Original Draft Date: December 2018

ETHERIA: A Social Forum dApp on the Blockchain

Sections of report:

Code and Demo Video on Google Drive:
https://drive.google.com/drive/folders/1CvKxutz2_IRE6RMNc43TxOZyyrLgY7Jt?usp=sharing

NOTE: I deployed the contract on my MetaMask account, whose private key I will not upload.

I. Description of dApp and why it benefits from being on the blockchain

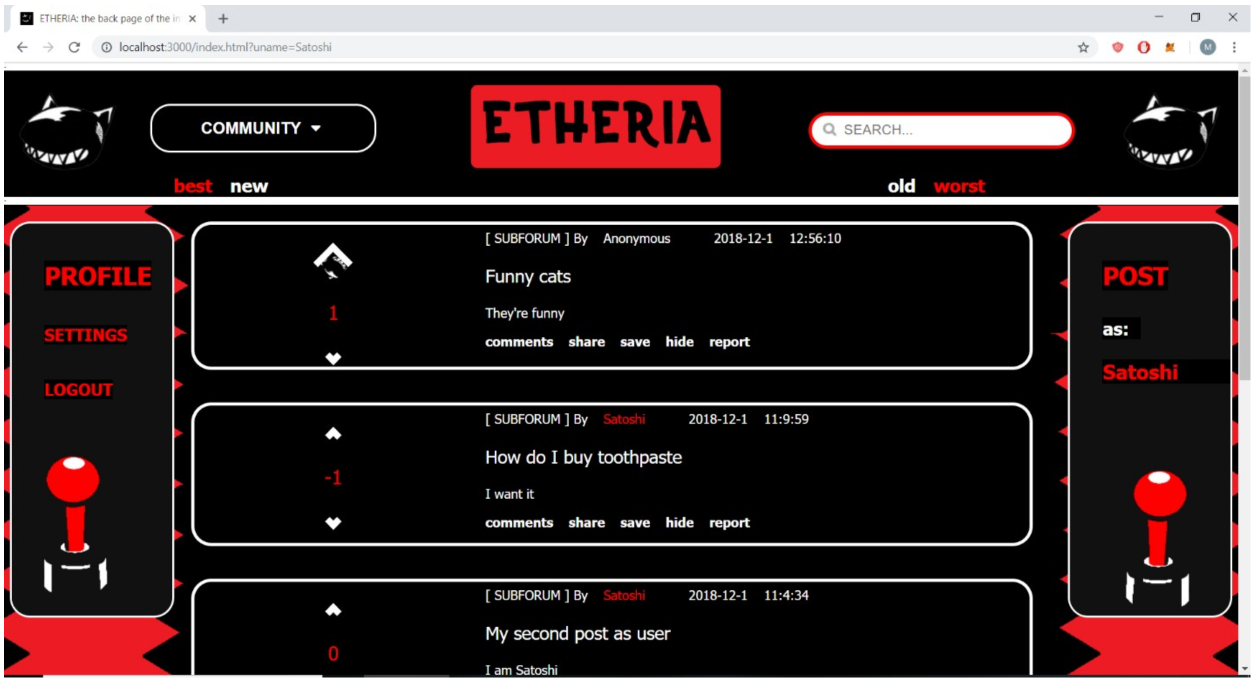
The main feature of this dApp is that it removes third parties from managing the posts and comments. This means that users can trust that their posts are not censored, and the posts associated to their accounts are not modified to make them say things the users do not want to say.

Posts, voting, comments, and signing up are done as transactions on the blockchain. Therefore, admins cannot choose which posts to feature on the home page, and mods cannot remove posts/comments. The downside is that users can't delete posts/comments after they submit them. Since each transaction has a cost, spamming is disencouraged lest a user incur costs.

Another main feature of this dApp is that a user account, associated to an address, can easily switch between posting/browsing as Anonymous or as a registered, named user. The reason for anonymity is so that other users can look at the content of a post itself, rather than at the reputation of the author of that post. The reason for named users is so that users can look at a user's posting history to get a sense of what they are like. Unlike Reddit, though each post (for both Anonymous and named authors) has votes in order to have users agree/disagree on them and also have most active posts be displayed on the 'best' sorting option, there is no accumulated karma. This is very important, as users posts are not featured based on popularity or other 'group think' metrics corruptible by centralized authorities.

