FROSTDAO: Collective Ownership of wealth using FROST

Rahim Klabér

To obtain the degree of Master of Science in Computer Science
Software Technology Track
To be defended publicly on August 30th, 2023

Student number:	4790227
Thesis committee:	Dr. ir. J.A. Pouwelse, supervisor
	Dr. ir. Jie Yang

Delft University of Technology
Faculty of Electrical Engineering, Mathematics & Computer Science
Distributed Systems Group

FROSTDAO: Collective Ownership of wealth using FROST

Rahim Klabér, Johan Pouwelse (thesis supervisor)

Distributed Systems
Delft University of Technology

—master thesis—

Abstract

The introduction of Blockchain technology has led to the idea of Decentralized Autonomous Organizations. (DAOs). DAOs enable decentralized and transparent collective wealth, allowing groups of individuals to pool their funds for collective management. DAOs rely on complex smart contracts to function. Despite Bitcoin being the most popular Blockchain, its simple smart contracts make DAO unfeasible. This thesis presents a DAO structure that addresses the challenge of DAOs on Bitcoin. The system utilizes threshold signatures to allow anyone to create Bitcoin wallets jointly controlled by a group. The system is implemented as an Android application and relies on no central party, allowing the system to be used by anyone worldwide. Our experiments show that our system is practical for real-world use for groups of 50 members. Joining an organization in the system and creating a transaction to spend funds can be done in under a minute. Further research is required to determine how far the system can scale while being practical.

1 Introduction

In their simplest form, banks act as a middleman between depositors and borrowers. They pool the deposits together and give out personal and corporate loans taking profit by charging a fee. In the years leading to the 2008 financial crisis, banks invested in excessively risky loans using depositors' funds, requiring government bailouts [1]. Recently, the financial system has once again been put to the test with the failures of multiple banks [2]. Banks act as gateways to today's financial world. Without a bank, a person cannot easily invest, pay online or get loans. Nevertheless, these bailouts have shown that bankers may recklessly act in the disinterest of their customers for profit.

Bitcoin presents itself as an alternative to the global financial system [3]. It gave individuals an alternative to banks by allowing anyone to transact on the Internet securely. With Bitcoin, no banks can invest your money into risky products. Some thought Bitcoin could be an alternative to the current financial system [4]. However, Bitcoin has primarily remained a tool for speculation [5]. High fees, low scalability, and a lack of user-friendly applications hinder Bitcoin use [6].

Collective wealth management would allow a group of individuals to manage the group's funds collectively. Each individual has an equal say in how the funds are used. Collective wealth management is a crucial first step to an alternative financial system. Such a system would allow a group of individuals to be their own bank, where each individual is part owner. Collective wealth management allows money to be pooled and invested. This process is transparent and a majority of the participants need to agree for any action to be taken. This idea has been realized with Decentralized Autonomous Organizations (DAOs) on other Blockchains, like Ethereum [7]. DAOs are Blockchain-based organizations operating autonomously without central control by using smart contracts [8]. While collective ownership of Bitcoins is possible, it is impractical due to high fees and low scalability [9], [10].

In this thesis, we contribute to the goal of making Bitcoin an alternative to the financial system. We describe and partially implement a critical primitive for the collective ownership of wealth. Using this primitive, individuals can create shared Bitcoin accounts with hundreds of others. Anyone can use our system, which is compatible with existing Bitcoin tools and services.

Specifically, this thesis makes the following contributions:

- *Collective wealth* - We designed a DAO allowing for the collective ownership of Bitcoins and implemented it as an Android application.
- Performance analysis - We analyze the scalability of our system using various experiments. As part of this, we uncovered performance issues with IPv8's EVA protocol.

2 Problem Description

We focus on the problem of wealth under democratic control. Our goal is to enable a leaderless group of collaborating humans of unbounded size to control a Bitcoin wallet of unconstrained wealth democratically.

Decisions in traditional organizations are made by a small group, with the other members having little influence. This changed with the internet. The internet revolutionized how individuals collaborate and work towards a common goal [11], [12]. Individuals working together over the internet form informal organizations. One example is Wikipedia, a free encyclopedia with thousands of contributors [13]. However, internet organizations are not fully decentralized; Managing funds cannot be done by the whole group. Thus the persons managing the finances are more powerful. Decentralized Autonomous Organizations (DAOs) changed this by allowing every member of an organization to decide how the funds are managed. DAOs are leaderless organizations using Blockchain technology to operate without central control or any central authority [8].

The limitations of existing solutions hamper the possibility of DAOs on Bitcoin. Existing solutions are either expensive in terms of Bitcoin fees or expensive in terms of computation [10], [14]. No solution is scalable enough to enable a DAO with thousands of members. In addition, most DAOs are created using advanced smart contracts, which Bitcoin does not support.

Existing DAOs have several problems. First, They are often not decentralized or leaderless. Instead, they are controlled by a committee that enacts the will of the DAO [15]. Regular members can vote on proposals by using services that record the votes. However, the votes have no real power, and the DAO is always at risk of losing its funds if the committee decides to turn malicious. Second, they are hard to use by the everyday user, which limits participation. Some actions are complex and are only done by those adept with technology. The complexity of DAOs also means they are highly susceptible to hacks [16].

Thus, the challenge lies in creating a truly decentralized and leaderless DAO that is scalable and secure to enable collective wealth management. Bitcoin fees and computation must be minimized. Every member must have a say in governance. The DAO must be secure against hacks. The code should be bug-free, and all actions must be transparent to detect malicious behavior.

3 System Design

We aim to solve the problem by creating a peer-2-peer, leaderless, and decentralized system. This system will allow groups of individuals to form organizations similar to Decentralized Autonomous Organizations (DAOs) that enable them to collectively and democratically manage their wealth [8].

