Asynchronous and Decentral Group Management in Messengers with Delegates Proof of Stake

And Hell [1]

Abstract: Mobile messaging applications are used widely for group communication using group chats. Most messenger platforms rely on their centralized infrastructure to maintain the group states. This can imply privacy issues and allow potential misuse by the messenger providers. To resolve this privacy implications, a decentral approach can be implemented. The decentral protocol presented in this work is based on the Delegated Proof of Stake consensus protocol and uses a blockchain to store the groups state. The main focus of this work is the optimization of the protocol to be able to deal with the asynchronous environment of mobile applications.

Keywords: Blockchain; DPoS; Groupchats

1 Introduction

Group Chats are an essential feature of mobile messenger applications, such as WhatsApp, Signal and Telegram. In order to add or remove members from a group, management is needed. Most messenger systems rely on centralized management by storing the groups state gr on the system server. [2]

$$gr = (id, \mathcal{M}, \mathcal{M}^*, info) \tag{1}$$

Table 1 defines the components of the four tuple introduced in equation 1.

id	:= Unique group identifier
$\mathcal{M}$	:= Set of members/users
$\mathcal{M}^*$	:= Set of admins with $\mathcal{M}^* \subseteq \mathcal{M}$
$info$	:= Meta information (e. G. name, icon, description, ...)

Tab. 1: Group state

Signal recently published a paper that introduced an end-to-end encrypted, centralized group management [CPZ19], ditching there previously used decentral system [Ma]. Other systems store the group information unencrypted, as they distribute group messages from

[1] Germany hello@andhell.de

[2] The notation of equation 1 was introduced by Rösler et al. in their analysis of group messaging protocols [RMS17].

the server [Te; Wh]. In Signal group messages are processed like direct messages, such that the server does not need to know the groups members.

The unencrypted, central storage of group information violates the users privacy. Even in Signals encrypted handling of group information, the existence of a group is disclosed to the service and may allow the server to return incorrect information about the group [CPZ19].

The goal of this work is to providing a decentral group management systems, that can be used in the asynchronous environment of mobile messenger, where clients are not always conneted and the users only open the messenger from time to time.

1.1 Delegated Proof of Stake

Delegated Proof of Stake (DPoS) is a consensus algorithm used for the blockchains of EOS and BitShare [Bia; EO]. Compared to Proof of Work systems like BitCoin [Na09], DPoS does not rely on a competition to find hashes which requires a high computational effort. Instead all stakeholders (e.G. the coin holders) can elect delegates, that then can build and distribute new blocks. These blocks contain transactions and votes and are authenticated by the signature of a currently elected delegates. Every x blocks, the votes are recounted and the set of delegates is updated if needed [Bib].

The order in which the delegates sign new blocks is defined by a fixed or randomized schedule and all delegates that share a valid block in their assigned time slot are rewarded with new tokens [EO].

1.2 Related Work

In [He19] a group management system based on the DPoS protocol was presented. Instead of relying on a central service to maintain and distribute the state gr, group members share messages between each other to manage the group.

In this work, several requirements for a group management system where defined.

Same State All members m in a group with state gr uses the same local state gr_m.

$$\forall m(gr_m = gr) \qquad \text{for } m \in \mathcal{M}$$

Confidentiality by Group Management Only admins $m \in \mathcal{M}^*$ should be able to update the group state gr (e. G. add or remove members).

Privacy Only the members $m \in \mathcal{M}$ of a group should know that the group exists.

Each group is defined by its own blockchain that is distributed between the members. The members of the group elect delegates, which are able to sign suggestions. Suggestions, such as *add* or *remove* member and *info*, can be shared by all members and are confirmed if a elected delegate shares a new block, including the suggestions. If no delegate builds such a block, the suggestion is ignored and not applied to the group. Suggestions can be seen as the transactions of traditional DPoS systems, beside being optional to confirm.