Why the DApp is useful: Moderators on a social forum, such as Reddit, are able to remove or modify user posts that are publicly available. Their jobs include preventing obscene content from being shown. However, hosting on centralized servers means that the social forum is vulnerable to corruption by authorities or hackers; in other words, individual freedom of speech is able to be censored. Additionally, forums such as Reddit utilize a voting system to make votes be seen. But moderators potentially are able to manipulate this voting system to push big corporation approved posts, which weren't actually voted by users, to the top. A decentralized social forum would ensure that public information, individual opinions, and individual votes are returned to the hands of the users. The trade-off is that no content would be censored, so this forum is more susceptible to 'trolls' (and their bots) who may upvote content unfavorable to the existing community.

There already exist similar DApps (in development) such as Steemit. However, Steemit uses Steem tokens to allow content creators with more Steem tokens, which are rewarded by how much content is voted on, to decide how Steem tokens are distributed; this project will not use such a system because it incentivizes users to post to gain currency, instead of encouraging people to post opinions that they want to spread or generate discussion of (while maintaining that they are the author of these opinions). Unlike Steemit, this DApp will be designed so that the aim of users is not to gain currency, but to freely spread their opinions to the rest of the website without suspicion that their content will be removed or altered by middlemen. Sites such as Reddit are often criticized for promoting 'group think' because users post popular opinions to gain votes. A site that encourages users to post in order to gain tokens would likely facilitate group think. Another DApp, Synereo, cuts out the middleman between users and creators, and allows users to directly reward creators with 'Amps', Synereo's cryptocurrency. However, Synereo is still only in Phase 1 of development.



[SUBFORUM] By Anonymous 2018-12-1 12:56:10
1
Funny cats
They're funny
comments share save hide report

[SUBFORUM] By Satoshi 2018-12-1 11:9:59
-1
How do I buy toothpaste
I want it
comments share save hide report

[SUBFORUM] By Satoshi 2018-12-1 11:4:34
0
My second post as user
I am Satoshi

Figure 1: A screenshot of index.html. More footage can be found in the demo video on Drive

II. first_contract.sol (list not exhaustive)

NOTE: Since blockchains transactions are public, one can see whichever wallet addresses participated in the transactions. This makes it impossible to truly post as Anonymous from one's own wallet address. Thus, the “Anonymous” feature in this demo has a flaw. There is a solution I came up with that might fix this; however, due to Metamask only being able to have 1 account and due to lacking the time to use another tool to connect to Ethereum without errors, I was not able to implement this. I will describe this solution in section III.

This smart contract that was deployed in the Remix IDE. The web3 library was used to bridge the HTML to the smart contract. Ropsten, the local test network for Ethereum, was accessed via Metamask. Testing was also done using testrpc. The contract’s functions include:

- signup(): Uses a mapping to associate an address to a struct User, which stores the username, id, whether the user has signed up or not, and the id of the posts and comments they made
- event updatePage(): called after a user votes or comments and reloads the posts/comments to display the new results without having the user manually refresh the page. This originally occurred for posts, too, but since the transactions take longer to be confirmed in Metamask, and one would have to go from submit_post.html to index.html after submitting a post, this was not done for posts.
- addPost(): The post title, post content, date+time of the post, the user’s account address via msg.sender, the user’s name, and whether the user posted as anonymous or not is sent to the smart contract. With Metamask, due to obstacles working with asynchronous Javascript functions (see ‘overview of challenges’) and also creating + returning an array (an array with “extra members” about memory was returned, whether the temp array was in storage or memory) within a function, I could not loop through an array of post IDs and obtain return the mapping of that ID to a struct Post one at a time. So instead, I created an array for each property of a post; among these include array for post content and post addresses. The value associated at each index position X of the array is the value for post ID X. For example, post ID 2 (the 3rd post made) would have value ‘3’ in votesarr[], which stores and updates the current number of votes for post ID 2.
- Continuation of addPost(): anonymous post properties are pushed into certain arrays with empty values to maintain array position IDs (when these posts are called, the value ‘loggedin’ determines if the post should load it as linked to a user profile or not). The address for anonymous posts is NOT recorded; it is the empty address(0).
- addComment(): Sends comment content, date+time, whether a user is logged in or not, and parent post number, and creates a new Comment struct in the smart contract. This struct is accessed via 2 ways: within a Post struct’s array and by a mapping. The details for why this occurs are given in ‘Overview of Challenges’. Like in addPost(), anonymous vs named comments are added in different ways.

