting the full LiquidFeedback proposition development and decision making process for this framework (as one of many possible applications) is a complex task, which will be commented on in the following section.

Adopting the LiquidFeedback process to the LiquidFeedback Blockchain

Adopting the LiquidFeedback process to the LiquidFeedback Blockchain would require a complete reimplementation even though most algorithms (apart from their implementation) could be used. One of the biggest problems is to temporarily hide cast ballots in order to avoid tactical voting. On one hand, votes must be propagated through the network. On the other hand, they need to be kept secret. [GoD] The most promising solution to this problem is a threshold cryptosystem where a majority of nodes may decrypt all cast ballots, which shall not happen until the voting phase has ended and some additional time has passed for history to become frozen.

Another big obstacle is correctability of the system when votes have been recorded wrongly (e.g. due to a hacked node). The article, "Practical Consequences of Arrow's Theorem for Open Electronic Voting" [Practical], also published in this Issue #6 of The Liquid Democracy Journal, covers this issue in depth and reveals some problems that are inherent to all forms of electronic voting.

Availability of code

A working prototype (without the freezing threshold and flood protection) has been published and is available at the official homepage of "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems" at:
http://www.liquid-democracy-journal.org/

[GoD] Jan Behrens: "Game of Democracy". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 2, October 7, 2014, pp. 11-22. ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/2/The_Liquid_Democracy_Journal-Issue002-02-Game_of_Democracy.html

[PLF] Behrens, Kistner, Nitsche, Swierczek: "The Principles of LiquidFeedback". ISBN 978-3-00-044795-2. Published January 2014 by Interaktive Demokratie e. V., available at http://principles.liquidfeedback.org/

[Practical] Jan Behrens: "Practical Consequences of Arrow's Theorem for Open Electronic Voting". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 6, July 27, 2018, pp. 34-37. ISSN 2198-9532. Published by Interaktive Demokratie e. V.

[Roadmap] Andreas Nitsche: "Roadmap to a decentralized LiquidFeedback". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 6, July 27, 2018, pp. 13-17. ISSN 2198-9532. Published by Interaktive Demokratie e. V.

Decentralized Accreditation

by Jan Behrens, Andreas Nitsche, and Björn Swierczek, Berlin, July 27, 2018

Whenever organizing democratic decisions, a crucial task is the accreditation of eligible voters. The democratic principle “one person – one vote” is vital for the overall fairness of any election or decision making process. Usually the accredition is executed by authorized agents of an organization (in case of decisions within an organization) or by polling clerks (in case of public elections). If a democratic process is properly designed, then the overall process (from accreditation to casting a ballot) can be publicly supervised such that it is not possible to use the accreditation process to violate against the principle of “one person – one vote”. However, even in these cases, a centralized organization (possibly including a voter register or similar database) is usually required.

The previous articles in this Issue #6 of “The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems” explained how a completely decentralized democratic decision making system can be created. For such an electronic system, each eligible voter has the same rights and there is no central authority with certain administrative privileges.

This article will explain how to connect a totally decentralized system with real-world processes that are necessary to perform an accreditation of eligible voters. It should be noted that “one person – one vote” doesn't always imply that every human being gets a vote. Usually an organization wants to empower their members and not people outside of the organization. A municipality wants to empower its citizens or residents. In case of hierarchical organizations, certain decisions might be restricted to people belonging to a certain division of the organization (e.g. a local chapter). But it isn't just a matter of figuring out who should be deemed an eligible voter; we also need to find ways to identify those people.

A simple approach

The simplest approach to collectively perform an accreditation is to conduct a vote on every new person to be added to the set of eligible voters. When we consider a decentralized decision making system such as presented in the previous article, we do not just need to approve the eligible voter but also provide means of authentication of that voter (i.e. after successful accreditation, the system needs to be able to validate whether some input has been really sent by or on behalf of that voter).

Technically, a validation of a person's request could be done using an asymmetric cryptographic key. In a world where every person has appropriate end-user devices that are trusted, this would be the most natural solution. In this case, a person's public key fingerprint would need to be made public, e.g. by reading it in front of an audience at a public assembly. Then everyone would be entitled to submit a request to the system that the owner of the corresponding private key is to be added to the list of eligible voters.

Joining resources

The idea that every person can have a public/private-key pair and that they are technically able to keep their private key secret is an idealization (see also [PLF, section 3.5]). The idea might have been practicable during the 90s, but quickly end-user devices were increasingly becoming interconnected in such a way that regular software updates are common. Only a small fraction of internet users has real control over their devices; most users will regularly install many megabytes (or even gigabytes) of unknown machine code that has been provided to them from companies like Microsoft, Apple, or Google.

Consequently, storing a private key on an end-user device isn't necessarily the best choice these days. While aiming for decentralization, a technological monoculture (e.g. if a majority of participants uses a certain operating system or a certain chipset) could introduce a single point of failure to the system.

A different approach for identifying people would be to entrust a local chapter of an organization (or any other known group of people) with the particular job of maintaining a person's account (and to act as proxy for that person). Such an entity could use united resources to monitor security issues in a better way than the average person might do when being on their own. Each person could freely choose the entity he or she entrusts with managing his or her account (or manage the account themselves, if desired). This way, single points of failure might be more easily circumvented. Combining this with a public accreditation process, a person would state his or her identity and publicly announce which technological provider will be his or her proxy.

At the absolute discretion of the voters?

If, from a technical point of view, all eligible voters decide about including a new member in a democratic decision, one may wonder if that constrains the statutes of an organization? Would the approval of each new member application depend on the absolute discretion of a majority of members?

While the approval of a new entrant is technically implemented through a majority decision, this doesn't impose constraints on the statutes of an organization using such a decentralized accreditation system. The semantics of the technical voting procedure on approving a new member can be defined in such a way that the voters do not state whether they *want* the new member to be accepted but whether they confirm or do not confirm that the member fulfills the requirements necessary to be approved. It could be each member's obligation to truthfully answer this question. Of course, if a majority of members fail to act according to the rules, the overall system fails. As previously discussed in the article “Roadmap to a Decentralized LiquidFeedback” [Roadmap], it is a reasonable (and necessary) assumption that a majority of participants behave benevolently.

Of course, the use of transitive delegations (Liquid Democracy) could keep the additional overhead for most participants as small as possible. Members of a political party, for example, could simply delegate the job of accreditation to someone they trust (or to someone who they trust to know someone trustworthy).

Difference from web of trust

The proposed system is different from a classical web of trust. While a web of trust (such as used with PGP, GnuPG, or similar cryptographic software) is also capable of verifying one's true identity (and to connect it with a public/private-key, for example), a classical web of trust isn't capable of creating a consensus because the trust level of an entity depends on the point of view from which you look at the network (i.e. each participant concludes different trust levels).

The key feature of a distributed decision making system, however, *is* to find a consensus[*]. A web of trust could be modified in such way that there is a “seed” of trusted members, i.e. the trust wouldn't depend on the point of view but would originate from those members. Treating certain members different from other members, however, doesn't fulfill the equal treatment of all participants (and thus conflicts with our goals for a decentralized accreditation system). Therefore, the “seed of trust” needs to be dynamic and ultimately reflect all members equally.[†]

Removing accounts

It is not sufficient to provide mechanisms for new members, but there also need to be mechanisms for removing the accreditation of a person. Even in cases where there is no expulsion of members allowed, it is always possible for a human to die. In this case, his or her accreditation would need to be removed to avoid misuse of his or her voting power. The previous statements regarding new entrants also apply to removing member accounts. The

[*] *With “consensus” we do not mean unanimous decisions but a consensus based on majorities. See also [PLF, section 4.13].*

[†] *It would still be possible to temporarily restrict new members from deciding about accepting other members, but this should only be a temporary condition.*

mechanism for removing accounts could also serve as a last resort for healing identity theft such as loss of a private key.

Summary

Decentralized accreditation is similar to the concept of "web of trust" but differs from it in certain regards. Most importantly, decentralized accreditation aims to create a common point of view on a set of eligible voters and their means of authorization.

A simple approach to decentralized accreditation is to conduct votes for each new person to join a group of eligible voters. While from a technical point of view, this empowers the electorate to decide about each accredited person, the elecorate could be bound to rules when executing this task. These rules must be well-defined and take into account removal of member accounts as well.

While it is possible to use public/private-key pairs for authorization, it is not required that each eligible voter securely stores a private key him- or herself. Technical providers can handle identity management and authorization of several voters at the same time as long as each eligible voter is able to freely choose such technical providers and/or use their own technical infrastructure.

[PLF] Behrens, Kistner, Nitsche, Swierczek: "The Principles of LiquidFeedback". ISBN 978-3-00-044795-2. Published January 2014 by Interaktive Demokratie e. V., available at http://principles.liquidfeedback.org/

[Roadmap] Andreas Nitsche: "Roadmap to a decentralized LiquidFeedback". In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 6, July 27, 2018, pp. 13-17. ISSN 2198-9532. Published by Interaktive Demokratie e. V.

Practical Consequences of Arrow's Theorem for Open Electronic Voting

by Jan Behrens, Berlin, July 27, 2018

In the second article [Roadmap] in this Issue #6 of The Liquid Democracy Journal, we mentioned two contradictory design criteria for electronic decision making systems: on the one hand, considerations regarding tactical voting require that cast ballots are temporarily hidden and that ballots should not be changed or removed after disclosore of the cast ballots; on the other hand, we want to ensure that wrongly cast ballots (e.g. through hacker attacks) can be corrected afterwards.

This article will elaborate a deeper conflict that can be seen as a consequence of Arrow's Impossibility Theorem in combination with complexity of computer systems and which affects all forms of electronic decision making.

Arrow's impossibility theorem

Kenneth Arrow showed in 1951 that it is impossible to create a voting system where all voters may express their preferences while fulfilling several desirable properties of the system at the same time. In consequence, tactical voting cannot be ruled out when there are more than two voting options. [Gibbard] [PLF, section 4.14]

One might be tempted to restrict people from expressing their true preferences on a ballot (or to allow voters to express more than their preferences, e.g. a “satisfaction rate” that is a real number between 0 and 10). But these attempts are not suitable to eliminate the possibility of tactical voting either. [PLF, section 4.14]

Given more than two voting options, we have to face the possibility of tactical voting. Thus, preferential ballots should be hidden between submitting and tallying, such that individual voters cannot gain advantages by delaying their own vote and casting a strategically modified ballot based on other ballots that have been already submitted by other voters. [GoD]

It should be noted that the above considerations only apply when there are more than two voting options. A binary decision (that is totally independent from other decisions) is not subject to tactical considerations because it makes always sense to vote for your favorite option. The next section will explain why democratic decisions cannot generally be broken down to a sequence of binary decisions without influencing the outcome of the vote.