[He19] describes several shortcomings regarding the chosen voting process to elect delegates and the handling of forks. Members can start *Voting Windows* in which every member can share votes if they get online during this time frame. This implies that members which did not get online are excluded from the vote, as a synchronous component is introduced in the asynchronous environment of messenger applications. After collecting all votes, the members need to confirm to each other, that they computed the same election result. This introduces a second step and increases the protocols complexity. Forks are resolved by dropping the branch that does not get continued, this leads to information loss and violates the *same state* requirement for the time the fork exists.

The presented protocol also does not take delivery issues like delayed or lost messages into account. As these are expected in an asynchronous mobile environment, the group management protocol needs to be resilient against delivery issues.

2 Group Management with DPoS

The following section introduces an updated version of the protocol with focus on the new voting and forking system.

2.1 Notation

The table 2 shows some additional notations, which will be used in the following section.

$[x_1, ..., x_n]$	List of the items x_1 to x_n
$\mathcal{B}_g r$	The Blockchain of the group gr
B_{-1}	The latest block of a Blockchain $\mathcal{B}_g r$
$hash(x) \rightarrow$ byte[64]	Hash for x
$sig(u_{id}, x) \rightarrow$ byte[]	Signature for x from signer u_{id}
$verify(u_{id}, sig, data) \rightarrow$ boolean	Verifies a Signature
$MerkleRoot([x_1, ..., x_n]) \rightarrow$ byte[]	Merkle root of a Merkle tree over the list $[x_1, ... , x_n]$
$MerklePath(x_i, [x_1, ..., x_n]) \rightarrow$ [byte[]]	Merkle Verification Path to the leaf x_i with $1 \leq i \leq n$ of a Merkle tree over the list [...]
$verify(x_i, mvp) \rightarrow$ boolean	Verifies a Merkle path

Tab. 2: Additional Notation

More on Merkle trees can be found in [Me88].

2.2 Protocol

The blockchain $\mathcal{B}_{gr}$ that is distributed between the members $\mathcal{M}_{gr}$ describes the current state of a group $gr = (id, \mathcal{M}_{gr}, \mathcal{M}^*_{gr}, info)$. If a new block B_S with a suggestion S is added to $\mathcal{B}_{gr}$, the group state is updated according to the action a of S ($gr' = Update(gr, a)$).

$$\mathcal{B}_{gr} + B_S \rightarrow gr' = Update(gr, a) \tag{2}$$

2.2.1 Suggestion

A suggestion S can be shared by any member $m \in \mathcal{M}$ at any time.

$$S = (u_{id}, action, value, b_{id}, b_{hash}, sig) \tag{3}$$

Table 3 defines the components of the suggestion tuple introduced in equation 3.

	Definition	**Comment**
u_{id}	$:= m \in \mathcal{M}$	Sender
action	$:= add \parallel remove \parallel info$	The action to perform
value	$:= u_{id} \parallel info$	The user or new *info*
b_{id}, b_{hash}	:= Block Reference	Block id and the blocks hash
sig	$:= sig(S \setminus sig)$	Signature

Tab. 3: Suggestion

Each suggestion includes a block reference (b_{id}, b_{hash}), which links it to a specific block in $\mathcal{B}_{gr}$. This prevents the suggestions to be included in a different blockchain and allows the members to check the assumptions the sender makes about the current groups state gr. This mechanism is known as Transactions as Proof of Stake (TaPOS)[EO].

If a suggestion is applied, its action updated the group according to table 4.

(action, value)	**Update**
(add, u_{id})	$\mathcal{M}' = \mathcal{M} \bigcup \{u_{id}\}$
$(remove, u_{id})$	$\mathcal{M}' = \mathcal{M} \setminus \{u_{id}\}$
$(info, i)$	$info' = i$

Tab. 4: Suggestion actions

A suggestions S is valid for the blockchain $\mathcal{B}$ with the latest block B_{-1} and can be confirmed if the conditions in table 5 hold.