3.1 Bitcoin threshold signatures

The cornerstone of our system is the combination of Bitcoin and the FROST threshold signature scheme [17]. Bitcoin allows anyone to send and receive money over the Internet. Bitcoin is decentralized and has tamper-proof and verifiable transactions, allowing anyone to use it without relying on a central party. In our system, each group of participants jointly controls a Bitcoin account that requires a majority to spend funds. We use the FROST threshold-signature scheme to enable collective wealth management without using complex smart contracts [17]. A threshold-signature scheme allows t members of a group of size n, where $t \leq n$, to create a signature jointly. A threshold-signature scheme consists of an algorithm for generating a shared key pair and a signing algorithm for creating joint signatures. FROST was chosen for its good performance

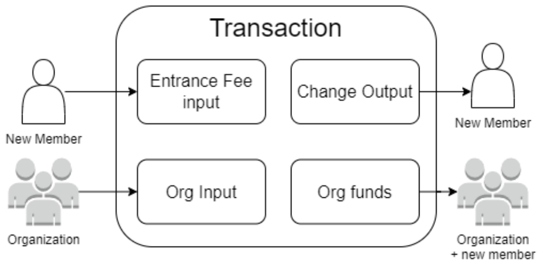


Figure 1: Bitcoin transaction for joining an organization when an entrance fee is needed.

because its key generation and signing algorithms only require two network rounds. The complexity of key generation and signing are both $\mathcal{O}(n^2)$ as every participant broadcasts messages to one another. Each participant FROST receives a key share after key generation, which can be used to sign data jointly. Compared to traditional collective management on Bitcoin [18], threshold signatures can scale to hundreds of participants. In addition, Bitcoin accounts that use threshold signatures are indistinguishable from standard accounts, allowing our design to be compatible with existing tools.

Threshold signatures have a significant drawback. Joining or leaving a group requires every participant to be online. A single participant could temporarily halt the joining process by choosing not to participate. To counteract this, a maximum inactivity duration can be specified, after which the offending member is kicked.

3.2 Decentralized Communication

To support our goal of decentralization, the system uses a peer-2-peer network for communication. Each participant will communicate with another directly to prevent reliance on any central actor. We rely on the IPv8 library for communication [19]. IPv8 allows for the construction of fully peer-2-peer networks through so-called *Communities*. *Communities* are peer-2-peer networks that contain application-specific functionality. Additionally, IPv8 employs hole-punching, which allows devices behind routers to establish peer-2-peer connections [20]. Hole-punching is required because routers use network address translation (NAT) to allow multiple devices to share an IPv4 address.

3.3 Governance

Our governance module sits on top of IPv8's networking and enables democratic decision-making for every action the group takes. The module is responsible for two things: handling join requests and handling proposals. Join requests allow individuals to request membership, while proposals enable members to suggest actions for the group to take.

Adding a new member to the group involves creating a new Bitcoin threshold account and transferring funds from the old account to the new account. The process has four messages, depicted in Figure 4. It consists of the following steps:

1. The new member creates and broadcasts a request to join the group with the *join request* message.

2. The group members receive the request and send back responses with their votes using the *join request response* message. The message contains a Boolean value that represents agreement or disagreement. The message also contains the number of individuals in the group, which decides how many messages to wait for.

3. The new member waits until they receive agreements from all group members. Otherwise, the process stops.

4. The new member starts the FROST key generation process by broadcasting a *key gen commitment* message.

5. Each participant broadcasts a *key gen commitment* message after receiving the *key gen commitment* message from the new member.

6. Each participant sends a key share message to every participant after receiving *key gen commitment* messages from all participants. Each key share is unique.

7. The key share messages received are combined into a key. The key is a cryptographic key that can be used together with the keys of other participants to spend funds from the Bitcoin account.

8. Once the key generation is done, a Bitcoin transaction is created to migrate funds from the old Bitcoin account to the new one. The transaction may also contain an entrance fee from the new member if required. The transaction is described in Figure 1

Proposals are Bitcoin transactions that the group can submit. Proposals allow the group to vote on which actions to take. The process has four messages, depicted in Figure 3. This process has the following steps:

1. A member creates a proposal and broadcasts it to the other members using a *sign request* message. The message contains the proposed Bitcoin transaction.

2. The other group members respond with their votes using a *sign request response* message. The message contains a Boolean value representing the vote.

3. The proposer waits for agreements from a majority of the group before starting the signing procedure. The process is canceled if not enough members respond with agreements during the timeout duration.

4. The proposer starts the signing process by broadcasting a *preprocess* message. The message will contain the total number of participants, so all participants know how many messages to wait for.

5. After receiving the *preprocess* message, the other participants will also broadcast *preprocess* messages. This time without the number of participants.

6. Each participant creates a signature and sends a *signature share* message to the proposer after receiving *preprocess* messages from every participant.

7. The final signature is created by combining the signature shares, and the transaction is completed by adding the signature. Afterward, the transaction is submitted to the Bitcoin network. Each Bitcoin input requires a signature. Thus, the signing process may need to run in parallel depending on how many inputs there are.

Figure 2 describes the process with 2 participants.

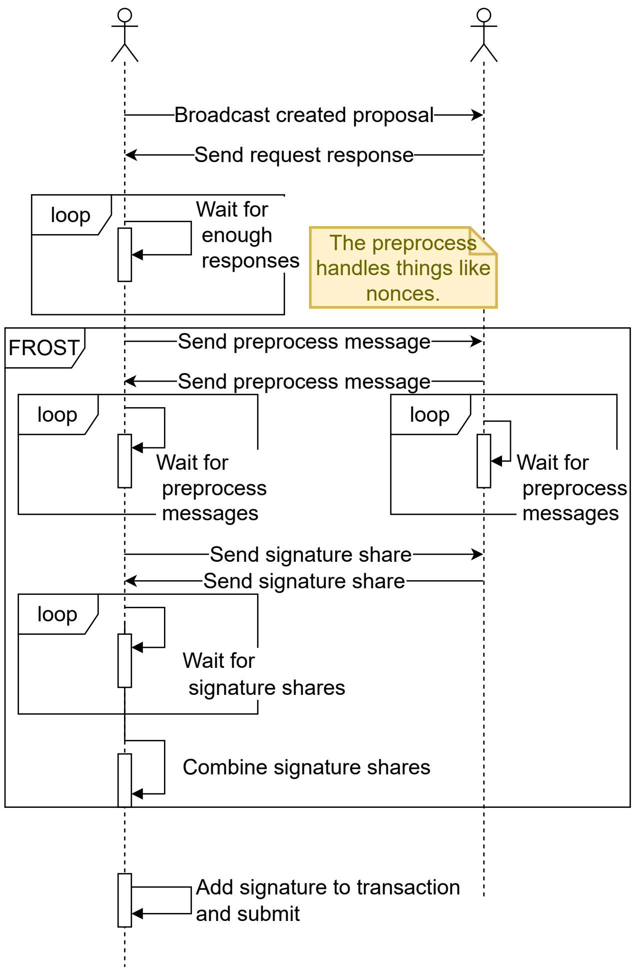

Broadcast created proposal
Send request response
loop
Wait for enough responses
The preprocess handles things like nonces.
FROST
Send preprocess message
Send preprocess message
loop
Wait for preprocess messages
loop
Wait for preprocess messages
Send signature share
Send signature share
loop
Wait for signature shares
Combine signature shares
Add signature to transaction and submit

Figure 2: Sequence diagram of the signing procedure. The diagram shows the procedure with two participants.