Majority cycles and binary decisions

The general existence of collective majority cycles has already been discovered in the 18th century by Marquis de Condorcet. [Condorcet] [PLF, p.151] An example for a collective majority cycle is that there is a majority preferring 'B' over 'C', another majority preferring 'A' over 'B', and yet another majority preferring 'C' over 'A'. Research in the last century revealed that under certain preconditions, there is a high probability that all voting options are tied in such a Condorcet paradox. [Schofield] In consequence, breaking down a complex decision into a sequence of yes/no-questions has a major influence on the overall outcome depending on the order in which the questions are being asked. [GoD] Having a committee determining the order of the questions would give power to the committee to decide about the outcome of the vote in certain scenarios and thus violate the democratic goal to treat everyone equal in the decision making process.*

Even if this problem of losing fairness is disregarded, breaking a decision down into a sequence of yes/no-questions doesn't solve the problem of tactical voting either. Consider an example where a voter prefers 'A' over 'B' over 'C'. The first ballot is about eliminating 'B' or 'C'. Naturally, you would expect the voter to vote for 'B' (i.e. for eliminating 'C'). But considering that 'C' is an option almost nobody favors, it might be smarter to eliminate 'B' because in the subsequent ballot between 'A' and 'C', the first preference of the voter (i.e. 'A') might win. Any attempt to convert political decisions into a series of binary questions will, in case of majority cycles, not just introduce unfair side effects but also cannot solve the problem of tactical voting.

Complexity of technology

Another circumstance we have to consider when designing or assessing electronic decision making systems is complexity of technology and its impact on verifiability, particularly in those cases where cryptography or remote authentication and authorization methods are used. Computer systems are too complex to be verified by their users. In consequence, hacking attempts are possible that might stay undetected or at least undetected until a ballot has been closed.

Using open ballots allows for a verification of the overall process by the participants. [PLF, chapter 3] However, depending on the impact of the decisions made with an electronic sys-

* *Also when randomizing the order, not all voting options could be treated equally. Additionally such a system could be vulnerable to clones (see also [Tideman]).*

tem, we might not just want verifiability but also the ability to recover from a hacking attempt and to correct wrongly cast ballots (which would be possible given an open ballot where identities are disclosed). But if it was possible to correct a ballot after tallying, tactical advantages could be gained by claiming to be hacked and "correcting" one's ballot in such a way that tactical considerations regarding all other cast ballots are taken into account. In the end, those people who correct their votes last would gain an unfair advantage.

An unsolvable conflict

Obviously, a correction of ballots cannot be forbidden and made possible at the same time. There seems to be an unsolvable conflict: either we declare all ballots as immutable once they have been tallied by an electronic voting system, or we allow corrections by each person who cast a vote (which is possible in case of recorded votes). In the latter case, people who fake a manipulation (e.g. by claiming their system was hacked and/or posting their credentials) would gain a tactical advantage.

Work around

In many contexts (e.g. policy making), it would be irresponsible to abandon the possibility of correcting manipulations because this could put hackers in a dangerous position of power. We thus have to deal with corrections but try to keep the impact on the overall fairness of the decision making process as little as possible. One solution could be to proceed with vote corrections as follows:

If the correction of ballots does not change the overall outcome of the vote, these corrections can simply be performed and the result doesn't change. If, however, the overall outcome would be different with the corrected ballots, then we distinguish between two cases.

Case I: the decision was a binary decision. In this case, the ballots get corrected and the overall result is changed accordingly.

Case II: the decision involved more than two voting options. In this other case, the outcome of the vote is declared void and the vote must be repeated. Declaring the vote void instead of allowing another voting option to win reduces the potential impact by faking a manipulation in order to gain tactical advantages to maintaining the status quo (i.e. a decision can only be declared void but not changed). Furthermore, voters who make use of their possibility to correct votes could be treated differently in future elections: their votes could be published prematurely and they could be given a certain time to have them corrected until the remaining votes are made public and tallied.

It should be noted that this approach would treat the affected voters in Case II in an unfair fashion if they were true victims of a hacking attempt (or victim of an administrator's privilege abuse in case of a central server system where the administrator could manipulate votes easily). However, reducing the impact to disadvantages in tactical voting (by letting certain voters vote prematurely and publish their votes) is a lesser problem than giving hackers the power to completely control a voter's ballot.

Future work

It has been shown that the impossibility to verify the correct behavior of electronic decision making systems has an effect on the overall fairness of a compromised system even in cases where all votes are done by roll call (recorded identities for each ballot, which get published along with the ballot). A proposal has been made how to practically deal with later corrections of ballots while trying to preserve the equal treatment of all voters as good as possible. Not all impacts have been fully analyzed. For example, a premature publication of certain ballots would allow estimations on the final outcome of the voting procedure and might foster tactical voting of the other voters. For a decentralized decision making system based on the LiquidFeedback Blockchain [LFB], a precise algorithm for publication order and timings would need to be created.

The considerations in this article are a rough analysis of a generic problem that affects all forms of electronic decision making. In order to better assess the consequences and potential solutions or workarounds, it would be helpful to formalize the problem and provide a mathematical proof for (a formalized variant of) the previous statements. We believe this undertaking to be non-trivial because it would, for example, require a modeling of real-world processes such as “publication” of certain data or a “public objection” to one's ballot.

[Condorcet] Marquis de Condorcet: “Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix”. Imprimerie Royale, Paris, 1785.

[Gibbard] Allan Gibbard: “Manipulation of Voting Schemes: A General Result”. In “Econometrica”, Vol. 41, No. 4, July 1973, pp. 587–601. Published by the Econometric Society (Wiley-Blackwell).

[GoD] Jan Behrens: “Game of Democracy”. In “The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems”, Issue 2, October 7, 2014, pp. 11-22. ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/2/The_Liquid_Democracy_Journal-Issue002-02-Game_of_Democracy.html

[LFB] Jan Behrens, Axel Kistner, Andreas Nitsche, Björn Swierczek: “The LiquidFeedback Blockchain”. In “The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems”, Issue 6, July 27, 2018, pp. 18-29. ISSN 2198-9532. Published by Interaktive Demokratie e. V.

[PLF] Behrens, Kistner, Nitsche, Swierczek: “The Principles of LiquidFeedback”. ISBN 978-3-00-044795-2. Published January 2014 by Interaktive Demokratie e. V., available at http://principles.liquidfeedback.org/

[Roadmap] Andreas Nitsche: “Roadmap to a decentralized LiquidFeedback”. In “The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems”, Issue 6, July 27, 2018, pp. 13-17. ISSN 2198-9532. Published by Interaktive Demokratie e. V.

[Schofield] Norman Schofield, Bernard Grofman, Scott L. Feld: “The Core and the Stability of Group Choice in Spatial Voting Games”. In the “American Political Science Review”, Vol. 82, No. 1, March 1988, pp. 195–211. Published by Americian Political Science Association (Cambridge University Press).

[Tideman] Nicolaus Tideman: “Independence of clones as a criterion for voting rules”. In “Social Choice and Welfare”, Vol. 4, Issue 3, 1987, pp. 185-206. Published by Springer.

A mathematical view on the sockpuppet problem

Jan Behrens

April 12, 2016

Abstract

We will consider two attempts to curtail the risk of sockpuppets by reducing the voting-weight of non-verified accounts to a small (but non-zero) value. It can be proven that at least in those cases where the non-verified accounts have an impact on the outcome of the vote, a malicious attacker can overrule *any* binary decision by simply adding a finite number of sockpuppets, even if the voting-weight of each non-verified account is reduced to a small but non-zero value or if the total voting-weight of all non-verified accounts is limited.

Disclaimer

Please note that allowing sockpuppets in any vote or in any decision-making system also violates the democratic principle of "one man – one vote". Therefore, the findings of this paper are only a supplementary reason for proper accreditation in democratic processes.

Conventions

- The symbol for the natural numbers $\mathbb{N}$ shall refer to a set that includes the number zero, i.e. $\mathbb{N} := \{0, 1, 2, 3, \ldots\}$.
- The symbol for rounding a number $r \in \mathbb{R}$ (or $r \in \mathbb{Q}$) down to the next integer value is $\lfloor r \rfloor$ such that $r \geq \lfloor r \rfloor \in \mathbb{Z}$ and $r - \lfloor r \rfloor < 1$.
- The symbol for rounding a number $r \in \mathbb{R}$ (or $r \in \mathbb{Q}$) up to the next integer value is $\lceil r \rceil$ such that $r \leq \lceil r \rceil \in \mathbb{Z}$ and $\lceil r \rceil - r < 1$.
- $\min(a, b) := a \text{ if } a \leq b \text{ else } b$
- $\max(a, b) := a \text{ if } a \geq b \text{ else } b$
- The term "sockpuppet" describes a fake identity created with malicious intent to manipulate public opinion and/or decisions by deception.
- Mr. Evil or Ms. Evil are used as names to refer to an attacker whose goal is to manipulate a decision by overruling the overall outcome of a vote using sockpuppets.

1 Reducing the voting-weight of non-verified accounts by a factor $x < 1$

We consider a voting system where internet users may create "non-verified" accounts on their own behalf (e.g. by using anonymous e-mail addresses or social-media accounts). These accounts may be legit or malicious (i.e. sockpuppets). A legit account is an account which is operated by a person who doesn't operate any other account in the system. In either case, each non-verified account gets a reduced voting-weight of $x < 1$, while users who verify their account through an accreditation process (with verification of identity) get a voting-weight of 1.

1.1 Conventions

P : The number of verified accounts (of which each has a voting-weight of 1)

x : Voting-weight of each non-verified account

n : The number of sockpuppets controlled by Mr. Evil

S : The number of all other (possibly legit) non-verified accounts

Consequently, the total voting-weight of all verified accounts is equal to P, whereas nx is the voting-weight of Mr. Evil, and Sx is the total voting-weight of all other non-verified accounts. Therefore, $P + Sx + nx$ is the total voting-weight of all accounts in the system.

1.2 Proposition 1

Let:

$$x > 0,\ P \in \mathbb{N},\ S \in \mathbb{N} \tag{1}$$

Then there exists an $n \in \mathbb{N}$ such that:

$$nx > \frac{P + Sx + nx}{2} \tag{2}$$

This means: For any (possibly very tiny) positive voting-weight $x > 0$ of each non-verified account and arbitrary counts of verified accounts P and (possibly legit) non-verified accounts S, there exists a number n of additional sockpuppets which Mr. Evil can add to gain a voting-weight of more than 50%, i.e. an absolute majority.

