SETHER

Blockchain meets Social Networks

SETHER

BLOCKCHAIN MEETS SOCIAL NETWORKS

Sether is platform that provides blockchain integration with social network APIs, allowing any blockchain enterprise to communicate, get information, analyze and monitor social networks.

The platform also provides custom analytic and monitoring capabilities for blockchain operations and enterprises. Users can run custom queries on social network data to get desired insights, find influencers, see the reach of their posts, create watchlists to be notified when certain events happen and see the sentiment for various social network objects such as posts, messages, pages, events and more.

Contents

1 Introduction

The intent of the Ethereum blockchain is to create an alternative protocol for building decentralized applications, providing a different set of tradeoffs that will be very useful for a large class of decentralized applications, with particular emphasis on situations where rapid development time, security and the ability of different applications to interact with each other, are important.

Ethereum does this by building what is essentially the ultimate abstract foundational layer: a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions.

But for smart contracts to operate based on real world events (and provide maximum value), they must first receive real world data and take decisions based on that data. Smart contracts can run algorithmic calculations as well as store and retrieve data, but because every node runs every calculation, it's simply not feasible to make arbitrary requests for data. Data Oracles resolve this issue by providing the results of any query to any contract.

Oraclize and ChainLink provide services for linking existing APIs to the blockchain, but are limited in that each request has a price and making multiple requests to achieve a single action is usually too expensive and require many blockchain transactions that can introduce serious delays in the decision making process.

Using simple Oracle services to work with social networks and do complex queries and aggregations requires the developer to create complex smart contracts that make a lot of requests to simple data oracles, which is not feasible. One example is the OAuth authentication mechanism used by most social networks that requires the developer to do several operations just to login to the service.

Interacting with social networks brings even more value if you can benefit from solid analytic and monitoring services that can be used in smart contracts to make decisions based on user behavior, social events or even on how the competition is doing.

The need to use specialized data Oracles is required to enable smart contracts interact with the social network ecosystem.

3   Architectural Overview

Sether bridges two main environments: the on-chain environment and the off-chain world. It will support public chains like Ethereum, Bitcoin, Rootstock mainnets and testnets, private Ethereum-based chains (such as Monax) and non-blockchain applications by providing an abstraction layer.
Sether has been designed to accommodate a pluggable service architecture, making possible to add, replace and update social network providers and analytic services.
It is well known that social networks make frequent updates to their APIs, releasing many versions that deprecate or introduce various features. Sether will make these makes transparent for developers, by keeping its API always up to date.

For Ethereum, the main flow of a blockchain application that needs to use the Sether API is an asynchronous message exchange:

1. A user or an application broadcasts a transaction by executing a method of a smart contract that extends the Sether smart contract. The method contains a special instruction that will fire an event that is catched by Sether, who is constantly monitoring the blockchain for these events.
2. Sether will compute the result, which is then signed and broadcasted as a blockchain transaction carrying the result. In the default configuration, such a transaction will execute the *__sthCallback* function of the smart contract.

Visually, the main flow can be represented as a sequence diagram:
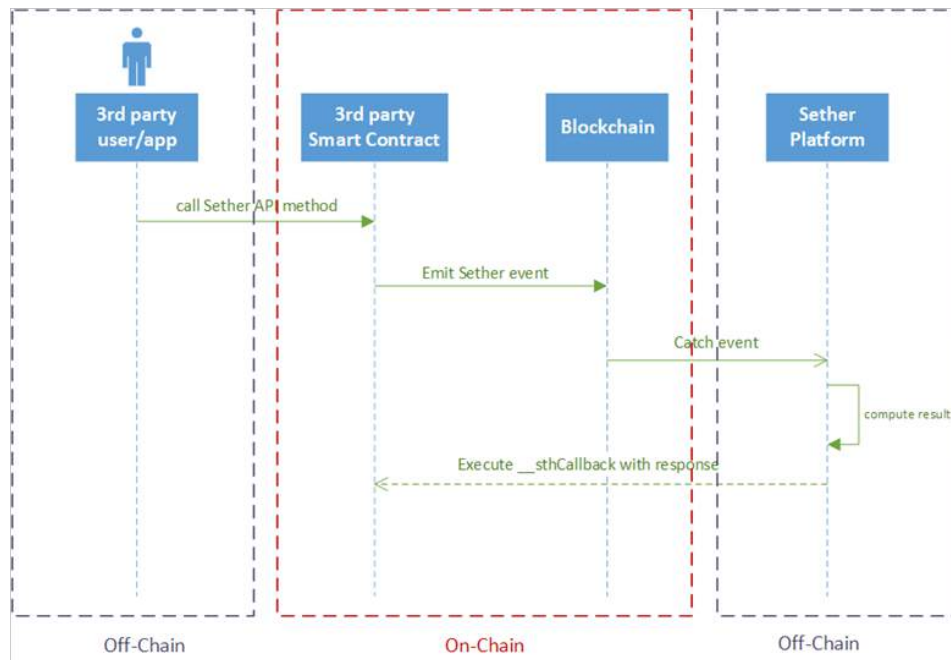
3.1 On-Chain Architecture