$a = remove \wedge value \in \mathcal{M}$		Only remove members
$a = add \wedge value \notin \mathcal{M}$		Only add users that are not already a member
$\exists b(hash(b) = b_{hash} \wedge b.b_{id} = b_{id})$	$b \in \mathcal{B}$	Referenced Block is in $\mathcal{B}$
$b_{id} \geq B_{-1} - n$		Referenced block is not older than n blocks
$verify(u_{id}, sig, S \setminus sig)$		Valid Signature

Tab. 5: Valid Suggestion

2.2.2 Block

A suggestion block B_S can be shared by the delegates $m \in \mathcal{M}^*$. As soon as a delegate gets online and received a suggestion S he likes to confirm, he can sign and share a block, as long as S is not already confirmed by a block from another delegate. Unlike a traditional DPoS systems, a specific order does not need to be followed by the delegates. This allows the group management to work in the asynchronous messenger environment as it is not requierd for them to be online at a specific time.

$$B_S = (b_{id}, pbh, S, sig, u_{id}, MVP, VMR, nVMR, votes) \tag{4}$$

Table 6 defines the components of B_S introduced in equation 4.

	Definition	**Comment**
b_{id}	$:= B_{-1}.b_{id} + 1$	Latest $b_{id} + 1$
pbh	$:= hash(B_{-1})$	Hash of the previous/latest block
S	:= Suggestion	Confirmed suggestion
u_{id}	$:= m \in \mathcal{M}^*$	Delegate / Signer
MVP	$:= MerklePath(u_{id}, \mathcal{M}^*)$	Delegate proof
VMR	$:= MerkleRoot(\mathcal{M}^*)$	Delegate proof
$nVMR$	$:= MerkleRoot(\mathcal{M}^{*'})$	Merkle root of $\mathcal{M}^*$ with the new *votes*
$votes$	:= [vote]	List of new votes
sig	$:= sig(u_{id}, Block \setminus sig)$	Block signature

Tab. 6: Block

Delegate Proof When building and sharing a new block, the block signer u_{id} needs to proof that he is indeed a currently elected delegate ($u_{id} \in \mathcal{M}^*$). To do so he creates a *delegate proof* with the Merkle root *VRM* and a Merkle verification path *MVP* from the Merkle tree over $\mathcal{M}^*$. The *VRM* can be used to verify that the block signer used the correct set of delegates and by providing a valid *MVP* from his u_{id} to the *VRM*, he can proof that $u_{id} \in \mathcal{M}^*$.

The *nVRM* is used to update $\mathcal{M}^*$ if the new votes change the election result. This process is explained in section 2.3.

Confirmation Blocks S can also be empty. Such a block is called *Confirmation block*, as it confirms the current group state gr without updating it. These blocks are shared by the delegate that first gets online after a time x since the last block and no new suggestions are available for confirmation, to add new votes to the blockchain.

Block Verification After a block B is received, each member validates the block in relation to the latest/previous block $B_{-1} \in \mathcal{B}$. The block is accepted if the conditions from table 7 are hold:

$b_{id} = B_{-1}.b_{id} + 1$	Latest $b_{id} + 1$
$pbh = hash(B_{-1})$	Latest hash matches
$Valid(S)$	Valid suggestion
$verify(u_{id}, sig, B \setminus sig))$	Valid signature
$verify(u_{id}, MVP)$	Valid Merkle path
$VMR = B_{-1}.\text{nVRM}$	Merkle root matches the latest $nVMR$
$nVMR = MerkleRoot(\mathcal{M}^{*'})$	Next Merkle root is for the update $\mathcal{M}^*$
$\forall v(Valid(v)) \quad v \in votes$	All votes are valid

Tab. 7: Block verification

2.3 Elections

Votes can be send by every member $m \in \mathcal{M}$ at every time.

$$Vote = (u_{id}, \mathcal{V}_{u_{id}}, sig, b_{id}, b_{hash}) \tag{5}$$

Table 8 defines the components of a *Vote* introduced in definition 5.

	Definition	Comment
u_{id}	$:= m \in \mathcal{M}$	Sender of the Vote
$\mathcal{V}_{u_{id}}$	$:= [v_1, \ldots, v_n]$ with $v_i \in \mathcal{M}$	The votes (List of users)
sig	$:= sig(u_{id}, Vote \setminus sig)$	Signature
b_{id}, b_{hash}	$:= B_{-1}.b_{id}, hash(B_{-1})$	Block reference

Tab. 8: Vote

Like suggestions, votes also include a block reference as TaPOS.