1.3 Proof of proposition 1

Choose n as follows:

$$n := \left\lfloor \frac{P}{x} \right\rfloor + S + 1 \tag{3}$$

$$= \left\lfloor \frac{P}{x} + S \right\rfloor + 1 \tag{4}$$

3

Note that $n \in \mathbb{N}$ because $\lfloor \frac{P}{x} + S \rfloor \in \mathbb{N}$. We further define $n' \in \mathbb{R}$:

$$n' := \frac{P}{x} + S \tag{5}$$

Then, because $x > 0$:

$$\begin{aligned}
& n = \lfloor n' \rfloor + 1 && (6)\\
\Rightarrow \quad & n > n' && (7)\\
\Leftrightarrow \quad & n > \frac{P}{x} + S && (8)\\
\Leftrightarrow \quad & n > \frac{P + Sx}{x} && (9)\\
\Leftrightarrow \quad & nx > P + Sx && (10)\\
\Leftrightarrow \quad & \frac{nx}{2} > \frac{P + Sx}{2} && (11)\\
\Leftrightarrow \quad & nx > \frac{P + Sx}{2} + \frac{nx}{2} && (12)\\
\Leftrightarrow \quad & nx > \frac{P + Sx + nx}{2} && (13)
\end{aligned}$$

Since $n \in \mathbb{N}$ and because (2) is identical to (13), proposition 1 is true. □

In other words: If Mr. Evil creates $n = \lfloor \frac{P}{x} \rfloor + S + 1$ sockpuppets, then he has more voting-weight than the other (verified and non-verified) accounts combined, i.e. he obtains an absolute majority by fraud, which empowers him to override the outcome of the vote.

4

2 Limiting the total voting-weight of all non-verified accounts to a constant value $T_{\max}$

In the following, we will consider another attempt to stop Mr. (or Ms.) Evil from overruling majorities. We limit the *total* voting-weight of all non-verified accounts by reducing the voting-weight of each non-verified account proportionally as more non-verified accounts are created. Therefore, the total voting-weight of all non-verified accounts stays constant if $n \to \infty$.

As shown in this section, this attempt is also futile because whenever the non-verified accounts have an impact on the decision, a finite number of sockpuppets grants complete control over the outcome of the decision.

2.1 Conventions

P_{yes} :	The number of verified accounts voting for "Yes"
P_{no} :	The number of verified accounts voting for "No"
S_{yes} :	The number of (possibly legit) non-verified accounts voting for "Yes", disregarding Ms. Evil's sockpuppets
S_{no} :	The number of (possibly legit) non-verified accounts voting for "No", disregarding Ms. Evil's sockpuppets
n :	The number of additional sockpuppets controlled by Ms. Evil
$T_{\max}$:	Maximum total voting-weight of all non-verified accounts
T' :	Total voting-weight of all non-verified accounts if Ms. Evil would not use her sockpuppets
T :	Total voting-weight of all non-verified accounts if Ms. Evil manipulates the vote with her sockpuppets

Note that the voting options "yes" and "no" are chosen without loss of generality, i.e. "yes" / "no" could be replaced with "no" / "yes", "proposal A" / "proposal B", "status quo" / "amendment C", etc. Further note that the number of all non-verified accounts is $S_{\text{yes}}+S_{\text{no}}$ without Ms. Evil's sockpuppets and $S_{\text{yes}}+S_{\text{no}}+n$ with those sockpuppets.

2.2 Premises

Let:

$$T_{\max} > 0, \tag{14}$$
$$P_{\text{yes}} \in \mathbb{N},\ P_{\text{no}} \in \mathbb{N}, \tag{15}$$
$$S_{\text{yes}} \in \mathbb{N},\ S_{\text{no}} \in \mathbb{N} \tag{16}$$

We further premise that the non-verified accounts without Ms. Evil's sockpuppets have an actual impact on the overall outcome of the vote. Keeping in mind that "yes" and "no" were chosen without loss of generality, we do this by assuming:

$$P_{\text{yes}} > P_{\text{no}} \tag{17}$$
$$P_{\text{yes}} + T' \cdot \frac{S_{\text{yes}}}{S_{\text{yes}} + S_{\text{no}}} < P_{\text{no}} + T' \cdot \frac{S_{\text{no}}}{S_{\text{yes}} + S_{\text{no}}} \tag{18}$$

with

$$T' := \min(T_{\max}\,,\ S_{\text{yes}} + S_{\text{no}}) \tag{19}$$

being the total voting-weight of the non-verified accounts (excluding Ms. Evil's sockpuppets), and

$$S_{\text{yes}} + S_{\text{no}} > 0. \tag{20}$$

T' is defined in such way that it is $\leq S_{\text{yes}} + S_{\text{no}}$ (ensuring that each non-verified account gets a voting-weight of at most 1) but also limited by $T_{\max}$ (i.e. the total voting-weight of those accounts doesn't exceed $T_{\max}$). T' is then split up equally among all non-verified accounts (see inequality 18).

2.3 Proposition 2

Given the premises stated in the previous subsection 2.2, there exists an $n \in \mathbb{N}$ such that

$$P_{\text{yes}} + T \cdot \frac{S_{\text{yes}} + n}{S_{\text{yes}} + S_{\text{no}} + n} > P_{\text{no}} + T \cdot \frac{S_{\text{no}}}{S_{\text{yes}} + S_{\text{no}} + n} \tag{21}$$

with

$$T := \min(T_{\max}\,,\ S_{\text{yes}} + S_{\text{no}} + n) \tag{22}$$

being the effective total voting-weight of all non-verified accounts including Ms .Evil's sockpuppets.

6

This means: If we limit the total voting-weight T (or T' respectively) of all non-verified accounts to a constant but non-zero value $T_{\max}$ (inequality 14 with definitions 19 and 22), and split it up equally among all non-verified accounts, then, for any binary yes/no decision, there exists a number of additional sock-puppets n which Ms. Evil can add to overrule that decision (inequalities 18 and 21) if the other non-verified accounts $S_{\text{yes}} + S_{\text{no}} > 0$ had an impact on the overall outcome of the vote (inequalities 17 and 18).

2.4 Proof of proposition 2

From inequality (20) and $n \in \mathbb{N}$, we know that:

$$S_{\text{yes}} + S_{\text{no}} + n > 0 \tag{23}$$

We choose $n \in \mathbb{N}$ as follows:

$$n := S_{\text{no}} + 1 \tag{24}$$

Then inequality (21) can be transformed as follows:

$$\begin{aligned}
& P_{\text{yes}} + T \cdot \frac{S_{\text{yes}} + n}{S_{\text{yes}} + S_{\text{no}} + n} > P_{\text{no}} + T \cdot \frac{S_{\text{no}}}{S_{\text{yes}} + S_{\text{no}} + n} \\
\Leftrightarrow\ & P_{\text{yes}} + T \cdot \frac{S_{\text{yes}} + S_{\text{no}} + 1}{S_{\text{yes}} + S_{\text{no}} + n} > P_{\text{no}} + T \cdot \frac{S_{\text{no}}}{S_{\text{yes}} + S_{\text{no}} + n} && (25) \\
\Leftrightarrow\ & P_{\text{yes}} + T \cdot \frac{S_{\text{yes}} + 1}{S_{\text{yes}} + S_{\text{no}} + n} > P_{\text{no}} && (26)
\end{aligned}$$

Definition (22) with inequalities (14) and (23) implies that $T > 0$. Because (17) demands that $P_{\text{yes}} > P_{\text{no}}$, and (16) implies that $S_{\text{yes}} \geq 0$, we can easily see that inequality (26) is true. Therefore (21) is true. □

As shown in the following subsections, it is possible to provide another definition for n, which yields to an even smaller number of sockpuppets in many cases.

2.5 Proposition 3

The following alternative definition of n also fulfills inequality (21) with the given definition of T in (22) and the given premises in (14) through (20):

$$n := \max(\lfloor n' \rfloor + 1\,,\ \lceil T_{\max} \rceil) \tag{27}$$

with

$$n' := S_{\text{no}} \cdot \frac{1 - \dfrac{P_{\text{yes}} - P_{\text{no}}}{T_{\max}}}{1 + \dfrac{P_{\text{yes}} - P_{\text{no}}}{T_{\max}}} - S_{\text{yes}} \tag{28}$$

Note that n' is well-defined, because inequality (17) requires that $P_{\text{yes}} > P_{\text{no}}$ and (14) states that $T_{\max} > 0$.

2.6 Proof of proposition 3

From inequality (20) and $n \in \mathbb{N}$, we know that:

$$S_{\text{yes}} + S_{\text{no}} + n > 0 \tag{29}$$

Definition (22) with inequalities (14) and (29) implies that $T > 0$. Knowing $T > 0$, we use inequality (17) for the following estimation:

$$\begin{aligned}
& P_{\text{yes}} > P_{\text{no}} \\
\Leftrightarrow\quad & P_{\text{yes}} - P_{\text{no}} > 0 && (30)\\
\Leftrightarrow\quad & \frac{P_{\text{yes}} - P_{\text{no}}}{T} > 0 && (31)\\
\Rightarrow\quad & 1 + \frac{P_{\text{yes}} - P_{\text{no}}}{T} > 0 && (32)
\end{aligned}$$

Definition (27) implies $n \geq T_{\max}$. Furthermore, it is presumed in (20) that $S_{\text{yes}} + S_{\text{no}} > 0$. Therefore:

$$\begin{aligned}
& T_{\max} \leq n && (33)\\
\Rightarrow\quad & T_{\max} \leq S_{\text{yes}} + S_{\text{no}} + n && (34)
\end{aligned}$$

From (22) and (34), it follows that:

$$T = T_{\max} \tag{35}$$

8

Definition (27) implies that $n > n'$. Using the definition of n' in (28), we reason:

$$n > \underbrace{S_{\text{no}} \cdot \frac{1 - \dfrac{P_{\text{yes}} - P_{\text{no}}}{T_{\max}}}{1 + \dfrac{P_{\text{yes}} - P_{\text{no}}}{T_{\max}}} - S_{\text{yes}}}_{n'} \tag{36}$$

$$\overset{(35)}{\Leftrightarrow} \quad n > S_{\text{no}} \cdot \frac{1 - \dfrac{P_{\text{yes}} - P_{\text{no}}}{T}}{1 + \dfrac{P_{\text{yes}} - P_{\text{no}}}{T}} - S_{\text{yes}} \tag{37}$$

$$\overset{(32)}{\Leftrightarrow} \quad n \cdot \left(1 + \frac{P_{\text{yes}} - P_{\text{no}}}{T}\right) > S_{\text{no}} \left(1 - \frac{P_{\text{yes}} - P_{\text{no}}}{T}\right) - S_{\text{yes}} \left(1 + \frac{P_{\text{yes}} - P_{\text{no}}}{T}\right) \tag{38}$$

$$\Leftrightarrow \quad n \cdot \left(1 + \frac{P_{\text{yes}} - P_{\text{no}}}{T}\right) > S_{\text{no}} - S_{\text{yes}} - \frac{P_{\text{yes}} - P_{\text{no}}}{T} \cdot (S_{\text{yes}} + S_{\text{no}}) \tag{39}$$

$$\Leftrightarrow \quad n + \frac{P_{\text{yes}} - P_{\text{no}}}{T} \cdot n > S_{\text{no}} - S_{\text{yes}} - \frac{P_{\text{yes}} - P_{\text{no}}}{T} \cdot (S_{\text{yes}} + S_{\text{no}}) \tag{40}$$