Sether is an oracle service. It will return replies to queries made by or on behalf of a user contract. The on-chain interface of Sether will be a smart contract that will offer a rich API to query, monitor and analyze social network data.
User contracts have to extend the smart contract of Sether in order to use the API methods. The on-chain API methods are described in Chapter 11 "API Documentation".

The easiest way to understand how Sether will be used, is to analyze a sample smart contract developed using the Sether API. The example below shows how to get the number of Facebook shares of a given post. We can observe two important aspects of API usage:

1. The example contract has to extend the Sether smart contract
2. The Sether smart contract is defined in the setherAPI.sol file, which can be pulled from the Sether Github repository available here: https://github.com/mware-solutions/sether

```
pragma solidity ^0.4.11;
import "github.com/mware-solutions/sether/setherAPI.sol";

contract ExampleContract is usingSether {
    function ExampleContract() payable {
    }

    function __sthCallback(bytes32 id, string result) {
        if (msg.sender != social_address()) throw;
        if(result > 10000) {
                // do something
        }
    }

    function verifyPostReach(string access_code) public payable {
            sth_ana_getReach('11784025953_10155105941610954', 'FACEBOOK', access_code)
    }
}
```

3.2 Off-Chain Architecture

Sether will have an elastic, plug-and-play extensible infrastructure of compute nodes split in the following categories: **ROUTERS**, **COLLECTION** and **ANALYSIS**.

ROUTERS - will fetch requests from contracts that extend the Sether contract and will return responses back to them via the callback method.
The routers will manage the requests as jobs organized in a queue and will dispatch them based on the requested information type. Each such job is a set of tasks that are processed in a request pipeline that interacts either with the target social network, either with other type of node. Each task performs a specific operation and passes its result to the next task, ultimately reaching a final result. ROUTER nodes have built-in protocols for data conversion in different blockchain formats.

COLLECTION - download social network data needed by ROUTER and ANALYSIS nodes. Downloaded data is fetched in-memory and never stored for privacy reasons. COLLECTION nodes have built-in protocols for each social network including for example authentication protocol negotiation, HTTP requests or JSON parsing. COLLECTION nodes pass collected data back to ROUTER or further to ANALYSIS nodes, based on the orchestration made by ROUTER nodes.

ANALYSIS - these nodes can execute monitoring and analytic tasks on data coming from COLLECTION nodes:
- custom queries in Open Cypher format (https://www.opencypher.org).
  We chose the Cypher query language because of its closeness to human language, high query complexity and very wide syntax coverness and applications. Social network data can be represented in a graph structure and Cypher is the industry's most widely adopted graph database query language.
- keywords monitoring
- computation of graph algorithms: PageRank, Degree Centrality, Eigen Centrality, Betweenness Centrality. More will be added based on community feedback
- language detection
- sentiment analysis
- user profiling
- behaviour analysis

ANALYSIS nodes will store data in a fast performance in-memory graph to allows such operations to be executed in milliseconds. The implementation of the in-memory graph can be found on Github (https://github.com/mware-solutions/memory-graph).