In both processes, the member proposing is responsible for signaling the start of the procedure. This significantly improves performance, as otherwise, each participant would need to broadcast readiness individually.

3.4 Identity

IPv8's identity mechanism is used to identify participants in the system. The identities are cryptographic keys, allowing participants to prove their identity by signing a message. The identity mechanism may be insufficient depending on whether the organization has anonymous members. In this case, the organization could be overtaken by a Sybil attack[21]. To prevent this, a Self-Sovereign Identity mechanism could be used[22]. Using Self-Sovereign Identity, participants could identify themselves without giving out privileged information.

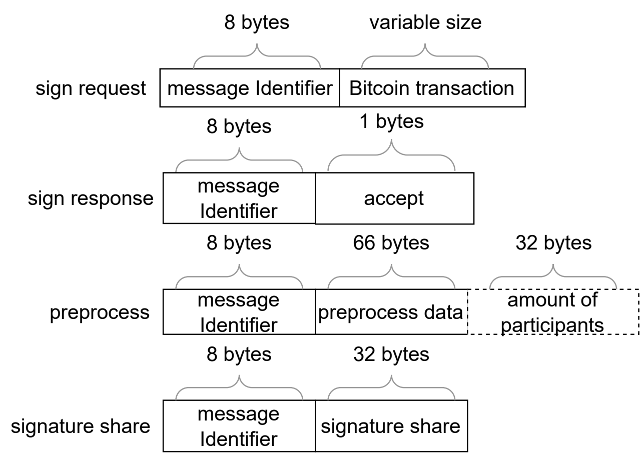


Figure 3: Breakdown of the signing protocol messages. Dashed borders represent a field that is not always required.

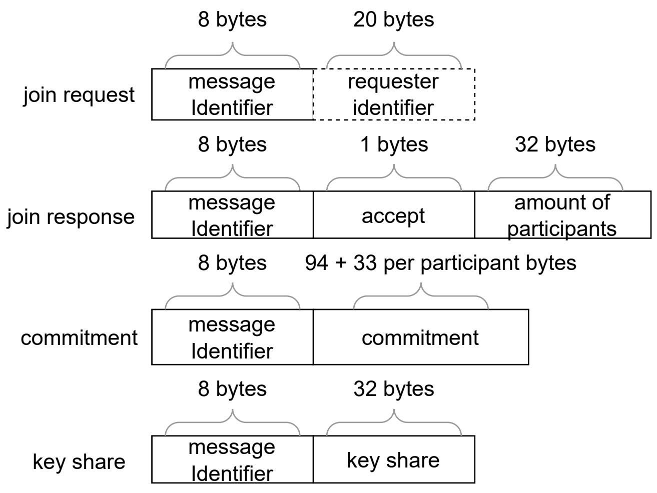


Figure 4: Breakdown of the key generation protocol messages. Dashed borders represent a field that is not always required.

4 Implementation

We have created an Android application that partially implements our design. Our work expands the protocol stack under development for several years at the Delft University of Technology, which contains experimental support for Bitcoin, BitTorrent streaming, trust framework, self-sovereign identity, offline digital Euros, and decentralized learning [23]–[28]. Using the application, users can join a group, create a proposal, and vote on proposals. The open-source application is publicly available on GitHub[1].

4.1 Shared Bitcoin Wallet

The Android application contains a personal wallet and the group wallet. The personal wallet is only used for testing purposes. We use the BitcoinJ open-source library [29] for Bitcoin support. BitcoinJ tracks the Bitcoins in the group wallet to allow participants to create proposals easily. BitcoinJ stores its data in an SQLite database.

To support threshold signatures, we used an audited open-source Rust library [30]. We created a wrapper around the library and then exposed the wrapper to Android via Java's native interface [31]. Our wrapper modifies the signature created by the library to be compatible with Bitcoin. After the key generation procedure, each participant receives a key share. The key share is stored in a database for persistence.

Bitcoin inputs represent spendable Bitcoin. One or multiple inputs must be spent to create a transaction. The Android application only supports Bitcoin transactions with one input, as Bitcoin transactions require signatures for each input used in a transaction. Additionally, only payment transactions are supported.

[1]https://github.com/rahimklaber/trustchain-superapp/tree/frost_dao/frostdao

4.2 Joining an Organization

After clicking the join button on the screen depicted in Figure 5, The application will send a join request message and waits for responses. The joining procedure starts if enough responses are received within the timeout duration. Every participant will join the same DAO since DAO discovery is not implemented. Once the process is complete, it will be possible to view details of the group account, create proposals, and reject or accept proposals. Our proof-of-principle prototype lacks features such as entrance fees and migrating funds when a new user joins.

4.3 Spending funds

The DAO can make investments and spend funds by creating and accepting proposals. Any member of the DAO can create a new proposal. Other members then have the choice of accepting or declining the proposal. If enough members accept, a signature is created using FROST, and a Bitcoin transaction is submitted to the network. This process is depicted using two devices in figures 5 through 10. We used our own test Bitcoin network to create fake Bitcoins. The process is as follows:

- Figure 5 - The DAO wallet is funded with Bitcoins, shown on the main screen. In the Background, the app listens for new transactions relating to the DAO account and uses these transactions to calculate the Balance.
- Figure 6 - The details for the proposal are filled in. The top field contains the destination address, and the bottom field contains the amount of Bitcoin in its smallest unit.
- Figure 7 - After clicking propose on the previous screen, the proposal is broadcast to the DAO and is shown on the proposals screen. The member who created the proposal can see details of the proposal and can the transaction hash.