$$\Leftrightarrow \quad \frac{P_{\text{yes}} - P_{\text{no}}}{T} \cdot n > S_{\text{no}} - S_{\text{yes}} - \frac{P_{\text{yes}} - P_{\text{no}}}{T} \cdot (S_{\text{yes}} + S_{\text{no}}) - n \tag{41}$$

$$\Leftrightarrow \quad \frac{P_{\text{yes}} - P_{\text{no}}}{T} \cdot (S_{\text{yes}} + S_{\text{no}}) + \frac{P_{\text{yes}} - P_{\text{no}}}{T} \cdot n > S_{\text{no}} - S_{\text{yes}} - n \tag{42}$$

$$\Leftrightarrow \quad \frac{P_{\text{yes}} - P_{\text{no}}}{T} \cdot (S_{\text{yes}} + S_{\text{no}} + n) > S_{\text{no}} - (S_{\text{yes}} + n) \tag{43}$$

$$\overset{(29)}{\Leftrightarrow} \quad \frac{P_{\text{yes}} - P_{\text{no}}}{T} > \frac{S_{\text{no}} - (S_{\text{yes}} + n)}{S_{\text{yes}} + S_{\text{no}} + n} \tag{44}$$

$$\overset{T>0}{\Leftrightarrow} \quad P_{\text{yes}} - P_{\text{no}} > T \cdot \frac{S_{\text{no}} - (S_{\text{yes}} + n)}{S_{\text{yes}} + S_{\text{no}} + n} \tag{45}$$

$$\Leftrightarrow \quad P_{\text{yes}} - P_{\text{no}} > T \cdot \frac{S_{\text{no}}}{S_{\text{yes}} + S_{\text{no}} + n} - T \cdot \frac{S_{\text{yes}} + n}{S_{\text{yes}} + S_{\text{no}} + n} \tag{46}$$

$$\Leftrightarrow \quad P_{\text{yes}} + T \cdot \frac{S_{\text{yes}} + n}{S_{\text{yes}} + S_{\text{no}} + n} > P_{\text{no}} + T \cdot \frac{S_{\text{no}}}{S_{\text{yes}} + S_{\text{no}} + n} \tag{47}$$

Because inequalities (21) and (47) are identical and (27) ensures $n \in \mathbb{N}$, we conclude that proposition 3 is true. $\square$

In other words: If $\textsc{Ms. Evil}$ creates $n = \max(\lfloor n' \rfloor + 1,\ \lceil T_{\max} \rceil)$ additional sockpuppets (definition 27) and if the other non-verified accounts would be relevant for the outcome of the vote if $\textsc{Ms. Evil}$ didn't use her sockpuppets (inequalities 17 and 18), then $\textsc{Ms. Evil}$ can overrule the overall outcome of the vote (inequalities 18 and 21).

Note that we could still construct cases where $\textsc{Ms. Evil}$ can't gain control over the outcome of the vote. In those cases, however, the non-verified accounts would not have any effect on the outcome of the vote anyway, which is why non-verified accounts could have been excluded from casting votes in the first place.

In *all* cases where non-verified accounts have an actual impact on the outcome of the vote, manipulation is possible by adding a finite number of sockpuppets. It is therefore futile to try to curtail the sockpuppet problem by limiting the total voting-weight of all non-verified accounts.

2.7 Example

We choose the following example where inequalities 17 and 18 are fulfilled, i.e. where the non-verified voters $S_\text{yes} + S_\text{no}$ have an impact on the outcome:

$$\begin{aligned} P_\text{yes} &= 503 \quad & S_\text{yes} &= 48 \\ P_\text{no} &= 497 \quad & S_\text{no} &= 952 \\ T_\text{max} &= 10 \end{aligned}$$

According to proposition 3, it is certain that Ms. Evil can control the outcome of the vote with 191 sockpuppets:

$$\begin{aligned} n &= \max(\lfloor n' \rfloor + 1\,,\ \lceil T_\text{max} \rceil) \\ &= \max\left(\left\lfloor S_\text{no} \cdot \frac{1 - \dfrac{P_\text{yes} - P_\text{no}}{T_\text{max}}}{1 + \dfrac{P_\text{yes} - P_\text{no}}{T_\text{max}}} - S_\text{yes} \right\rfloor + 1\,,\ \lceil T_\text{max} \rceil \right) \\ &= \max\left(\left\lfloor 952 \cdot \frac{1 - \dfrac{503 - 497}{10}}{1 + \dfrac{503 - 497}{10}} - 48 \right\rfloor + 1\,,\ \lceil 10 \rceil \right) \\ &= \max\left(\left\lfloor 952 \cdot \frac{1 - \frac{6}{10}}{1 + \frac{6}{10}} - 48 \right\rfloor + 1\,,\ 10 \right) \\ &= \max\left(\left\lfloor 952 \cdot \frac{4/10}{16/10} - 48 \right\rfloor + 1\,,\ 10 \right) \\ &= \max\left(\left\lfloor 952 \cdot \frac{1}{4} - 48 \right\rfloor + 1\,,\ 10 \right) \\ &= \max\left(\lfloor 238 - 48 \rfloor + 1\,,\ 10 \right) \\ &= \max\left(\lfloor 190 \rfloor + 1\,,\ 10 \right) \\ &= \max\left(191\,,\ 10 \right) \\ &= 191 \end{aligned}$$

3 Conclusion

All voting systems where internet users may create non-verified accounts on their own behalf are also susceptible to the creation of "sockpuppets". Sockpuppets are fake identities created with malicious intent to manipulate public opinion and/or decisions by deception. Often, the reduction of barriers is brought up as an argument in favor of easy account creation and against proper accreditation systems (i.e. proper user identification and verification).

However, it can be proven that at least in those cases where the non-verified accounts have an impact on the outcome of the vote, a malicious attacker can overrule *any* binary decision by simply adding a finite number of sockpuppets, even if the voting-weight of each non-verified account is reduced to a small but non-zero value (section 1 of this paper) or if the total voting-weight of all non-verified accounts is limited (section 2 of this paper).

Since creation of sockpuppets already violates the democratic principle of "one man – one vote", these findings are only a supplementary reason for proper accreditation in democratic processes. Refer to the book "The Principles of LiquidFeedback" (ISBN 978–3–00–044795–2), section *"Who may participate? (And how are these people identified?)"* (pages 120–124) for further reading.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 693514.

Work report on Unified WeGovNow User Management (UWUM) development

Jan Behrens, Axel Kistner, Andreas Nitsche, Björn Swierczek

2016-12-12

1 Presentation of UWUM in Berlin

A first draft of UWUM has been presented in the kick-off meeting "Connecting The Bits" on April 14, 2016 in Berlin. The overall idea was to build a single-sign-on (SSO) solution on OAuth 2.0's Authorization Code[1] flow.

For access tokens, the use of bearer tokens[2] was proposed. Furthermore, it was agreed on that TLS is to be used to secure all communication between UWUM and other components.

In addition to single-sign-on, UWUM's capabilities were planned to include:

- a style endpoint, which allows applications to retrieve style information (e.g. a color scheme),
- a navigation endpoint, which allows applications to incorporate a common nagivation bar into their user interfaces, and
- a service discovery endpoint, which allows applications to retrieve a list of other applications within the system and their capabilities/protocols.

This way, WeGovNow is designed to be a modular system that may be extended with different services which are all connected through UWUM.

It was agreed that UWUM will be implemented by LiquidFeedback such that it is possible to use synergetic effects between the necessary creation of an API for LiquidFeedback and the newly created features required by UWUM.

[1]See `https://tools.ietf.org/html/rfc6749#section-1.3.1` for a short overview on the Authorization Code flow and `https://tools.ietf.org/html/rfc6749#section-4.1` for a detailed description.

[2]`https://tools.ietf.org/html/rfc6750`

2 Authentication and Authorization

For reasons of interoperability and security, we aimed to create an implementation that is fully compliant with RFC 6749.[3] In this section, the extensions necessary in addition to that document will be explained below. All functionality has been implemented by the time of publishing this work report except where otherwise noted.

2.1 Roles

RFC 6749 defines several roles in subsection 1.1.[4] The UWUM component as implemented by LiquidFeedback takes the role of the "authorization server". Other WeGovNow components will take the role of "clients" but may also act as "resource server" for other components.

2.2 Choice of protocol flow

UWUM requires the Authorization Code flow[1] for secure user authentication, i.e. when used for single-sign-on (SSO). (Note that subsection 10.16 in RFC 6749 explains why the Implicit flow[5] as defined by OAuth 2.0 is *not* suitable for secure user authentication.[6])

The Implicit flow[5] is still supported for clients which only require authorization but do not rely on secure user authentication (e.g. pure JavaScript clients which access other components but do not store themselves any resources which would need to be protected by SSO).

2.3 Types of clients

RFC 6749 distinguishes between "confidential clients" (which are capable of secure client authentication, e.g. by maintaining confidentiality of their client credentials) and "public clients" (which are incapable of secure client authentication). UWUM requires all clients which use OAuth 2.0's Authorization Code[1] flow (and thus receive long-lasting refresh tokens) to be capable of secure authentication; i.e. every use of the token endpoint (see subsections 2.7 and 2.8) will require client authentication (except when an access token scope downgrade

[3] `https://tools.ietf.org/html/rfc6749`

[4] `https://tools.ietf.org/html/rfc6749#section-1.1`

[5] See `https://tools.ietf.org/html/rfc6749#section-1.3.2` for a short overview on the Implicit flow and `https://tools.ietf.org/html/rfc6749#section-4.2` for a detailed description.

[6] `https://tools.ietf.org/html/rfc6749#section-10.16`

is performed, see subsection 2.14). The use of "public clients" is only supported for those clients which utilize the Implicit[5] flow because these clients will not handle any long-lasting tokens.

2.4 Client registration

Client registration is mentioned in section 2 of RFC 6749, even though the standard explicitly states that "the means through which the client registers with the authorization server are beyond the scope of [the] specification".[56] UWUM provides two methods of client registration:

- registering clients through the municipality (or their technical administration) or an organization running a particular installation of WeGovNow,
- registration of any other ("dynamic") client on a per-user basis by each user who wishes to use that client to access WeGovNow (machine accessibility).

These two registration methods are described in the following two subsections respectively.

2.4.1 Clients approved by the municipality

Clients approved by the municipality authenticate through TLS (X.509) certificates which are signed by the municipality or a certificate authority acting on their behalf. For example, the operator of the UWUM server could issue a certificate to the operator of each respective client. Furthermore, the operator of the UWUM server configures a list of automatically granted access scopes[7] for the particular client (not every client has the same automatically granted access scopes, e.g. some clients might not require voting rights). Any other access scope may be granted on a per-user basis by the respective end-user or be disallowed by the municipality for a particular client (through white or black lists).