III. Proposed Solution to Anonymous posting

As mentioned above, wallet addresses are visible for every transaction. Currently, Anonymous posting works by simply not associating a post with an address/username. Anonymous posts will not have their addresses recorded onto the blockchain; also, each address should still only be able to vote on each post/comment once, though the vote can be changed. But the transactions themselves are visible in a public ledger. Therefore, this current implementation would not achieve 'true' Anonymous posting. One solution is to create many, depending on the current userbase size, of 'Anonymous' accounts. When people log-in as Anonymous, they are randomly assigned one of these accounts that is currently not being logged into.

Each time someone logs in as Anonymous, they must pay an aggregate Lake account. This fee, paid for using their own wallet, is the estimated amount that it takes to make a single post/vote/comment. Since each of these transactions is around the same fee, one can just take the max or average value of these fees (Eg posting is 0.1, voting is 0.01, and commenting is 0.2, so use 0.1 as the fee). This is because posting as Anonymous requires a fee too (to prevent spamming and depleting funds). Through a smart contract, the fee that is paid to the Lake account is evenly distributed to all the Anonymous accounts; no one is in charge of the Lake account. The fee is evenly distributed to prevent the Anonymous account from being traced to the address that paid to be assigned an Anonymous account. How the Lake distributes this transaction is tricky; it would take some work to calculate what fee is sustainable given the number of Anonymous accounts vs the number of users logging in as Anonymous during certain time periods (as frequency of Anonymous posting differs by time period).

Even distribution from Lake to Anonymous is bad if the number of Anonymous accounts is too high compared to the number of users using them, as this would give the accounts insufficient funds for transactions. Thus, a better solution is to split the fees directly to a few or just one account. This account is NEVER the account that user X, after paying, is assigned to! Instead, many accounts are paid for beforehand by previous users. After several transactions are done, these 'waiting accounts' become eligible to be accounts that are assigned to users.

After an Anonymous transacts a single post/vote/comment, they are logged out and must pay another fee to log in as Anonymous again; when they do log back in, they are randomly assigned an account different than the one they were previously logged into. Thus, each time someone logs in, due to the sheer amount of Anonymous accounts available compared to the userbase size, they are assigned a different account. This prevents several Anonymous transactions, all done within the same 'login', from being linked to one another. This also serves to associate a fee with every transaction.

A tricky obstacle for Anonymous posting is voting. There are good reasons why Anonymous accounts should not be able to vote. First off, votes from named accounts are not visible to the public. The public is only aware that transactions occurred, and though they may be able to deduce whether the transaction was a vote/comment/post, they are not able to see whether a user voted up or down. That action is not recorded within the blockchain; only the state of a vote is changed, and whether a user has voted or not. So voting as Anonymous should only be done if it is possible to vote without being associated with even voting at all!

So let's assume there is a way to vote without being associated with an account. The second issue is that even though there are a finite number of Anonymous accounts, one could still spam Anonymous voting. But an argument against this is that one can just use bots to create multiple named accounts (each one requiring a fee to sign up) and spam votes too! So let's just say

Anonymous can vote. Since multiple people will be using an account more than once, how can more than one vote, from different people, using the same Anonymous account be done? Creating an account for just one transaction is wasteful. Additionally, allowing voting once on a post/comment per address would defeat the purpose of Anonymity.

Let's look at what happens when a user decides to vote as Anonymous. First, the smart contract should check if that Anonymous account voted yet on that specific post/comment. If it did not, then it can vote. If it did, then it CANNOT change its vote; due to how Anonymous accounts are set up, all Anonymous votes are final! This is similar to how posts/comments are final, too. Thus, the user must be logged out of that Anonymous account, and the smart contract must re-assign the user to a new Anonymous account that did not vote yet on that specific post/comment. The user should not be assigned a new fee when their accounts are switched this way, as they have already paid to log into that previous account.

One issue with this is that spammers can vote so many times that they use up ALL the Anonymous accounts on a single post/comment, thus not allowing ANYONE ELSE to vote on that post/comment. It is also useless to associate a wallet address with whether it has voted on a post/comment, since the only thing this would accompish is to obfuscate whether one has voted up or down; but all transactions already do this in the first place (whether one is Anonymous or not) as they do not record what happens, only that a transaction has occurred between what addreses. In conclusion, since spamming is an issue, voting does not seem viable when one is Anonymous.