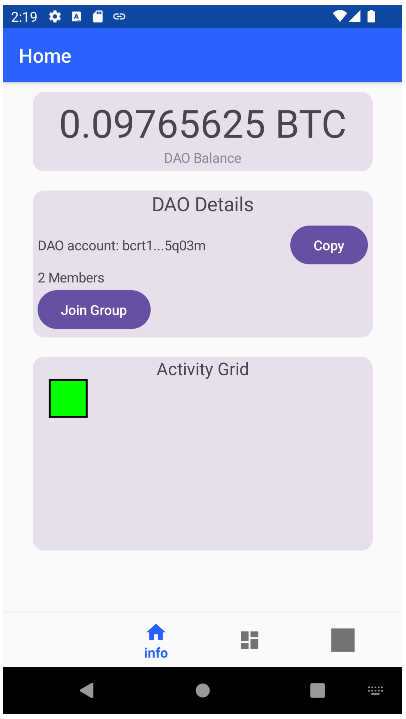


Figure 5: The DAO account has been funded

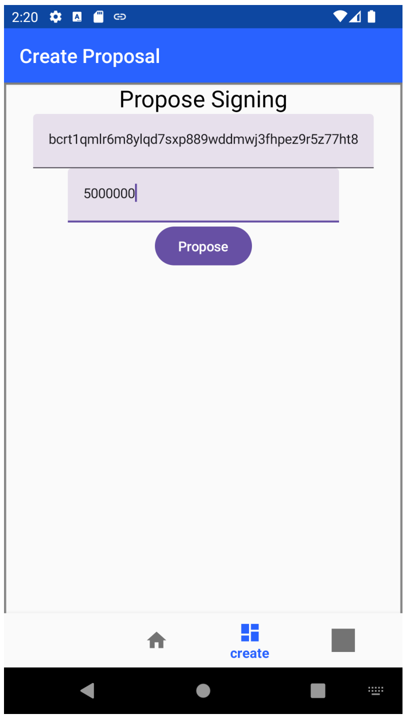


Figure 6: The details to create a proposal are filled.

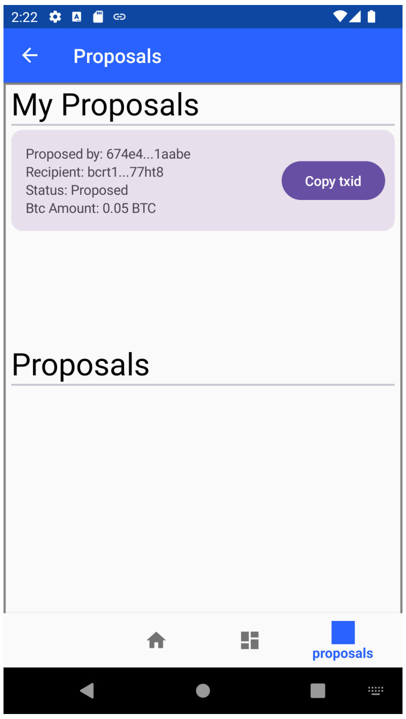


Figure 7: The proposal has been created.

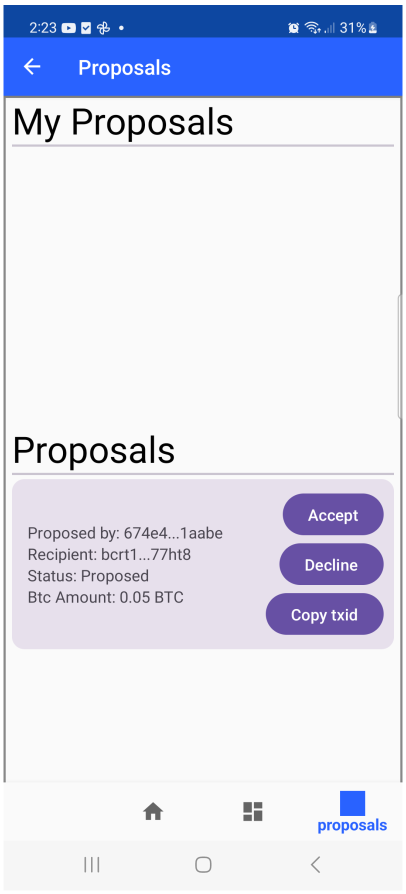


Figure 8: The other device has received the proposal.

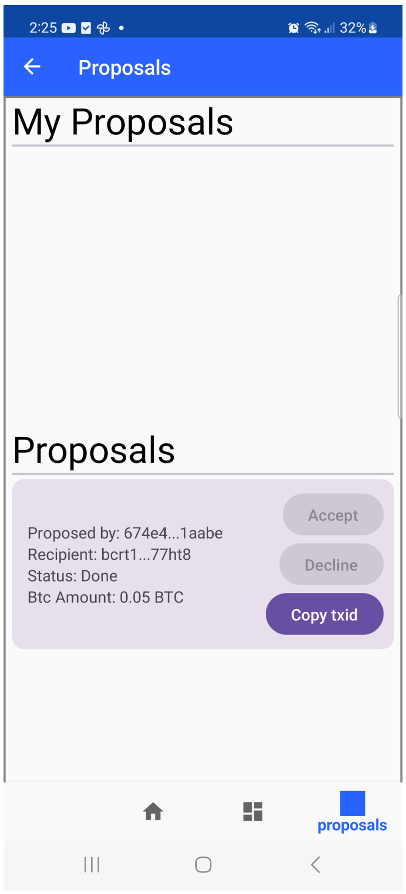


Figure 9: The proposal has been accepted.