This results in the following information being stored per client:

- name of client,
- OAuth 2.0 client identifier (`client_id`),
- redirect URI(s)[8],
- common name (CN) of the TLS certificate,

[7] `https://tools.ietf.org/html/rfc6749#section-3.3`

[8] See `https://tools.ietf.org/html/rfc6749#section-3.1.2` for redirection URIs. One redirect URI is the default redirect URI, other redirect URIs may be selected through the `redirect_uri` parameter, see: `https://tools.ietf.org/html/rfc6749#section-4.1.1`

- automatically granted scopes[7],
- white list of scopes (optional),
- black list of scopes (optional, i.e. may be empty).

2.4.2 Dynamic clients

For the sake of machine accessibility, it would be nice to allow unregistered clients. Unfortunately, OAuth 2.0 requires some sort of client registration (at least) for the following security reasons:

- allowing capability to authenticate a client,[9] in order
  - to avoid refresh token abuse by a third party in case of accidentially exposed refresh tokens,[10]
  - to avoid authorization code abuse (which could expose access and refresh tokens to a malicious 3rd party) in case of exposed authorization codes,[11]
- restriction of choice of the redirect URI[12], in order
  - to avoid redirection URI manipulation,[13]
  - to avoid open redirector attacks.[14]

In order to be able to provide an open platform, however, it should still be possible to use clients which have not been explicitly approved by the operator of the WeGovNow platform. Assuming there will be more than one WeGovNow installation (e.g. run by different municipalities, each operating their own system), this is necessary in order to enable third parties to provide generic clients that can be used by *any* WeGovNow platform, even those not known to the operator of the client.

Consequently, registration of these clients should happen dynamically without further human interaction.[15] This requires to automatically establish a channel

[9]See `https://tools.ietf.org/html/rfc6749#section-2.3` and `https://tools.ietf.org/html/rfc6749#section-10.1`

[10]`https://tools.ietf.org/html/rfc6749#section-10.4`

[11]`https://tools.ietf.org/html/rfc6749#section-10.5`

[12]`https://tools.ietf.org/html/rfc6749#section-3.1.2`

[13]`https://tools.ietf.org/html/rfc6749#section-10.6`

[14]`https://tools.ietf.org/html/rfc6749#section-10.15`

[15]We assume that every user of WeGovNow is legally entitled to use any client of his or her choice to access his or her data and to perform actions. In cases where a particular operator of LiquidFeedback (e.g. a municipality) wants to decline this right, the use of dynamic clients could be disabled.

of trust between the client and the UWUM server through secure authentication. UWUM relies on the following mechanism to archive secure authentication of a dynamic client:

- a dynamic client is only referenced by its domain, and
- at the choice of each client, registration is performed either
  - by adding a certain entry to the domain's DNS zone[16] or
  - temporarily through a REST API call to the UWUM server with a client-side TLS (X.509) certificate issued to the respective domain and signed by a publicly trusted certificate authority (e.g. "Let's Encrypt")[17].

Taking into account that it cannot be outruled that TLS certificates could accidentially be exposed to a malicious 3rd party and considering that there might be at least one publicly trusted CA which is vulnerable to a state-level attack,[18] we restrict the redirection URI[12] to the following static path on the web server's root level:

```
/liquidfeedback_client_redirection_endpoint
```

This repels any attempts of "authorization code redirection URI manipulation" as explaiend in subsection 10.6 of section 10 ("Security considerations") of RFC 6749 ("The OAuth 2.0 Authorization Framework")[13] even in cases where dynamic client registration could be forged.

Any client that cannot follow the above redirection URI convention must be registered by the municipality or organization running a particular installation of WeGovNow (see subsection 2.4.1).

As an additional security mechanism, the dynamic registration is always done for a set of access token scopes[7] to be used with a particular OAuth 2.0 flow. Thus a client's redirection endpoint registered for the Authorization Code flow cannot be used by the Implicit flow or vice versa unless the registration is broadened accordingly.

[16]A TXT DNS resource record needs to be added to the subdomain "`_liquidfeedback_client`" of the respective domain which must include a so-called magic string (namely "`dynamic client v1`") as first entry.

[17]The operator of LiquidFeedback is therefore required to decide on a list of trusted CA's. Many operating systems already ship with such a list of root certificates.

[18]Note that similar security considerations also apply to DNS and the risk of DNS cache poisoning or similar attack vectors. This could, however, be fixed by DNSSEC such that future versions of UWUM might lift the described restrictions for domains which are cryptographically secured.

The operator (e.g. a municipality) may still decide to disallow the use of non-approved (dynamic) clients completely. This would, however, limit machine accessibility and render the platform less open for extensions and unforseen use cases. An appropriate configuration option will be provided which can also be used to limit the access token scope of dynamic clients (using a white or black list).

Unless dynamic clients are entirely disabled, an additional security warning will be displayed to the user when authorizing such a client. The user will be requested to verify that:

- the client domain is trustworthy,
- the client domain is used to host a legit application to access LiquidFeedback,
- the spelling of the domain name (whose client is going to be authorized) is correct,
- the granted scope of access (access token scope) is intended by the user.

Clients which want to avoid these warnings must be approved by the municipality or organization that is operating the LiquidFeedback system (see subsection 2.4.1).

2.5 Access token types

As previously mentioned, bearer tokens[2] as defined in RFC 6750 will be used as access tokens. Therefore, the access token type ("`token_type`")[19] returned by UWUM is always set to "`bearer`".

2.6 Access token scopes

The following set of generic[20] access token scopes[7] has been specified:

`authentication`: Authenticate the current user by reading its unique static ID and current screen name.

[19]`https://tools.ietf.org/html/rfc6749#section-7.1`

[20]Application specific scopes could be introduced if they turn out to be necessary in the future. It would also be thinkable for dynamic clients acting as a resource server to provide a set of application specific scopes as part of their registration. Further security analysis would be required for such an extension. See also subsection 5.8 for considerations on generic versus application specific scopes.

`identification`: Identify the current user by reading its unique identification string. Automatically implies scope "`authentication`".

`notify_email`: Read the notification e-mail address of the current user.

`read_contents`: Read any user generated content (without authorship, ratings and votes).

`read_authors`: Read the author names of user generated content (author's static ID and screen name).

`read_ratings`: Read ratings (see scope "`rate`" below) by other users.

`read_identities`: Read the identities (identification strings) of other users.

`read_profiles`: Read the profiles of other users (e.g. phone number, self-description, etc).

`post`: Post new content.

`rate`: Rate user generated content (e.g. thumbs up/down, "+1", support an initiative, rate a suggestion).

`vote`: Finally vote for/against user generated content in a decision (e.g. vote on an issue in LiquidFeedback)

`profile`: Read profile data of current user (e.g. phone number, self-description, etc).

`settings`: Read current user's settings (e.g. notification settings, display contrast, etc).

`update_name`: Modify user's screen name.

`update_notify_email`: Modify user's notification e-mail address.

`update_profile`: Modify profile data (e.g. phone number, self-description, etc).

`update_settings`: Modify user settings (e.g. notification settings, display contrast, etc).

Note that any of these scopes can also be suffixed with "`_detached`" to request the scope for usage also when the user is not logged in (which will be explained in subsection 2.9).

2.7 User authentication (single-sign-on)

OAuth 2.0 by itself is not suitable for user authentication. Both the Authorization Code flow[1] and the Implicit flow[5] can be extended to provice user authentication and thus allow to implement a single-sign-on (SSO) system. Because the Implicit flow would require additional security mechanisms to be implemented at client side (where bad implementations result in security vulnerabilities),[6] UWUM extends the Authorization Code flow for the purpose of implementing an SSO solution as described in the following.

In order to protect against authorization code substitution attacks, the UWUM server checks the OAuth 2.0 client identity before accepting an authorization code.[21] This is both a requirement stated in subsection 4.1.3 of RFC 6749 ("The OAuth 2.0 Authorization Framework")[22] and a recommended countermeasure to avoid authorization code substitution attacks in subsection 4.4.1.13 of RFC 6819 ("OAuth 2.0 Threat Model and Security Considerations")[23].

The Access Token Response[24] of the OAuth 2.0 Authorization Code flow gets extended with the field "`member_id`" which returns the LiquidFeedback member ID of the signed-in user. OAuth 2.0 clients not aware of this extension are requested to ignore this field as stated in subsection 5.1 of RFC 6749.[25] Nonetheless, these clients may still pass the returned access token to the validate endpoint (see next section) in order to determine the `member_id` of the user who has logged in.

2.8 Endpoints

RFC 6749 defines two endpoint URIs at the authorization server side: the "authorization endpoint"[26] and the "token endpoint"[27]. These are defined as follows:

- `https://`*server_name*`/api/1/authorization` (GET)
- `https://`*server_name*`/api/1/token` (POST)[28]

Note that a base path may be appended to the *server_name* component if applicable.

[21] Note that, if the client is authenticating with the UWUM server, the `client_id` parameter can be ommitted by the client when accessing the token endpoint (see next footnote).

[22] `https://tools.ietf.org/html/rfc6749#section-4.1.3`

[23] `https://tools.ietf.org/html/rfc6819#section-4.4.1.13`

[24] `https://tools.ietf.org/html/rfc6749#section-4.1.4`

[25] `https://tools.ietf.org/html/rfc6749#section-5.1`

[26] `https://tools.ietf.org/html/rfc6749#section-3.1`

[27] `https://tools.ietf.org/html/rfc6749#section-3.2`

[28] The server name for the token endpoint may differ for those requests where TLS client certificates are used. See subsection 5.2 for explanation.

RFC 6749 does not specify any method for a resource server to "ensure that an access token presented to it by a given client was issued to that client by the authorization server".[29] Therefore, an additional validation endpoint has to be specified:

- `https://`*server_name*`/api/1/validate` (POST)

The validation endpoint does not require any parameters except the access token (bearer token) to be passed using the mechanisms described in section 2 of RFC 6750.[30] It returns a JSON object with the following fields:

- `scope`: a space separated list of scopes[7] associated with the access token (with any "`_detached`" suffix stripped off, see next subsection 2.9),
- `member_id`: an integer set to the id of the user who logged in,
- `logged_in`: a boolean set to false if the user has meanwhile logged out.

Note that the scope of an access token may change when the user logs out. This is explained in the following subsection 2.9. Subsection 2.12 will pick up the issue of user logout again.

There may be situations where an OAuth 2.0 client wants to check whether a user is currently logged in without actually forcing the user's web browser to perform a login if no user was logged in. To provide this functionality, a 4th endpoint (also out of scope of the OAuth 2.0 specification) is added at the authorization server side:

- `https://`*server_name*`/api/1/session` (POST)

This "session" endpoint can be accessed directly by a user's web browser (through a script performing a CORS[31] HTTP request with credentials). Its usage is further explained in subsection 2.10.