As an example of Sether API use case, imagine a company which develops an application that rewards its users when they share company products on social networks. The on-chain and off-chain communication flow would be the following:
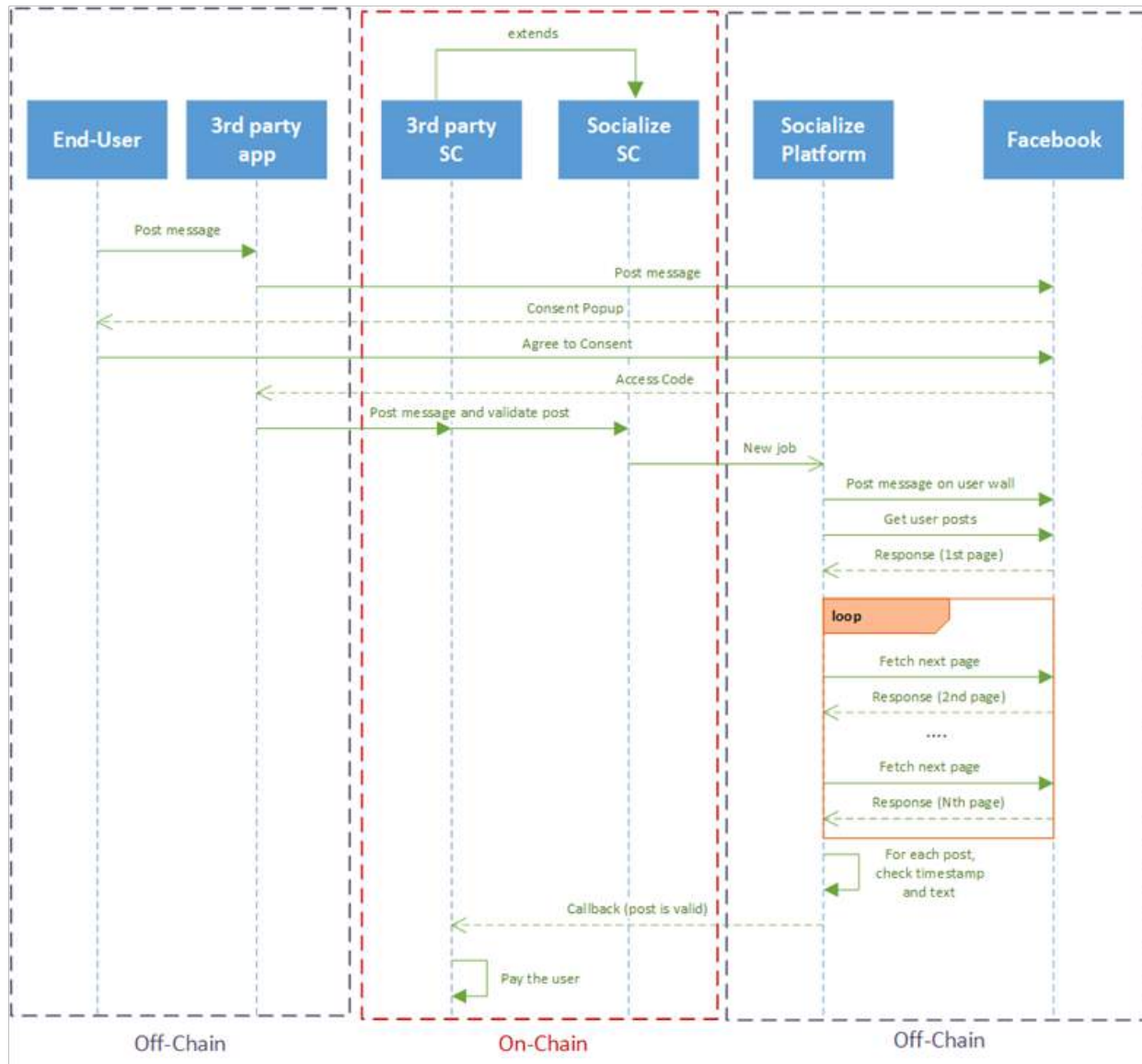
1. While using the 3rd party app (ex. online store), in order to be rewarded, the user wants to share a product on his Facebook account

2. The 3rd party app will ask the user permission for the 3rd party Facebook app to post the message on his wall

3. The user agrees

4. The 3rd party app will request a temporary access code from Facebook

5. The 3rd party app will ask his smart contract to post the message and validate that the message was actually posted.

6. The 3rd party smart contract calls Sether the smart contract API

7. The Sether smart contract will emit an event that will be picked up by the off-chain Sether platform.

8. The off-chain Sether platform will create a new job in an enclave and start the the interaction with Facebook. More information about enclaves is given in the Confidentiality section.