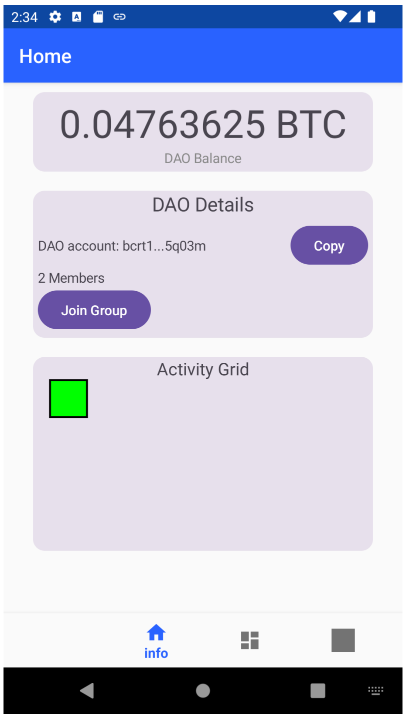
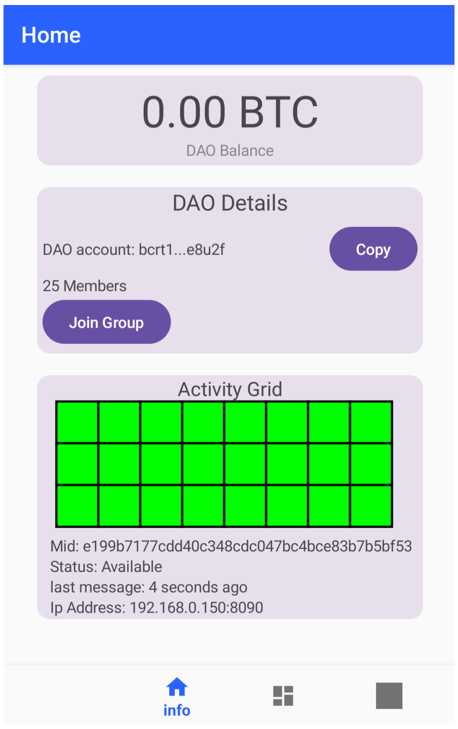


Figure 10: The Bitcoins have been spent


Home

0.00 BTC

DAO Balance

DAO Details

DAO account: bcrt1...e8u2f

Copy

25 Members

Join Group

Activity Grid

Mid: e199b7177cdd40c348cdc047bc4bce83b7b5bf53
Status: Available
last message: 4 seconds ago
Ip Address: 192.168.0.150:8090

Figure 11: Figures depicting activity grid of a DAO with 25 members. Each square represents a member of a DAO, and the color represents their status. Pressing a square reveals information about that member. For testing purposes, the screenshots are obtained from a real Android device, while the other DAO members are running on PC hardware.

- Figure 8 - The members who did not create the proposal can accept or reject the proposal.
- Figure 9 - Once the proposal is accepted, the accept and reject buttons are disabled and the signing process starts. After signing, the transaction is submitted, and the proposal status is updated.
- Figure 10 - The Balance is updated after the Bitcoin network confirms the transaction.

4.4 Activity Grid

The application provides an overview of members' statuses through the activity grid. This is a grid with a square for each member in the DAO. Each square has one of 2 colors depending on the member's status. The square is green if the member is active, which we define as having received a message from them within one minute. Otherwise, the square is red, denoting inactivity. Users who have not yet joined the DAO have a yellow square. Clicking on a square reveals information about that peer: their identifier, status, when a message was last received from them, and their IP address. An example of the activity grid is given in Figure 11.

4.5 Quality Assurance

We used unit and integration tests to ensure the code is bug-free and the code coverage is shown in Table 1. The core

Class / Package	Line coverage	Lines of code
FrostManager	93%	956
SchnorrAgent	94%	216
FrostCommunity	65%	141
FrostViewModel	0%	377
ui	0%	980

Table 1: Code coverage of the FROSTDAO application.

of the application, which consists of FrostManager, SchnorrAgent, and FrostCommunity, has been extensively tested. However, harder-to-test code, like the UI and Bitcoin code, is not well-tested.

We used unit tests to ensure our code had no significant flaws. Specifically, we tested key generation and signing while mocking communication. Integration tests were used to test key generation and signing without mocked communication. The integration tests discovered many bugs that were caused by race conditions. We fixed many of the bugs, but some still occasionally occur. The bugs rarely occur and we are unsure if IPv8 or our code is responsible for the bugs.

We also manually tested the Android application to test the entire application, particularly the Bitcoin integration. To do this, we created our own private Bitcoin network, enabling us to create testing Bitcoins and instantly confirm transactions. We used two mobile devices for this.

4.6 Challenges

Developing any distributed system is challenging, especially fully peer-2-peer systems. During development, we encountered numerous problems and challenges. This includes challenges relating to communication, Bitcoin, and reliability.

IPv8 relies on UDP under the hood, which is not reliable. To address this, we added acknowledgments and timeouts on top of IPV8. Determining the correct timeout duration and the amount of retransmission is challenging, especially for unreliable networking.

The size of the *key commitment* message (see Figure 4) during key generation scale with the number of participants. If we want the UDP packets not to be dropped, we need to limit their size to around 1400 Bytes [32], which some messages do not fit into. Therefore, we use IPv8's EVA protocol, which splits data into multiple packets. Using EVA to send messages results in high latency. Additionally, EVA transfers have a high failure rate.

The system relies on being able to send messages to all members. However, IPv8 is not meant to create fully connected peer-2-peer networks. Each peer in a community will keep track of some peers by sending periodic pings. This is not a problem in smaller networks (30 members) but becomes a problem in larger networks. This can be somewhat mitigated by changing IPv8's configuration.

The signing process requires that every participant has an up-to-date view of the Bitcoin network. However, It often happened that some participants lagged behind. This resulted in the process failing.