2.9 Binding lifetime of access and refresh tokens to a users web session by default

Access tokens have an expiry time after which they will be invalidated.[32] In addition to the maximum access token lifetime returned in the Access Token Response,[24] UWUM additionally limits the lifetime of both access tokens and

[29]See section 10.3 of the RFC: `https://tools.ietf.org/html/rfc6749#section-10.3`
[30]`https://tools.ietf.org/html/rfc6750#section-2`
[31]Cross-origin resource sharing, see `https://www.w3.org/TR/cors/`
[32]`https://tools.ietf.org/html/rfc6749#section-5.1`

refresh tokens to the user's web session at the LiquidFeedback (UWUM) server by default (i.e. if the user logs out, the access tokens and refresh tokens will be immediately invalidated).

Some clients, however, require access longer than the user's login session. For this purpose, access token scopes[7] with the suffix "`_detached`" may be requested (e.g. "`vote_detached`" instead of "`vote`"). Whether an application may request these scopes (as well as which scopes may be requested for detached access) depends on the configuration for the particular client, or – in case of dynamic clients – on the configuration for all dynamic clients. An access or refresh token that contains only detached scopes will not be invalidated on user logout. Access tokens, however, will still be invalidated when their expiry time (as denoted by the "`expires_in`" field in the Access Token Response[24]) has elapsed, in which case a refresh token must be used to obtain a new access token. Access and refresh tokens which contain both detached and non-detached scopes will only have their non-detached scopes removed on user logout instead of being invalidated completely.

Other than the behavior described above, the "`_detached`" scopes behave as any other scope for the authorization[26] and token[27] endpoint. Only the validation endpoint ("`api/1/validate`") will strip the suffix "`_detached`" from the scope field in its response because it doesn't matter for a validating resource server whether a scope has been granted detached from a web session or not.[33]

Even if the token lifetime is bound to the web session (i.e. when only non-detached scopes are requested), a user's logged in web browser may still automatically re-authorize the client whenever he or she is logged in at UWUM and visits the client's website. If such a client was authorized by the user, the permission can be revoked by the user at any time using a designated configuration dialog provided by the UWUM server.

2.10 Checking user login without triggering a login

An interactive UWUM client application may want to determine whether a user is logged in without actually triggering a login. OAuth 2.0 does not provide such a mechnaism on its own.[34] UWUM therefore provides an additional "session" endpoint (`https://`*server_name*`/api/1/session`, see subsection 2.8) to allow

[33]Neither RFC 6749 nor RFC 6750 are violated because the authorization and token endpoint treat detached scopes like any other scope and a validation endpoint is not covered by these RFCs.

[34]Also, extending the authorization endpoint by accepting a "prompt" parameter as done by OpenID Connect is not feasible for user-registered clients because non-logged-in users could be redirected to malicious clients registered by other users, making the system susceptible to open redirector phishing attacks. See subsection 5.5

web applications to gather information about the current login status of a user without actually triggering any (interactive) login or permission grant procedure. This endpoint is directly accessed by the user's web browser through an XML-HttpRequest (XHR) call while setting the "`withCredentials`" option of the XMLHttpRequest object to true.

The call does not need any parameters and should not have any additional request headers set[35]. It returns a JSON object with the "`member_id`" attribute set to the ID of the current user (or to null if there is no logged-in user or if a user-registered client is not authorized to obtain the login status). Since the request is done by the user's web browser, the answer is *not* authoritative for the UWUM client and must only be used as a hint. **A returned user ID MUST still be confirmed via the regular OAuth 2.0 procedure using the authorization endpoint!** In this case, the authorization endpoint will not show a login window (because the user is already logged in).[36]

2.11 Caching the login state

A successful user authentication could be cached in the session store of the UWUM client (usually at the web server side in conjunction with a cookie). This, however, can create confusion for the user because he or she might show up as being logged into the system after having logged out or vice versa. A possible solution is to use the "`session`" endpoint as discussed in the previous subsection 2.10 through a JavaScript which then notifies the server side of the UWUM client by redirecting the web browser if a reconfirmation of the user's login status is necessary.

In either case, UWUM clients should reconfirm that the user has not logged out at least immediately before any state changing request (e.g. posting, rating, voting, etc.) by using the validation endpoint (see subsection 2.8). This check cannot be done directy by the web browser due to security reasons (as also explained in the previous subsection 2.10).

[35]Not setting additional request headers avoids CORS pre-flight requests, see `https://www.w3.org/TR/2014/REC-cors-20140116/#cross-origin-request-with-preflight-0`

[36]There is a chance for a race-condition if the user simultaneously logs out. This could be solved by returning an authorization code through a CORS call. However, implementation of such a protocol is out of scope for WeGovNow and would require further security analysis.

2.12 Logout

2.12.1 Checking for logout

As explained in subsection 2.9, an access or refresh token is automatically invalidated on logout if only non-detached scopes have been requested. For all other cases, the "`logged_in`" boolean field returned by the validation endpoint (see subsection 2.8) may be used to detect a logout by the user.

The "session" endpoint (as further explained in subsection 2.10) may also be used to check whether a user might have logged out (without consuming much resources on the server-side of the UWUM client).[37] Note, however, that a request from the web browser to the session endpoint is not suitable for the UWUM client application to validate that a user is really logged in or to securely confirm that his or her session has really ended (see subsection 2.10).

2.12.2 Performing logout

Depending on design criteria, logout could be performed either

- through a direct link in the UWUM navigation bar or
- through a link in the UWUM navigation bar which leads to a user page where there is a second link for the actual logout procedure.

Technical implementation requirements differ for these two cases. In the first case, the logout is performed in the context of any UWUM client; while in the second case, the final logout link or button can be displayed in the context of a web page returned by the UWUM server (which is a different origin). Due to protection against cross-site-request-forgery (CSRF), an appropriate access token or dedicated logout token would need to be part of the link in the first case (the case of using a direct link for logout). In this case, an appropriate OAuth 2.0 access token scope would need to be added to avoid unwanted exposure of the logout token (or an access token with respective scope).

A decision on this issue has not been taken yet; user interface design considerations and technical security considerations should determine which of the discussed two approaches is more suitable. Also refer to subsection 5.10, which discusses certain design limitations due to privilege separation.

[37]A future extension of UWUM could also allow UWUM clients (or their JavaScript components at the web browser side) to issue a request which is held open by the UWUM server for a set amount of time in order to allow pushing a change of the user's login status just-in-time (see also subsection 5.4).

2.13 Requesting several access token scopes at once

To avoid unnecessary delays, a client may (as an extension to RFC 6749) request several access token scopes[7] (i.e. sets of access ranges) at once by using the parameters "`scope1`", "`scope2`", etc. in the Authorization Request. The corresponding result parameter "`access_token`" will have "1", "2", etc. appended to its name (e.g. "access_token1" etc.). Note that counting must start with "1". It is, however, allowed to include an optional non-numbered "`scope`" parameter in addition to "`scope1`", "`scope2`", etc. The result parameters "`token_type`" and "`expires_in`" are never numbered or duplicated due to size limitations in the Implicit flow (maximum URL length) but always relate to all returned access tokens.

The described behavior of this subsection is not part of OAuth 2.0. Using this extension is entirely optional for the client.

2.14 Downgrading access token scopes

As an extension to RFC 6749, the token endpoint has been extended in such a way that it can be used to downgrade access token scopes. This feature is important for meta-APIs because according to RFC 6749, the only way to obtain a new access token without the user's web browser is to provide a refresh token to the token endpoint.[38] Refresh tokens, however, are bound to a particular client and must not be shared by the client with any other party but the authorization server.[39]

A meta-API might receive an access token with a broader scope[7] than the scope necessary for calls made by the meta-API provider to another resource server. Using a greater scope than necessary for calls to resource servers, however, weakens the overall security of the system. In order to allow meta-API providers to downgrade the scope prior to using the access token, the token endpoint[27] accepts the string "`access_token`" as value for the "`grant_type`" parameter, which will tell the UWUM server that an access token (and not an authorization code or refresh token) is being presented to receive a new access token with a downgraded scope. The access token has to be provided according to the rules stated in section 2 of RFC 6750,[30] and one or more scopes must be requested through the "`scope`", "`scope1`", etc. parameters (see subsection 2.13 for details on requesting several scopes at once). Client authentication is not required. The old access token with the broader scope will not be invalidated and may still be used in future requests (e.g. to receive another access token with a different scope).

[38]`https://tools.ietf.org/html/rfc6749#section-6`

[39]`https://tools.ietf.org/html/rfc6749#section-10.4`

For security reasons, downgrading an access token scope will never extend the token lifetime, i.e. the returned access token will have the same remaining maximum lifetime than the access token presented to the token endpoint.[40]

2.15 Additional measures to prevent refresh token abuse

Conforming with section 10.4 of RFC 6749,[41] the UWUM server (LiquidFeedback) ensures that refresh tokens are bound to the client they have been issued to. As also suggested in subsection 10.4 of RFC 6749, further means to restrict refresh token abuse are implemented. Refresh tokens are replaced periodically and using a refresh token invalidates the corresponding scope[7] of all other previously issued refresh tokens, with the exception that refresh tokens which are still bound to a logged in user are unaffected.[42] An additional grace period avoids problems due to race conditions or aborted connections. This approach is similar to the example given in subsection 10.4 of RFC 6749 while being resistant against accidental race-conditions or connection aborts and allowing for a more flexible usage (e.g. different subsystems of the same client may store different refresh tokens indepenently).

2.16 Required CORS support for resource servers

Because RFC 6750 requires bearer tokens[2] to be accepted through the HTTP header "`Authorization`",[43] and because the "`Authorization`" header is not in the list of "simple response headers" as defined by the W3C recommendation on cross-origin resource sharing,[44] it is inevitable for all resource servers to support cross-origin resource sharing (CORS) with the respective "`Access-Control-Allow-Headers`" option[45] set to be able to fulfill the requirements of RFC 6750. Every UWUM component acting as a resource server should therefore enable and configure CORS accordingly. See `https://www.w3.org/TR/cors/` for details.

[40]This is the reason why client authentication would not grant any extra security here and thusly can be omitted.

[41]`https://tools.ietf.org/html/rfc6749#section-10.4`

[42]This is implemented by downgrading "_detached" scopes to their corresponding non-detached scopes.

[43]`https://tools.ietf.org/html/rfc6750#section-2.1`

[44]`https://www.w3.org/TR/cors/#terminology`

[45]`https://www.w3.org/TR/cors/#access-control-allow-headers-response-header`

2.17 HSTS

We recommend to use HTTP Strict Transport Security (HSTS)[46] for all WeGovNow components to increase security.

3 Additional endpoints for integration

Beyond user authentication and authorization, three more API endpoints are being defined for backend and UI integration:

- a "`navigation`" endpoint to incorporate a navigation bar,
- a "`style`" endpoint to retrieve style information, and
- a "`client`" endpoint for applicaton and service discovery.