Although it is public whenever an account pays to become Anonymous, the public will not know what action that account took as Anonymous, due to the assumption that there will be many transactions going on at once, and so it is hard to trace which Anonymous account that address was assigned to.

Here is a summary illustrating the proposed Anonymous posting/commenting procedure:

1. In order to use the site, each user must first make a named account associated with a wallet; Anonymous is not the default. So the user logs into a named account and wants to log in as Anonymous. The named account pays a fee to a Lake account.
2. The fee is paid to the Lake account, and the Lake randomly gives the fee to some randomly chosen Anonymous account X. Now the smart contract will randomly assign, not to the user's wallet address but to the computer that the user is using (this assignment is not public as it is not a blockchain transaction), a prepaid Anonymous account Y (DIFFERENT from X) to the user
3. The user will make ONE post/comment as the account, and whether they are logged out or logged in doesn't matter. Now when they make ANOTHER post/coment, they must be logged out of their anonymous account and be logged into their named account; they will pay ANOTHER fee to be logged into a DIFFERENT Anonymous account.

IV. HTML Pages

- Index.html: The home page that displays posts. Accessed by clicking the logo 'Etheria'
- Post.html: This is accessed by clicking on a post's name. It displays the posts and the comments replying to it. Users can post comments, and a few moments after the comment-sending

transaction is confirmed, the page will automatically reload with the new comment due to a Solidity event

- Profile.html: If accessed by the side bar from a non-anonymous user, it displays the currently logged in user's posts and comments. If accessed by clicking on the author of a post or comment, it displays the author's post and comments (along with the votes each one contained, though one is unable to votes on these posts/comments through the profile page). The users clicks on buttons to toggle between displaying posts and comments. Posts can be displayed in sorted order. Even though Anonymous posts/comments may be posted by the same address of a registered user X, Anonymous posts/comments will NOT be displayed on X's profile page
- Submit_post.html: Accessed by clicking 'post' from the side bar. The user can make a post.
- Login.html: Accessed by clicking 'login' from the side bar when a user is anonymous. Contains signup and login.

The dropdown menu for subforums is coded in script.jss, and the styles are coded in style.css

V. Features include (list not exhaustive)

- Users can submit posts/comments as either a named user or an anonymous user
- Users can vote on posts and comments. Once the transaction is confirmed, after a few moments, the page will automatically reload with the new vote due to a Solidity event.
- Sort by votes: Posts, both on the Home page and Profile page, can be sorted according to best (most upvotes are first), new, old, and worst. If, for instance, 'best' is selected, then upon changing the votes on a page, if the posts should now be in a new order, then the page will automatically reload to show the new changes. This is the same for all sorting options.
- Sign up : Each address account can have 1 account. Each username must be unique.
- Login: If the user did not sign up yet, logging in will have the user remain Anonymous. Once logged in, a user will remained logged in as they move from page to page, until they logout or close the webpages. This information is passed around via query parameters.
- Logout: On any page, a user can immediately logout of their account to browse, post, vote, etc. The user will remain on that page and will not be taken to another one. This makes it convinient to switch from being a named user to an anonymous user. Whether a user is currently named or anonymous is diplayed on the right side bar.
- Links from a post/comment: on both profile and index pages, a user can click on the name of a post to go to the page with that post. A user can click on 'comments' on a comment to go to the page with that comment. For named posts, a user can click on the author of a post/comment to go to the profile page of that author.

VI. Overview of Challenges (list not exhaustive)

- Asynchronous functions: In Metamask, all Javascript code must call Solidity functions using Asynchronous functions (meaning that when the function is called for some task, the system does not have to wait for that task to finish and can initiate a subsequent task). This made it difficult to call Solidity functions and use them in simple ways, such as storing them in variables or using them with Javascript code (as the results returned by the Solidity functions would not affect Javascript variables. Functions such as load_posts() in index.html required Javascript variables, namely it had to put all the posts within an array and then sort the array in Javascript, but now I could not push post objects into the array). Thus, the code I had finished writing to be used with testrpc had to be completely re-worked to accommodate Metamask. Due to my lack of experience with asynchronous functions, I resorted to crude work-arounds that involved DOM methods. Another solution would be to use nests of Asynchronous functions; I only did this in some cases to avoid very deep nested levels.