A vote V is valid for the blockchain $\mathcal{B}$ with the latest block B_{-1} and can be confirmed if the conditions in table 9 hold.

A shared vote must be added to the next block a delegate builds. Unlike suggestions, votes must not be ignored. For each block B_S that includes votes, the signing delegate needs to compute the new election result. First the last votes are retrieved from the blockchain,

$\exists b(hash(b) = b_{hash} \wedge b.b_{id} = b_{id})$	$b \in \mathcal{B}$	Referenced Block is in $\mathcal{B}$
$b_{id} \geq B_{-1} - n$		Referenced block is not older than n blocks
$verify(u_{id}, sig, Vote \setminus sig)$		Valid Signature
$\forall v(v \in \mathcal{M}) \quad v \in \mathcal{V}_{u_{id}},$		Voted users are in $\mathcal{M}$

Tab. 9: Valid Vote

including the new block ($\mathcal{B} + B_S$). Each users votes $\mathcal{V}_i = [v_1, \ldots, v_n]$ are stored in the set $\mathcal{V} = [\mathcal{V}_{u_1}, \ldots, \mathcal{V}_{u_m}]$.

In the next step, for each member u_{id} the received votes are computed using equation 6. Each user has one vote. If he chooses to vote for multiple users, the vote is distributed equally between his choices ($\frac{1}{|\mathcal{V}_i|}$).

$$x_{u_{id}} = \sum_{i=0}^{|\mathcal{V}|} \begin{cases} \frac{1}{|\mathcal{V}_i|} & \text{, if } u_{id} \in \mathcal{V}_i \\ 0 & \text{, else} \end{cases} \qquad \text{for all } u_{id} \in \mathcal{M} \tag{6}$$

nVRM of the new block is set to the Merkle root of the list of u_{id}'s of the $|\sqrt{\mathcal{M}}|$ members with the highest votes $x_{u_{id}}$. If $nVRM \neq VRM$ this step updates the set of delegates $\mathcal{M}^*$.

2.4 Forks

Forks can occur if delegates share new blocks that are linked to the same previous block. In this case the blockchain opens into multiple branches. Most systems resolve forks by only using the longest chain, e. g. the branch that gets continued first. This approach is not suitable for group management, as it can temporally break the *same state* requirement as members are on different branches.

Forks can be prevented if messages get a *Server-Side-Timestamp* (SST). Members then can compare which block was shared first and only use this block. In figure 1 the block 1339 with timestamp 13:45 gets chosen over the block with 13:46.

The SST is attached to every message m by the messaging services server ($m' = (m, SST)$).

3 Asynchronous Messaging

In the mobile environment of messenger applications it can occur that messages are not received or received in the incorrect order. A distributed group management system, needs to compensate these cases.

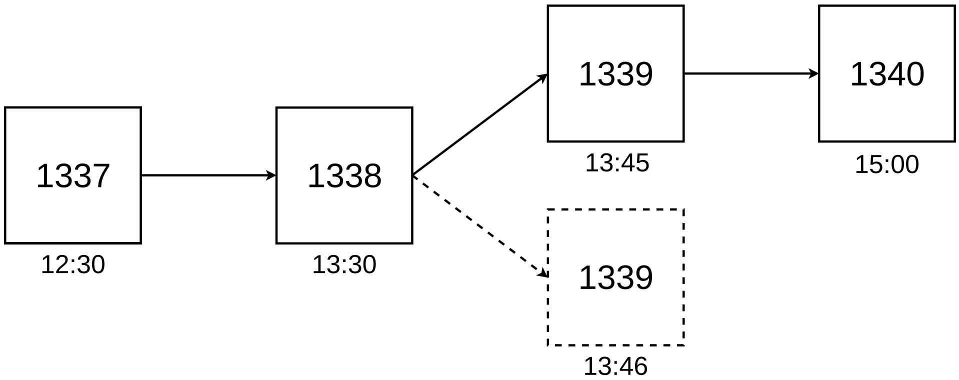

Fig. 1: Fork with *Server-Side-Timestamp*

The correct receipt of messages is only critical for block messages, as only these messages directly affect the state. Suggestion and vote messages can be either confirmed by a delegate that received the message or need to be resend.