Prototypes for the navigation and style endpoint have been implemented; the client endpoint for application and service discovery is currently only a stub.

3.1 Navigation endpoint

In order to integrate all WeGovNow applications in such a way that they look and feel like a single application, all WeGovNow applications share a common navigation bar. The "`navigation`" endpoint of the UWUM server returns this navigation bar to be included by each WeGovNow application. This way, modifications to the navigation bar can made at a central place without the need to change every single application.

Either a login button or the user name with a link to a user page (where logout is possible) is included in the navigation bar, depending on whether an access token is provided when calling the endpoint.[47] For the login button, an alternative URL may be provided by the caller of the navigation endpoint. This login URL may either be the authorization endpoint of the UWUM server with an appropriate "`state`" HTTP GET parameter included (note that the value must be percent-encoded[48]) or an URL provided by the UWUM client which initiates the OAuth 2.0 authorization and authentication procedure as described in section 2 of this document. Alternatively a unique placeholder (e.g. a GUID)

[46]`https://tools.ietf.org/html/rfc6797`

[47]Also a dynamic popup menu is thinkable. However, issues with JavaScript and privilege separation in case of animated submenus according to Material Design require further consideration. Refer to subsection 5.10 in that matter.

[48]`https://tools.ietf.org/html/rfc3986#section-2.1`

can be passed as login URL to allow caching of the rendered navigation bar and replacing the login URL locally at the UWUM client.[49]

Whether a structured JSON document or a pre-rendered HTML snippet is returned can be selected by another parameter passed to the navigation endpoint. A pre-rendered HTML snipped may be either returned encapsulated in a JSON response or raw for usage with the HTML5 include tag.

When the "`client_id`" parameter is provided to the navigation endpoint, the corresponding client tab gets highlighted (or marked as active in case of the structured JSON document response).

It is planned to collapse the navigation bar on small screens. This feature might interfere with application specific menus; refer to subsection 5.12 for that matter.

3.2 Style endpoint

The style endpoint provides basic color definitions for a primary and an accent color as 8-bit RGB triplet to be able to customize the unified visual look of all WeGovNow applications for a particular installation by central configuration. Additional colors can be derived from these two base colors. If the UWUM server gets configured with colors from the Material Design color palette, the corresponding Material Design color name of the primary and the accent color is also provided.

3.3 Endpoint for application and service discovery

The endpoint "`client`" is supposed to return a list of all system applications and, if an access token is provided, a list of all registered dynamic clients for the corresponding user. Implementation of this endpoint will require storing the base URL of all system applications at the UWUM server.

Further discussion with OntoMap is required for specification and implementation of this endpoint.

4 Test platform

A test platform has been created in mid September to start integration with the other consortium partners.

[49]Note that the characters "<", ">", "&", as well as the quotation mark character should be avoided in a placeholder string because these characters would get HTML entity encoded as described in subsection 8.1.4 of the HTML5 standard, see: `https://www.w3.org/TR/html5/syntax.html#character-references`

4.1 Benchmarks

The following benchmarks for integration have been defined, whose fulfillments have been published in the weekly status reports.

Client URLs established A client application (resource server and/or relying party) has been installed and its base URL and redirection endpoint[50] has been communicated to the consortium.

SSL key and certificate for end-users A private key and a publicly trusted SSL certificate has been created for the end-user web interface and SSL connections to that interface have been successfully tested.

Certificate signing request (CSR) for UWUM API A private key for accessing the UWUM API and a corresponding certificate signing request (CSR) has been created and submitted to FlexiGuided GmbH (LiquidFeedback).

SSL certificate for UWUM API A signed certificate for the UWUM API client key has been sent back to the consortium member and their client application has successfully established a secured connection with the UWUM server.

Authorization endpoint accessed The client application can redirect an end-user to the UWUM authorization endpoint.[51]

Authorization endpoint error response handling The client application is capable of receiving authorization errors[52] through its redirection endpoint[50] and displaying it to the end-user.

Access token request (including end-user identification) The client application has successfully received an authorization code and identified the end-user through an access token request.[53]

Access token request error handling The client application is capable of properly processing errors during the access token request.[54]

Using access tokens for API calls to other components The client application has successfully used an access token to perform a LiquidFeedback API call.

[50]https://tools.ietf.org/html/rfc6749#section-3.1.2
[51]https://tools.ietf.org/html/rfc6749#section-4.1.1
[52]https://tools.ietf.org/html/rfc6749#section-4.1.2.1
[53]https://tools.ietf.org/html/rfc6749#section-4.1.3
[54]https://tools.ietf.org/html/rfc6749#section-5.2

Access token verification The client application is capable of verifying the validity and scope of an access token.

Accepting access tokens from other components The client application provides at least one API call where an access token is used for authorization.

Accepting access tokens as "Authorization" header In conformance with RFC 6750 (Bearer Token Usage), the client application (resource server) accepts access tokens through the authorization request header field.[55]

Cross-origin resource sharing The client application allows cross-origin resource sharing (CORS) as described in subsection 2.16 of this document.

Cross-application navigation The UWUM navigation bar has been successfully integrated into the client application.

IPv6 IPv6 capabilities have been tested.

5 Technical challenges

In this section, we will describe obstacles encountered during implementation and during integration with the consortium partners as well as respective solutions.

5.1 Third party clients (non-registered clients vs. dynamic registration)

OAuth 2.0 demands client registration but does not specify how such client registration is to be implemented.

> "Before initiating the protocol, the client registers with the authorization server. The means through which the client registers with the authorization server are beyond the scope of this specification but typically involve end-user interaction with an HTML registration form."[56]

Manual client registration, however, is only suitable for a service-centered approach where a software provides only a single service (e.g. Facebook, Google, Twitter, etc). An open source solution, however, could be installed at several sites by different service providers. It is therefore not sufficient to register a client

[55] `https://tools.ietf.org/html/rfc6750#section-2.1`

[56] Ed. D. Hardt: The OAuth 2.0 Authorization Framework, October 2012. Section 2 (Client Registration), `https://tools.ietf.org/html/rfc6749#section-2`

at a single service provider if this client shall be usable for any service provider using the UWUM server software.

One possible solution would be the creation of a central (i.e. world-wide) UWUM client registry. Such central client registry, however, could be a single point of failure and would empower a central authority to control usage of the UWUM protocol (e.g. it would be possible to block certain clients). We consider this approach contrary to the concepts of open source and open data.

Therefore, we implemented a dynamic client registration protocol that keeps implementational complexity at a minimum while providing good security properties which outperforms many other solutions for client registration due to requiring direct access to the DNS zone of the domain (for adding a TXT record) or credentials (a publicly trusted TLS certificate with corresponding key) that should be accessible only by the domain owner. Dynamic client registration is described in subsection 2.4.2 of this document.

5.2 TLS client side certificates and web browser behavior

Web server software often offers three different settings for handling TLS client certificates:

- client-side certificates disabled,
- optional client-side certificate,
- mandatory client-side certificate.

Often these settings can be made only on a per-domain basis (i.e. for each virtual host). Furthermore, enabling client-side certificates (even if set to "optional") will cause web browsers to show up a dialoge when accessing pages on that domain.

For these reasons, a separate hostname has to be used for API endpoints when a TLS client-side certificate is to be provided (which affects the token endpoint[27]). The UWUM server will have to provide a configuration endpoint where dynamic clients may retrieve a deviant domain for the token endpoint; and dynamic UWUM clients (see subsection 2.4.2) will have to query this configuration endpoint prior to using the token endpoint).

5.3 Multi-domain certificates

TLS certificates may be issued for more than one domain using the "Subject Alternative Name" (SAN) extension. The current implementation of Liquid-Feedback, however, relies on an HTTP reverse proxy to include the distinguished name (DN)[57] of the certificate in a designated HTTP header. Some reverse proxy software, namely "NGINX" which is recommended for use with LiquidFeedback, does not properly support transmitting a domain list from the SAN extension. In case of "NGINX", header line folding[58] is used to pass multiple domain names from a TLS certificate to the respective backend (e.g. LiquidFeedback). Header line folding, however, has recently been deprecated by RFC 7230,[59] and it is not supported by LiquidFeedback (and not even by "NGINX" for incoming requests). The problem of header line folding in the context of multi-domain TLS certificates has also been discussed in the "NGINX" issue tracker under ticket #857.[60] The issue is currently not classified as bug[61] and it is unclear when a patch will be incorporated into the software.

For the technical difficulties explained above, we refrained from supporting multi-domain certificates at this stage. In case of UWUM clients approved by the municipality or operator of LiquidFeedback, this shouldn't be a problem anyway because the certificate authority will be under the control of the operator, such that it is easy to create a certificate using the DN/CN property. For dynamically registered clients, an alternative mechanism using DNS TXT records is available (see subsection 2.4.2).

If multi-domain certificates are supported in the future, it is vital that the token endpoint requires the "`client_id`" parameter to be set for all clients authenticating with such a multi-domain certificate. This way, code substitution attacks[23] can be repelled. (Note that RFC 6749 requires the "`client_id`" parameter to be set only if the client is not authenticating with the authorization server;[22] but this does not work for multi-domain certificates.)

5.4 Outdated logins

While a successful OAuth 2.0 authorization procedure (using the Authorization Code flow[1]) can be used to confirm that a user is logged in at the particular time of the Access Token Response[24], an UWUM client obviously can't assume that the login will be still valid at any later time.

UWUM currently provides two methods to check if a user has logged out; these are explained in subsection 2.12.1 of this work report. Considerations in regard to purposeful caching of the user's login status are found in subsection 2.11.

Even if no caching of the login status is performed, there is still the possibility that a user opens WeGovNow with two different browser windows or browser tabs.

[57]The DN contains a single domain as CN (common name).
[58]`https://tools.ietf.org/html/rfc2616#section-2.2`
[59]`https://tools.ietf.org/html/rfc7230#appendix-A.2`
[60]`https://trac.nginx.org/nginx/ticket/857`
[61]`https://trac.nginx.org/nginx/ticket/857#comment:2`

He or she might then log out in one window and afterwards switch to the other window where the logout has not been noticed yet, which creates confusion for the user. A possible solution is to regularly check if the user has logged out by utilizing the cross-origin-resource-sharing (CORS) XML-HttpRequest (XHR) as explained in subsection 2.10.

Regular requests to detect logouts, however, cause unnecessary resource consumption for all involved components. A better approach would be to have a permanent TCP connection between the web browser and the UWUM server (or alternatively between the UWUM client and the UWUM server if at the same time there is a permanent connection between the web browser and the UWUM client). There are different technologies thinkable for this approach. One method is to keep an XML-HttpRequest open for a set amount of time during which the server is capable of sending a message directly to the web browser. (The request has to be repeated after timeout or after a message has been received, whichever happens first.) Another technique would be to use WebSockets. None of these additional techniques have been implemented yet.