9. The platform can make quite a large number of requests to Facebook to fetch all user posts. Social networks use paginated result sets to keep responses small. Fetching the next page of results requires another call to the social network API, however the smart contract is executed only once.

10. The off-chain Sether platform sends the reply to the Sether SC, and therefore to the 3rd party SC.

11. The 3rd party SC receives the confirmation and rewards his user.

4   Security

Data oracles must be secure. If a smart contract security gets a false data feed, it may payout the incorrect party.

The blockchain, as a distributed system offers very strong security. Users rely on the blockchain to correctly validate transactions, preventing data from being altered. A supporting oracle service must have security standards aligned with the blockchain it supports.

An oracle must be an effective trusted third party, providing correct and timely responses. The security of any system is only as strong as its weakest link, so a highly trustworthy oracle is required to preserve the trustworthiness of blockchain services.

Sether will:

1. Receive a request from a smart contract
2. Get the data from a trusted external source.
3. Run queries and computations on the data if needed
4. Provide a response to the requester

Social network platforms are considered trusted. This dictates that an that oracle acts as a trustworthy bridge between the external source and the smart contract.

The request coming from the smart contract must also be secure. Smart contracts send requests in clear over the blockchain. This is not something desirable and there are many situations in which the request contains sensitive information. To ensure confidentiality, all requests will be encrypted using a public key that belongs to Sether. Only Sether will hold the private key that can decrypt the contents of the request, and this operation will be done in an **enclave**. More on enclaves will follow in the next chapter.

Sether will never store any request or response data. All operations will be done in-memory, and will immediately destroy any sensible data used.

Classical data oracle security issues do not entirely apply to Sether. The external sources - social networks - are known and assumed to be trustworthy, because they are accepted by the industry. Therefore, Sether does not need to prove the authenticity of external services.

An additional concern is availability. If Sether services are not available, smart contracts will not be able to do their work. Sether will use a network of redundant and highly available cloud nodes that can be monitored by the community. Moreover, the Sether smart contract will always return a comprehensible and catchable error if for some reason the backend becomes unavailable.

A data oracle should have it's code made available to the public for complete transparency. The software that will be used to implement the Sether platform will be completely open-source. Our work in progress is already on GitHub (https://github. com/mware-solutions) and can be monitored. The deployment of our software in the production environment will be handled by automated workflow that will ensure a transparent execution environment which is aligned with the public code available on GIT.

5   Confidentiality

Sether will use only hardware that supports Intel's recent Software Guard eXtensions (SGX) set of instruction-set architecture extensions. SGX is an Intel technology for application developers who are seeking to protect selected code and data from disclosure or modification.

Briefly, SGX permits an application to be executed in an environment called an enclave that claims two critical security properties:

1. Enclaves protect the **integrity** of the application. Its data, code, and control flow is protected against subversion by other processes.
2. Enclaves protect the **confidentiality** of an application. Its data, code, and execution state are opaque to other processes.