A lost or delayed block can be detected if another block with a b_{id} > the latest local $b_{id} + 1$ is received. Such blocks need to be stored in a local cache. If the missing block is received (due to a delay), the cached block can be added to the chain after the delayed block. If no block is received after a time X, the member sends a $Sync(b_{id})$ to a random set of members $M' \subseteq \mathcal{M}$, which reply with the blocks $\{b_{id}, ..., b_{latest}\} \subset \mathcal{B}$. The requesting member then compares the responses and rebuilds his local blockchain accordingly.

4 Example

Figure 2 shows several messages to a group, and how the votes, suggestion and blocks affect the groups state.

5 Results and Conclusion

The updated protocol is implemented and tested using a dotnet core application, that simulates a group chat. Multiple instances connect to a RabbitMQ message server [Ra], which forwards the messages and acts as the messenger service. Asynchronity is simulated by random sleep times of the clients. Delivery issues are also simulated by the clients by randomly dropping or delaying incoming messages. To keep the group *alive*, random suggestion or votes are send if the clients gets online/wakes up.

In this simulated environment the described protocol was able to maintain a stable group. The updated protocol solves the described shortcomings and introduced a mechanism to deal

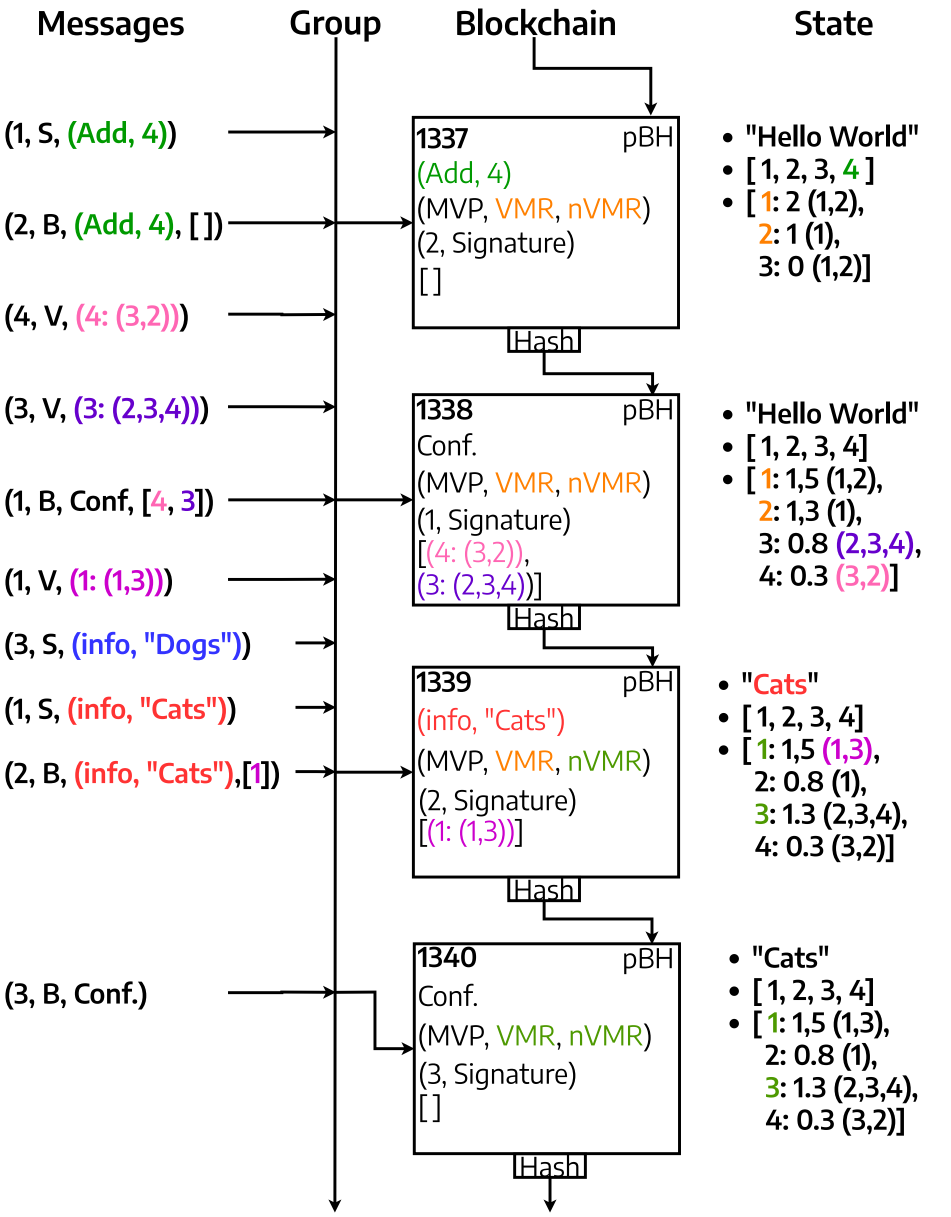

Fig. 2: Example Group

with message delivery issues. However this results allow no assumptions about real world performance, as the results strongly depend on the parameters chosen for the simulation, such as the range of the random sleep times.

The new voting system is less complexit and more resilient to delivery issues. It also provides a better representation of the users votes, as they do not need to be online at a specific time in order to share their vote and the set of delegates is directly updated if needed.

While an external *clock* resolves the forking issue, the SST might not be a suitable solution, as every message (due to the end-to-end encryption, the server cannot only *tag* block messages) needs to be touched. This increases the load on the delivery system. This could be resolved by a more sophisticated system, where delegates can anonymously request a signed timestamp from the service, that is include in the block.