5.5 Susceptibility to open redirector phishing attacks when allowing login checks through web browser redirection

Subsection 2.10 mentioned that an interactive UWUM client application may want to determine whether a user is logged in without actually triggering a login. OAuth 2.0 does not provide such a mechnaism on its own, and our research concluded that any form of redirection-based mechanism for providing this functionality[62] would be susceptible to open redirector phishing attacks as described in subsection 4.2.4 of RFC 6819 ("OAuth 2.0 Threat Model and Security Considerations")[63] as long as third parties are capable of registering a malicious client with a corresponding redirection URI[12] that is under the control of the third party.

The previously mentioned subsection of the threat model and security considerations document (RFC 6819) suggests client registration with redirect URI registration (and avoiding redirects to any non-registered redirect URI)[64] as only countermeasure for this threat. However, this countermeasure only works when manual client registration (and manual approval through the operator of the UWUM server) is mandatory. It particularly fails if dynamic client registration (e.g. as described in subsections 2.4.2 and 5.1 of this work report) is allowed.

[62]e.g. accepting a "prompt" parameter as done by OpenID Connect, see `http://openid.net/specs/openid-connect-core-1_0.html#AuthRequest`

[63]`https://tools.ietf.org/html/rfc6819#section-4.2.4`

[64]`https://tools.ietf.org/html/rfc6819#section-5.2.3.5`

Luckily, the technique of cross-origin resource sharing (CORS) allowed for the development of an alternative to the redirect-based approach. Subsection 2.10 explains the mechanism.

5.6 Handling of updated user related data (e.g. user's e-mail addresses)

When a WeGovNow application wants to send notification e-mails to users, it is not adequate to retrieve the e-mail address only once from UWUM as the notification e-mail can be changed by the user at any time. Such a change needs to be reflected by all applications using this e-mail address. Therefore an application needs to retrieve the current notification e-mail address *directly* before using it, in fact again before every usage.

For that purpose, another API endpoint `/api/1/notify_email` (GET) can be used (using an access token with the "`notify_email`" scope). To be able to retrieve the e-mail address while the user is not currently logged in, it will be necessary to request the "`notify_email_detached`" scope when identifying the user and to store the received refresh token permanently. The suffix "`_detached`" requests a scope for detached usage, i.e. for usage even after the user logs out.[65]

Similar situations can occur related to other member properties stored in one application but used in another one, e.g. the screen name. But these seem not to be as critical as to avoid using an outdated e-mail address. Such properties could be cached for a limited time before retrieving them again from the application storing this property.

5.7 Race conditions with refresh token rotation

As suggested in subsection 10.4 of RFC 6749,[41] refresh token rotation is employed to provide better security properties (e.g. in case of exposed refresh tokens and client certificates, or in case of the existence of a single compromized certificate authority which would render authentication of dynamic clients insecure).

Unfortunately, RFC 6749 does not specify how old refresh tokens are invalidated. Section 6 of RFC 6749 only says that[66]

- the authorization server MAY issue a new refresh token, in which case
- the client MUST discard the old refresh token and replace it with the new refresh token, and

[65] Note that when exchanging a refresh token for an access token after the user has been logged out, an UWUM client must also explicitly request the "`*_detached`" scope(s) it needs, e.g. "`notify_email_detached`" using the scope parameter of the `/api/1/token` endpoint.

[66] `https://tools.ietf.org/html/rfc6749#section-6`

- the authorization server MAY revoke the old refresh token.

Always revoking the old refresh token after transmission can have a bad effect on system stability, considering that responses might be interrupted. Furthermore, multiple backends of an UWUM client could simultaneously access the token endpoint. Such legit accesses by two legit backends of the same client would need to be distinguished from accesses by a legit client and a malicious third party who obtained a copy of a refresh token.

Subsection 2.15 explains the mechanisms employed by the UWUM server to mitigate the risk of refresh token abuse while solving the problems stated above.

5.8 Creating a set of suitable access token scopes

A useful set of access token scopes[7] is a vital aspect of privilege separation. From a security point of view, scopes should be as fine-graded as possible, particularly there should be different scopes for different applications (e.g. an application that wishes to rate user contributions in application X does not need an access token that allows to rate user contributions in application Y). Extensibility, on the other hand, would be complicated if access token scopes always refer to a single application (i.e. a single resource server in this context). Furthermore, it is a goal that the WeGovNow platform looks and feels like a single integrated application. When users grant access scopes to third party clients, such application-based scopes would be difficult to understand for the user, which by itself can have bad influence on the overall system security.

We therefore decided to provide a set of generic access token scopes as listed in subsection 2.6. For future extensions, see footnote 20.

5.9 Misconceptions regarding scopes vs. user privileges

Scopes must not be mistaken for user privileges. I.e. a scope does *not* grant a privilege to a user; it just means an application can trigger an action within the scope *if* the user is authorized to perform the action. For example, an application needs the scope "`vote`" to cast a vote on behalf of the user but casting a vote will only work if the user has the necessary voting privileges.

Programmers of UWUM clients must keep these differences in mind and execute an action only if both the scope and the users privileges are sufficent for the respective action.

5.10 JavaScript integration and privilege separation

Dynamically sharing JavaScript code between UWUM clients or between the UWUM server and an UWUM client violates privilege separation because it would enable one component to execute code in the security context of another origin. For example, one application 'A' could send a harmful JavaScript to be included in a web page returned by another application 'B' which then discloses the session cookie for application 'B' to application 'A'.

For this reason, the common navigation bar as returned by the navigation endpoint (see subsection 3.1) currently does not include any JavaScript code. UWUM clients may therefore even consider to sanitize the returned HTML code in such a way that any JavaScript is removed or rejected.

Interface design decisions, however, might suggest to use JavaScript for the navigation bar. Material design, for example, requires popup-menus to be animated, which cannot be done with CSS alone. Another reason for JavaScript might be dynamic modifications of the navigation bar (e.g. collapsing the navigation bar to a menu icon) depending on the screen size or the device of the user. Also other integration techniques might suggest the use of JavaScript.

An alternative to dynamically provided JavaScripts by the UWUM server would be a common library to be included locally by each WeGovNow component. Whenever this library is updated, administrators of each component can look over it before incorporating it. While this approach provides proper privilege separation, its downside would be the administrative overhead.

At least in regard to the navigation bar, it would eventually need to be decided whether

- there will be no JavaScript used by the navigation bar,
- the UWUM server will dynamically return JavaScript code for the navigation bar, or
- each WeGovNow component needs to include a pre-distributed JavaScript.

5.11 Logout through navigation bar

The common WeGovNow navigation bar (as returned by the "`navigation`" endpoint, see subsection 3.1) should also include a possibility to logout. Due to protection against cross-site-request-forgery (CSRF) and because the navigation bar will be included in responses from different web servers (different "origins"), a simple logout link does not work. Subsection 2.12.2 deals with different approaches to this problem.

5.12 Collapsing navigation bar and application menu

In case of mobile devices, it may be desirable to collapse the navigation bar to a single menu icon displayed in the corner of the screen. Despite the technical problems in regard to JavaScript (which are discussed in subsection 5.10), there is also a challenge in regard to a potentially existent second menu bar which is provided by the particular application currently selected.

It could be difficult for the user if two menu icons are being displayed (i.e. a meta-menu, which covers the entries of the navigation bar, and an application specific menu). A potential solution could be to combine both menus into a single one. In this case, however, the considerations of subsection 5.10 still apply.

5.13 UWUM clients without user interface

In addition to UWUM clients having a user interface, there are also WeGovNow applications thinkable which do not have any (end-)user interface. This includes both meta-API providers as well as other service components. In the context of WeGovNow, one meta-API provider could be OntoMap.

The current UWUM specification enables the development of meta-APIs because access tokens are not bound to a particular UWUM client and can be downgraded in regard to their access token scope (of which the latter is important for security, see subsection 2.14). Thus, a meta-API can simply require its callers to provide a valid access token which then can either be used directly or downgraded for further requests performed by the meta-API provider to other resource servers.

Nonetheless, the mechanisms described in this work report still require privileges that are bound to a particular user. For UWUM clients requiring access privileges that are not tied to a particular user (e.g. clients which aggregate data of all users and publish that information), the Client Credentials Grant[67] should be implemented.

5.14 Client authentication for resource servers

While UWUM enables (a) its clients to authenticate users and (b) resource servers to verify user authorization (both explained in section 2), it does not enable resource servers to authenticate clients. Such client authentication might be required by applications that want to establish a trusted channel to another application independently of user authorization.[68]

[67] `https://tools.ietf.org/html/rfc6749#section-4.4`

[68] An example could be OntoMap logging actions executed at other applications (which are then reported to OntoMap by the respective application with client authentication enabled).

Unfortunately, neither OAuth 2.0 nor UWUM enable applications to verify the identity of another application. Even if OAuth 2.0 uses client authentication for a variety of reasons[69] for the authorization endpoint[26], it doesn't provide such an authentication method to other applications. Extending the OAuth 2.0 work flow in this matter (e.g. by returning the `client_id` when an access token is presented to the validation endpoint[70]) would rise some issues:

- Tieing an access token (a bearer token[2] in case of UWUM) to a particular client does not make sense in case of applications that behave both as an OAuth 2.0 resource server and as a client (e.g. meta-API providers or applications which provide an API and have to perform further API calls to complete a requested action). Also, client impersonation would be possible. To give an example: if a received access token is tied to a particular client A, and if application A uses this access token to perform an action at application B, then application B would be able to impersonate application A. Furthermore, application B couldn't use the access token to authenticate as application B when performing further requests at the API of another application C.

- Using a custom scope to identify the origin of a request (e.g. a scope "`I_am_appX`") would also enable client impersonation (e.g. any application who receives an access token with the scope "`I_am_appX`" could then impersonate application X). An alternative could be to use scopes that reflect better the particular action to be performed, e.g. a scope "`write_appXs_log_at_appY`". It is self-evident that this would increase the number of scopes drastically (possibly quadradically), which, in turn, might create a maintenance/configuration mess. Other than that, there is another problem with using scopes for client authentication: following RFC 6750, there can only be one bearer token per request.[2] If a client needs to use a received access token for an API call at another component, then this access token could not be used to authenticate that client because it won't have the necessary scope. One possible solution could be to allow adding scopes to an existing access token or extend RFC 6750 in such a way that multiple access tokens could be used per request. All those solutions, however, go far beyond OAuth 2.0 and would require extra implementation work for all consortium partners. In the end, the created solution wouldn't be OAuth 2.0 anymore.

[69]See beginning of subsection 2.4.2 of this report.
[70]See subsection2.8 for an explanation of the validation endpoint.

The straight-forward way of authenticating clients is to use the existing mechanism already employed by all UWUM clients: TLS client-side certificates. This, however, requires TLS client certificate checking by each resource server that needs to authenticate other clients.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 693514.