SGX seeks to protect enclaved applications even against a malicious operating system, and thus against even the administrator of the host on which an application is running. SGX relies on software attestation. Attestation proves to a user that he is communicating with a specific piece of software running in a secure container hosted by the trusted hardware. The proof is a cryptographic signature that certifies the hash of the secure container's contents.
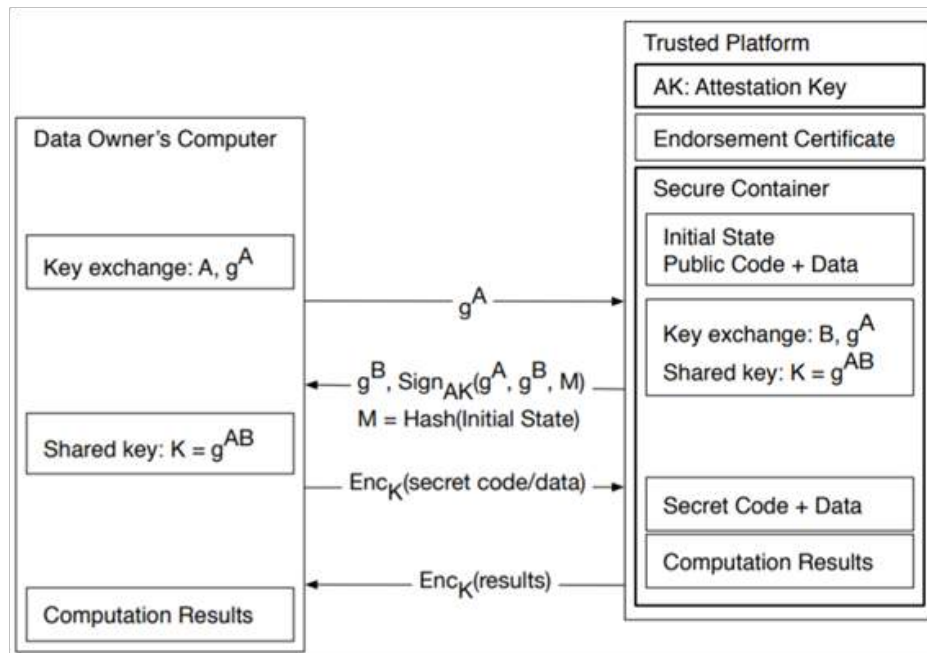
Software attestation proves to a remote computer that it is communicating with a specific secure container hosted by a trusted platform. The proof is an attestation signature produced by the platform's secret attestation key. The signature covers the container's initial state, a challenge nonce produced by the remote computer, and a message produced by the container.