- Using testrpc, the javascript function load_posts() loops through the posts stored on the smart contract and populates a string of HTML code, inserting values returned by Solidity functions. The issue with this is that originally, these posts were first stored in an array, and that array was sorted in Javascript. Using Metamask, I could now either sorted the array in Solidity, or find some crude work-around in Javascript. I opted to first load the posts frameworks, giving them each a page index ID (different from post IDs) but not populating them- only the number of posts (based on the number of posts currently in the smart contract) is needed. Then, using the new addPosts() as detailed in the above section 'first_contract.sol', all the posts are returned and put into an array with addPosts(). This array is then sorted. The sorted array index positions are now associated with the page index IDs, so say, if for instance posts are sorted by 'best', post 4 can now be in position 0 if it has the most votes.
- Within load_posts(), the voting buttons had to be associated to the sorted posts. With Metamask, this was a challenge because I could only populate the Javascript function voteup(), which calls the Solidity functions to alter the votes states of the smart contract based on post number, with the page index ID, but not the post ID after sorting. I resorted to an even cruder work-around where, when I called getPosts(), I stored an association between page index ID and post ID within dummy div tages using DOM, and when voteup() was clicked on, it would get the associated post ID for that page index ID and send that post ID to the solidity function votePostUp().
- Due to the crudeness of the sorting posts work-around, I did not sort comments. This made the load_comments() function easier to code, though I still had to rewrite it for Metamask to populate the page using DOM methods.
- I ran into issues when updating votes for comments displayed on index.html and profile.html. Namely, comments are stored within struct Post's arrays 'post_comments', so that each post page loads its own comments. Thus, post 1 can have comment ID 2, and post 2 can also have comment ID 2. To load comments on a user page, each comment was associated with a more 'global' ID that was based on order of upload, instead of parent post ID. But accessing the SAME comment through a Post struct vs accessing it through a mapping with ID as key did not seem possible. I could not create a comment in storage and associate that with the struct's array and a mapping, as that created 2 separate copies, so voting on one copy did not affect the other. Many other attempts to reference the same Comment struct in 2 different ways still resulted in 2 separate copies. Thus, I made another crude work-around where voting would just update both copies. This creates redundancy in storage, though one could simply associate comment ID with a copy of the voting state to reduce this. However, if one were to associate comment IDs

with parent comment IDs in comment threads (which were not implemented), this would take up even more redundant storage.

- Gas transactions: Sometimes Metamask would set the gas fee as 0, meaning I had to manually make it '1' so it would automatically find the right nonzero gas fee to use
- Eventually during coding, even though it worked fine before, testrpc gave errors such as "VM Exception while processing transaction: out of gas" despite starting testrpc with a huge gas limit such as "testrpc -l 4500000000000". This also occurred with ganache-cli, which is the newer testrpc. I avoided strings and opted to use bytes32 (bytes or bytes[] gave errors). Thus, I had to switch to using Ropsten, which was slower in processing transactions.
- In Solidity, one cannot store a mapping within a struct. There are also issues with using storage (allocates space within storage and references it) vs memory (a temp copy that's gone after a function) in certain specific cases; these are too numerous to list out. One cannot create a mapping with a struct as a key. There were many related issues such as these, but there are too many to list in this report.

VII. Possible extensions to this dapp include

- Logging in with other accounts: Metamask cannot 'mimic' having multiple accounts as it only injects 1 account to web3. A possible solution to this is to run a geth node locally, opening it to listen to rpc. However, due to constraints with other projects at this time, I did not have time to get this to work properly.
- Altering how Anonymous posting works (detailed in section III)
- Only vote once per post/comment, and a user cannot vote for their own posts/comments: Since I wanted to be able to vote on a post multiple times in a demo, I did not implement this. The associated solidity function would have taken an address+post and returned 1,-1, or 0. In CSS, I have code which make a vote permanently be enlarged and have the logo peaking out (as occurs when one hovers over an upvote/downvote) after it was voted on.
- Sort by comments
- Comment replies in a tree, create new comment box at comment the user is replying to, collapse threads
- Subforums: I have a dropdown menu to access subforums, though I did not implement them
- Search: Shown in index.html
- Settings: Shown in the side bar
- Share, save, hide, report: Shown below each post/comment
- User inbox, notify replies
- Modifying the smart contract to remove spam/troll posts
- Subforum voting features that allow users to ban other users based on votes: This can be done by randomly selecting (to avoid bias) users to determine whether someone should be banned or not, say for spam. Alternatively, an AI can be used, but how to update the AI's classification and decision processes is another issue.