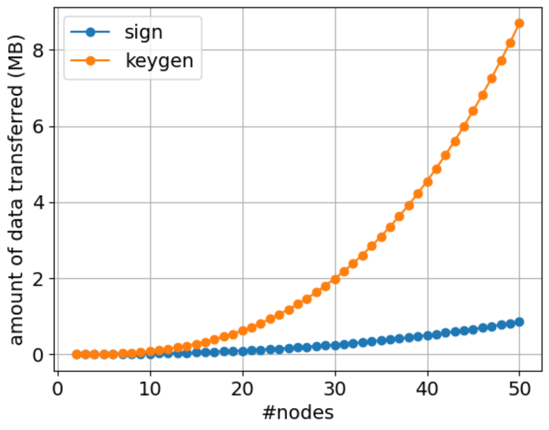


Figure 12: Amount of data in Kilobytes sent during key generation and Signing

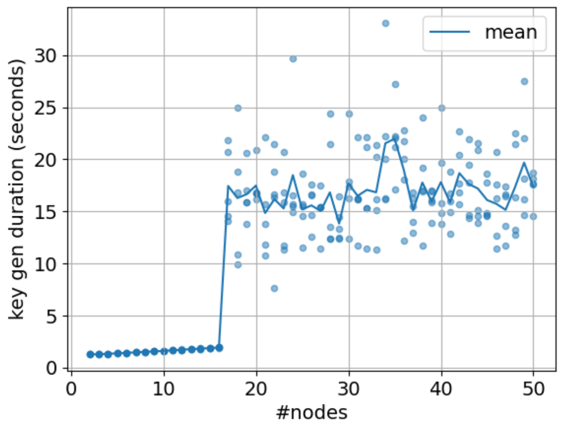


Figure 13: Duration of the key generation running on the IPV8 stack. The plot contains the result of 5 runs. The variance before 17 nodes is not significant.

5 Evaluation

In this section, we evaluate the performance of our system. We are interested in the performance of signing and key generation, as these are the most expensive parts of our system. Using a PC, we measure the performance in 3 ways. First, we measure the time required for key generation and signing. Second, we measure the amount of data transferred during key generation and signing. Third, we introduce artificial delays to investigate performance in a more real-life scenario. We also measure the performance using physical Android devices. We have a limited number of physical devices, so we are conducting an experiment to see how the system performs on Android and to ensure that the results are consistent with those of the PC experiment.

5.1 PC Experiments

We ran experiments on a Windows 10 PC with 32GB of RAM and a Ryzen 7 3700x CPU with 8 cores and 16 threads. We modified the code responsible for communication and signing to work in a Desktop environment with a Java virtual machine. This included compiling the native code to work on Windows. Our experiments were run in one application that created individual nodes representing organization members. Each node is an IPv8 node that runs the entire IPv8 stack. However, network latency is not a factor since all nodes are on the same PC. We ran the experiments with up to 50 nodes. Attempting to increase the node count further resulted in peer discovery failing. We modified the default IPv8 *maxPeer* configuration to allow each peer to connect directly to all other peers. Each experiment was run 5 times.

Figure 12 shows the amount of data sent during key generation and signing. While both key generation and signing have a quadratic message complexity, we notice that key generation scales much worse than signing. This is because the size of messages sent during key generation depends on the number of participants (see Figure 4), causing the amount of data transferred to have cubic complexity. When running key generation with 50 nodes, each node sent and received around 160KB of data. While the amount is not a lot, it will quickly grow with the number of participants and will present issues for mobile devices with limited bandwidth. In contrast to key generation, the signing protocol requires significantly less data to be sent. This is expected as each signing operation requires constant data per participant. These results show that key generation is significantly more expensive than signing.

Figure 13 shows the duration of the key generation protocol. Up to 16 nodes, the procedure has a low duration that increases a small amount when the number of nodes increases. After 16 nodes, the duration and variability increase dramatically. At this point, the size of messages sent during key generation is no longer small enough such that the UDP packets are delivered reliably. The dramatic increase in duration is due to EVA, IPv8's TFTP protocol for sending larger amounts of data. This protocol splits the data into chunks, sends each chunk via UDP, and uses acknowledgments to ensure that each chunk is delivered. EVA does not send the data immediately and instead schedules transfers in the future, which results in a large spike in duration. The large variability is due to the EVA protocol failing and needing to retransmit data and due to the protocol's scheduler. The signing protocol, shown in Figure 14, is much quicker than key generation, as the messages all fit inside UDP packets. In practice, signing will scale much better since only a majority of the organization needs to participate. Thus, in an organization with 50 members, only 26 need to participate.

The impact of the EVA protocol on the key generation duration was surprising. To understand precisely what the cause was, we investigated further. After a thorough review of EVA, we found the following potential causes:

- Each peer can only send or receive one transfer from another at a time.[2]
- Each peer can only be involved in 9 transfers simultaneously. This includes sending or receiving.[3] This is

[2]github.com/Tribler/kotlin-ipv8/.../eva/EVAProtocol.kt#L259

[3]github.com/Tribler/kotlin-ipv8/.../eva/EVAProtocol.kt#L296C17-L296C61

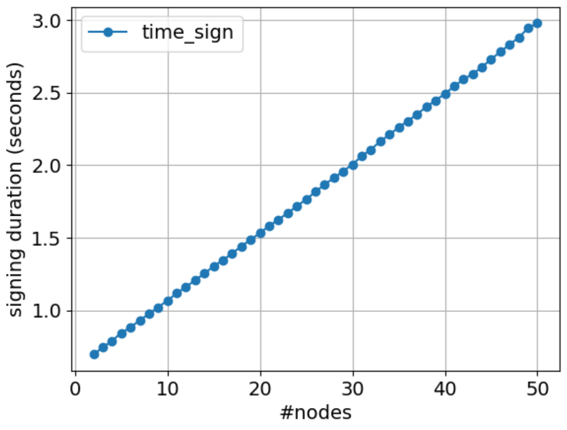


Figure 14: Average duration of the signing protocol running on the IPV8 stack. The variance is not significant.

specified by the *maxSimultaneousTransfers* variable.[4]

- Attempting to initiate a new transfer when not possible due to the above causes will result in the transfer being queued. Every 5 seconds, the transfer in front of the queue is attempted. [5] This is specified by the *scheduledSendInterval* variable.[6]
- EVA transfers have a timeout of 20 seconds. This means that if a transfer failure isn't handled, then one of the 9 transfer slots will be occupied for 20 seconds. [7] The timeout is specified by the *timeoutInterval* variable.[8]
- Our serialization is inefficient, resulting in a lot of overhead for large messages.

We improved EVA's performance with aggressive settings. We reduced the timeout duration from 20 seconds to 2 seconds and the scheduler interval from 5 seconds to 500 milliseconds. We also improved the message serialization such that EVA is only required starting from 37 participants instead of 16. We attempted to increase the number of concurrent transfers such that they were executed immediately, but this caused EVA to sometimes fail.