The remote computation service user verifies the attestation key used to produce the signature against an endorsement certificate created by the trusted hardware's manufacturer. The certificate states that the attestation key is only known to the trusted hardware, and only used for the purpose of attestation.
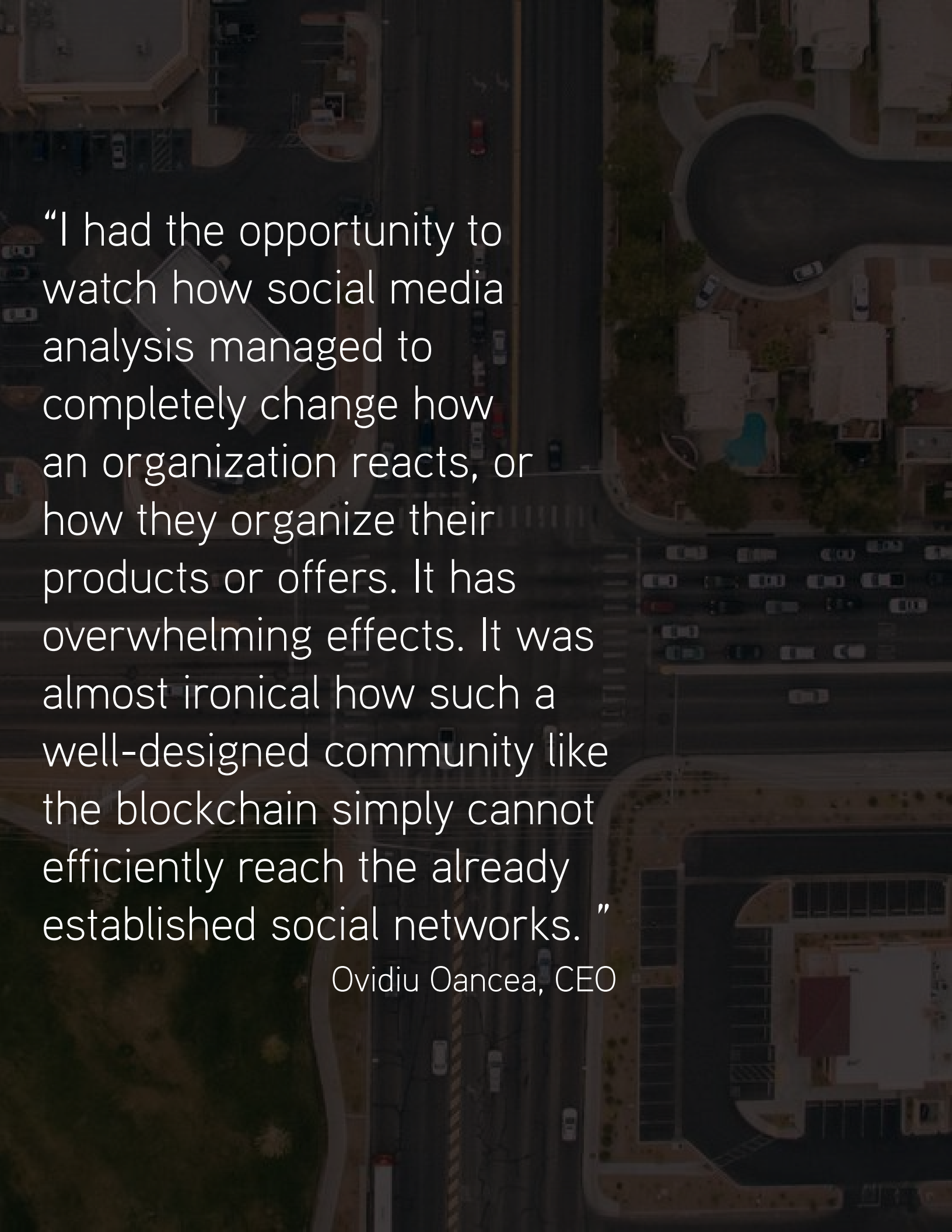
The attestation protocol proves that a specific piece of code ran on suitable hardware, producing a specific result. The proof is a signed statement (by the CPU's key), called a quote, which can be verified against the Intel server that executed the code.

Sether COLLECTOR nodes will handle the decryption and execution of the request in an enclave and distributing will distribute signatures that can provide a very strong assurance that the node executed a particular task. Signatures will be sent back to the *__sthCallback* method as a proof that the code was executed in an enclave.

Sether will have an online service where users can validate received signatures, to make sure their code and data were executed in a secure way.

Doing the decryption of the actual request in an enclave, it allows COLLECTOR nodes to securely manage sensitive information such as user credentials, making API calls very secure.

Sether will run on SGX-enabled Intel hardware and COLLECTOR nodes will use the Sether SGX framework to execute enclaved code. The framework is available on GitHub (https://github.com/mware-solutions/sether-sgx).

"I had the opportunity to watch how social media analysis managed to completely change how an organization reacts, or how they organize their products or offers. It has overwhelming effects. It was almost ironical how such a well-designed community like the blockchain simply cannot efficiently reach the already established social networks. "

Ovidiu Oancea, CEO

6   Use Cases

In general, there are many types of applications that can be built on top of Sether, both on-chain and off-chain. Enabling applications to interact with social network data opens the door to a whole new breed of services that can be built on the blockchain.

6.1   Customer Loyalty

Give discounts and promotional offers to specific customers if they promote your product on social channels. For example, if they tweeted about the last 5 items bought from your company, they automatically get a 15% discount on their next purchase. Using a smart contract.

6.2   Customer Targeting and Guaranteed Promotions

Use Sether to offer direct promotional discounts based on your followers social activity. For example, one of your Facebook followers posts about a new phone that he likes. Offer him a 30% discount on that phone if he buys it from your site, in the next 5 days. Automatically, using a smart contract, to prove the offer and discount are real.

6.3   Marketing Campaigns

Set clear terms with your promoters, enforcing targets and deadlines with a smart contract. Monitor, pay or give bonuses based on their social activity, reach, and public reaction, without actually getting involved. For example, you hired an influencer to promote your product and you need at least 1000 Facebook likes and 500 retweets by the end of your campaign. Pay him 0,1 ETH when you get 50%, and pay the rest if you get 100% by the deadline. Automatically, while you do something else.