6 Outlook

In further work, the properties of the proposed protocol can be compared to existing solutions, to figure out how the decentral approach performance regarding scalability and stability of group chats.

Finally the protocol could be integrated in an open source messenger like Signal, to test it under real world conditions.

References

[Bia] BitShares: BitShares Whitepaper, https://github.com/bitshares-foundation/bitshares.foundation/blob/master/download/articles/BitSharesBlockchain.pdf, Accessed Juni 30, 2019.

[Bib] BitShares: Delegated Proof of Stake (DPOS), https://docs.bitshares.org/en/master/technology/dpos.html, Accessed Juni 30, 2019.

[CPZ19] Chase, M.; Perrin, T.; Zaverucha, G.: The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption, Cryptology ePrint Archive, Report 2019/1416, https://eprint.iacr.org/2019/1416, 2019.

[EO] EOS: TechnicalWhitePaper, https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md, Accessed Juni 30, 2019.

[He19] Hellenbrand, A.: Decentral Group Management in Messengers with Delegated Proof of Stake, 2019.

[Ma] Marlinspike, M.: Private Group Messaging, https://signal.org/blog/private-groups/, Accessed Juni 21, 2019.

[Me88] Merkle, R. C.: A Digital Signature Based on a Conventional Encryption Function. In (Pomerance, C., ed.): Advances in Cryptology — CRYPTO '87. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 369–378, 1988, ISBN: 978-3-540-48184-3.

[Na09] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system, 2009, URL: http://www.bitcoin.org/bitcoin.pdf.

[Ra] RabbitMQ: RabbitMQ, https://www.rabbitmq.com/, Accessed Juni 30, 2019.

[RMS17] Rösler, P.; Mainka, C.; Schwenk, J.: More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema, Cryptology ePrint Archive, Report 2017/713, https://eprint.iacr.org/2017/713, 2017.

[Te] Telegram: Telegram FAQ, https://telegram.org/faq, Accessed Juni 21, 2019.

[Wh] WhatsApp: WhatsApp Security Whitepaper, https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf, Accessed Juni 21, 2019.