Figure 15 shows the duration of key generation with our improvements. Compared to Figure 13, the improvements have resulted in a much more natural progression in duration. The mean of the results is much smaller, suggesting that we have significantly reduced the overhead of EVA. However, the overhead is still considerable, especially in the worse case.

To understand the effect of latency on performance, we did some experiments with different latencies. Shown in Figure 16 is the added duration of the protocols with 100 and 200 milliseconds. We did experiments with up to 36 nodes to prevent EVA from affecting the results. In theory, the graphs should be horizontal lines. This is because each protocol has a constant number of broadcasts, which should

[4]github.com/Tribler/kotlin-ipv8/.../eva/EVAProtocol.kt#L35
[5]github.com/Tribler/kotlin-ipv8/.../eva/EVAProtocol.kt#L74
[6]github.com/Tribler/kotlin-ipv8/.../eva/EVAProtocol.kt#L29
[7]github.com/Tribler/kotlin-ipv8/.../eva/EVAProtocol.kt#L797
[8]github.com/Tribler/kotlin-ipv8/.../eva/EVAProtocol.kt#L32

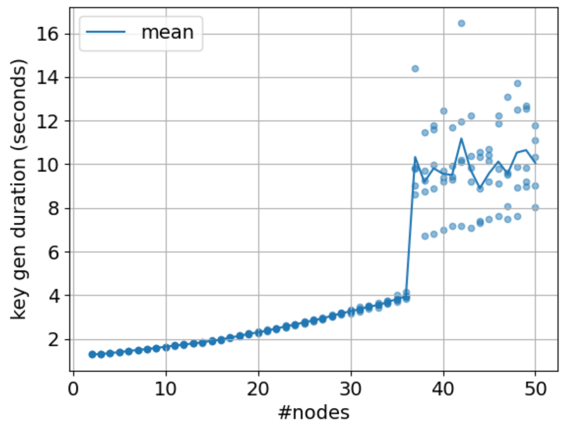


Figure 15: Duration of the key generation running on the improved IPV8 stack using our EVA and serialization improvements. The plot contains the result of 5 runs. The variance before 37 nodes is not significant.

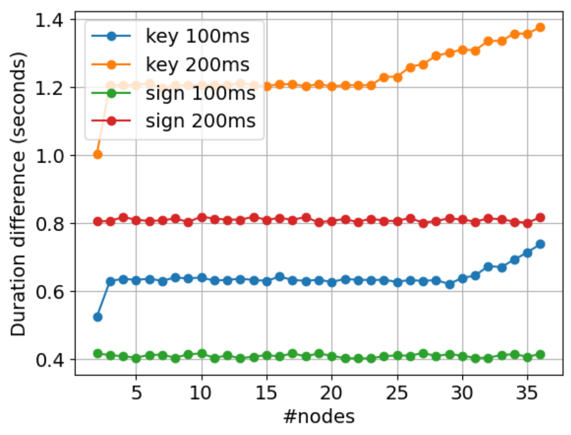


Figure 16: Average added duration of protocols with 100 and 200 milliseconds delay.

add a constant duration. This is the case for signing but not key generation. Key generation with 2 participants requires one less broadcast, which explains the outlier value for 2 participants. The rise at the end is harder to explain. We are unsure of the cause, but it is likely an issue with our code. More experiments should be run to see if the trend continues.

5.2 Android Experiments

We conducted experiments to determine how real using real devices impacts our system. We used 4 Android devices on the same network and repeated the experiments 10 times. We only measured key generation due to the complexity of signing on Android (see Figure 5). Nevertheless, our insights should apply to signing as well. The results are shown in Figure 17.

Except for key generation with 2 devices, the average durations are similar to that of the PC experiment. Interestingly, key generation with 2 devices has a higher duration than with 3 or 4 devices. Since the app was just opened and

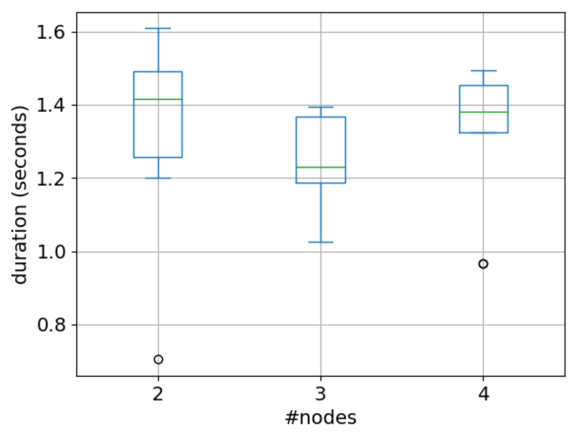


Figure 17: Duration of key generation on Android devices.

no previous actions were done, we can attribute the higher duration to code being initialized lazily. When using more than 2 devices, the code is already initialized. We expected all results to be higher than the PC experiments due to the communication delay, but some outliers were significantly lower. This is likely caused by the overhead of running 50 nodes on a single PC and minor differences between the PC and Android versions of the code.

When analyzing our results, we realized that the performance depends on the slowest participant. Key generation and signing have 2 rounds, with the second dependent on the first. In the worst case, a participant with an extremely slow network connection could significantly reduce performance. Although, this is more of a problem for key generation with its large messages and since signing can be optimized to 1 round with some precomputation. While the bottleneck seems obvious, we did not realize it because our PC experiments were so controlled.

6 Discussion and Future work

The results indicate that the performance of our system is practical for real-world use for the tested parameters; signing and key generation can be done in under 20 seconds. However, our results hint at limitations due to the performance of key generation. While EVA has significantly affected performance, this can be mitigated through engineering. More concerning is the scalability of key generation itself. The data suggests that the amount of data transferred during key generation scales exponentially, which would quickly lead to poor performance, especially on mobile devices. Future work could seek to understand how far this system can scale while still being practical to use.

7 Conclusion

DAOs enable decentralized and transparent collective wealth, allowing groups of individuals to pool their funds for economic activity. DAOs rely on complex smart contracts to function. Despite being the most popular Blockchain, existing technology cannot support DAOs on Bitcoin. In this thesis, we designed a system allowing a group of people to manage their wealth collectively using Bitcoin. To determine if our system is practical, we implemented it as an Android application. The Android Application uses a peer-2-peer network and does not rely on any central party, allowing the application to be used by anyone. The Android application uses the FROST threshold-signature scheme, allowing individuals to jointly control a Bitcoin account without relying on complex smart contracts. Compared to multi-signature implemented using Bitcoin script, threshold signatures do not increase the size of transactions based on the number of participants. Our experiments show that this technique is practical. Both key generation and signing take less than a minute for less than 50 participants. Increases in latency only add constant durations to the signing and key generation protocols. Our experiments also show that key generation performance can be improved by a significant amount, as the performance is limited by IPv8.

References

[1] V. V. Acharya and M. Richardson, "Causes of the financial crisis," *Critical Review*, vol. 21, no. 2-3, pp. 195–210, 2009. DOI: 10 . 1080 / 08913810902952903. eprint: https : / / doi . org / 10.1080/08913810902952903. [Online]. Available: https://doi.org/10.1080/08913810902952903.

[2] P. K. Ozili, "Causes and consequences of the 2023 banking crisis," *Available at SSRN 4407221*, 2023.

[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, 2008.

[4] S. Lo and J. C. Wang, "Bitcoin as money?," 2014.

[5] K. Hong, "Bitcoin as an alternative investment vehicle," *Information Technology and Management*, vol. 18, pp. 265–275, 2017.

[6] D. Khan, L. T. Jung, and M. A. Hashmani, "Systematic literature review of challenges in blockchain scalability," *Applied Sciences*, vol. 11, no. 20, p. 9372, 2021.

[7] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[8] S. Wang, W. Ding, J. Li, Y. Yuan, L. Ouyang, and F.-Y. Wang, "Decentralized autonomous organizations: Concept, model, and applications," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 5, pp. 870–878, 2019.

[9] S. Delgado-Segura, C. Pérez-Sola, G. Navarro-Arribas, and J. Herrera-Joancomartı, "Analysis of the bitcoin utxo set," in *Financial Cryptography and Data Security: FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers 22*, Springer, 2019, pp. 78–91.

[10] R. Chotkan, J. Decouchant, and J. Pouwelse, "Unstoppable daos for web3 disruption," in *Proceedings of the 3rd International Workshop on Distributed Infrastructure for the Common Good*, ser. DICG '22, Quebec, Quebec City, Canada: Association for Computing Machinery, 2022, pp. 37–42, ISBN: 9781450399289. DOI: 10.1145/3565383.3566112. [Online]. Available: https : / / doi . org / 10 . 1145 / 3565383.3566112.

[11] S. Faraj, S. L. Jarvenpaa, and A. Majchrzak, "Knowledge collaboration in online communities," *Organization Science*, vol. 22, no. 5, pp. 1224–1239, 2011, ISSN: 10477039, 15265455. [Online]. Available: http://www.jstor.org/stable/41303115 (visited on 07/06/2023).

[12] R. Kouzes, J. Myers, and W. Wulf, "Collaboratories: Doing science on the internet," *Computer*, vol. 29, no. 8, pp. 40–46, 1996. DOI: 10.1109/2.532044.

[13] D. M. Wilkinson and B. A. Huberman, "Cooperation and quality in wikipedia," in *Proceedings of the 2007 international symposium on Wikis*, 2007, pp. 157–164.

[14] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security," in *Applied Cryptography and Network Security: 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings 14*, Springer, 2016, pp. 156–174.

[15] R. Fritsch, M. Müller, and R. Wattenhofer, "Analyzing voting power in decentralized governance: Who controls daos?" *arXiv preprint arXiv:2204.01176*, 2022.

[16] X. Sun, S. Lin, V. Sjöberg, and J. Jie, "How to exploit a defi project," in *Financial Cryptography and Data Security. FC 2021 International Workshops*, M. Bernhard, A. Bracciali, L. Gudgeon, *et al.*, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, pp. 162–167, ISBN: 978-3-662-63958-0.

[17] C. Komlo and I. Goldberg, "Frost: Flexible round-optimized schnorr threshold signatures," in *Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27*, Springer, 2021, pp. 34–65.

[18] S. Goldfeder, R. Gennaro, H. Kalodner, *et al.*, "Securing bitcoin wallets via a new dsa/ecdsa threshold signature scheme," in *et al.* 2015.

[19] M. Skála, "Technology stack for decentralized mobile services," 2020.

[20] G. Halkes and J. Pouwelse, "Udp nat and firewall puncturing in the wild," in *10th IFIP Networking Conference (NETWORKING)*, Springer, 2011, pp. 1–12.

[21] J. R. Douceur, "The sybil attack," in *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260, ISBN: 978-3-540-45748-0.

[22] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity," *Computer Science Review*, vol. 30, pp. 80–86, 2018.

[23] M. Skála, "Technology stack for decentralized mobile services," 2020.

[24] T. Wissel, "Fairness and freedom for artists: Towards a robot economy for the music industry," 2021.

[25] J. Bambacht and J. Pouwelse, "Web3: A decentralized societal infrastructure for identity, trust, money, and data," *arXiv preprint arXiv:2203.00398*, 2022.

[26] B. Planje, "First deployed dao with true full decentralisation," 2023.

[27] R. Madhwal and J. Pouwelse, "Web3recommend: Decentralised recommendations with trust and relevance," *arXiv preprint arXiv:2307.01411*, 2023.

[28] T. Werthenbach and J. Pouwelse, "Towards sybil resilience in decentralized learning," *arXiv preprint arXiv:2306.15044*, 2023.

[29] P. Xiao, "Java programming for blockchain applications," in *Practical Java Programming for IoT, AI, and Blockchain*. 2019, pp. 347–388. DOI: 10.1002/9781119560050.ch10.

[30] L. Parker, *Modular frost*. [Online]. Available: https://github.com/serai-dex/serai/tree/develop/crypto/frost (visited on 07/06/2023).

[31] S. Liang, *The Java native interface: programmer's guide and specification*. Addison-Wesley Professional, 1999.

[32] C. Kaufman, R. Perlman, and B. Sommerfeld, "Dos protection for udp-based protocols," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, ser. CCS '03, Washington D.C., USA: Association for Computing Machinery, 2003, pp. 2–7, ISBN: 1581137389. DOI: 10.1145/948109.948113. [Online]. Available: https://doi.org/10.1145/948109.948113.