6.4   Bounty Campaigns

There is no way to make sure that both sides of a bounty campaign, the client and the promoter, keep their promises. Using a smart contract, you could automatically pay a promoter 0,1 ETH only if they make a minimum of 40 posts, with a minimum of 70 characters. Both sides are happy.

6.5   Transparent political support

There is a huge pressure to be transparent about how a political campaign was financed, and who was involved in it. Be transparent. Set your budget and pay your supporters using smart contracts to openly run your campaign on social media. Prove to your electors that no ghost companies, shallow investment funds or unwanted parties are involved

6.6   Behavior analysis

Using Sether behavior analytics module, you can monitor a specific behavior by

describing it as a collection of queries. For example, we would like to *find persons which can influence large groups of people (> 100), preponderantly pushover minors (more than 80% of the members) by posting mostly (>75%) negative posts.* This can be reflected in the following pseudocode Cypher query:

```
MATCH (p1:Person)-[:is_influencer_in]->(g1:Group {group1.no_of_members > 100})
MATCH (p1)-[:has_posted]->(post:Post) collect all_posts in allposts
MATCH (p1)-[:has_posted]->(post:Post {post.sentiment = NEGATIVE}) collect (posts) in negative_posts
MATCH (memberofgroup:Person)-[:is_member_of]->(g1) collect (memberofgroup) in allMembers
MATCH (young_member:Person {age < 15})-[:is_member_of]->(g1) collect (young_member) in young_Members
WHERE young_members.size/allmembers.size > 80% and negative_posts.size/allposts.size > 75%
RETURN p1.NAME
```

(Influencer and sentiment characteristics can be precomputed using the analytics module)

Using behavior analytics, applications can measure the impact of social marketing campaigns, prepare for crisis or spikes in social activity and get real-time insights by monitoring mentions of their brand, company, products, and competitors.

6.7   Proof of Action™
Enhance your current business and build new services and applications that engage your users on social networks through gamification. Ask them to carry out specific actions: post keywords, links, pictures or videos, give reviews, post comments, like specific targets, retweet messages. Use Sether to validate their actions, give rewards or take business decisions. Both for blockchain and off-blockchain applications. With Sether.

6.8   Incentivize good PR
Hire new members for your PR department using smart contracts. You can then reward them your if your company, brand or product gets positive feedback on social networks. For example, if the general public opinion gets a positive boost regarding a product, you can automatically bonus the team responsible for its success, using a smart-contract.

6.9   Off-blockchain applications
The Sether API can be used from outside social contract, by all other applications that need to incorporate social network interaction, monitoring and analytics. This will be possible using the Sether SDK that will be available for public use.

Off-blockchain users that want to use platform functionalities can purchase SETH tokens to pay for their usage.

By attracting off-blockchain users to the platform, Sether will generate additional significant revenue that will increase its value.

7   Subscription models

Sether will use SETH as its token to charge for the usage of the API, both for smart contract and off-blockchain developers. Consumers of the service can either pay for each individual api call or purchase monthly or yearly subscriptions.

The API will be free to use for developing applications on testnets, but will impose daily usage quotas on API calls.

On mainnets, Sether will automatically recover the fee at execution time. The standard fee for each API call will be the equivalent of 0.01 USD. The fee consist of two parts:
1. The amount of Wei which corresponds to the USD price for the API call.
2. The amount of Wei which Sether will spend in gas for sending back the callback transaction

The subscription model will be the following (equivalent SETH prices in USD are shown):

BASIC	PROFESSIONAL	BUSINESS	ENTERPRISE
30 USD / month	100 USD / month	400 USD / month	Custom pricing
25,000 calls per month	100,000 calls per month	500,000 calls per month	Custom solution

All prices are indicative will be converted to SETH using the latest SETH/USD exchange rate, and all payments will be processed using SETH tokens.

8 Roadmap

Sether is currently in full development, with all our team being engaged in the project, and we are getting close to an alpha version for the Ethereum blockchain, that will be released in January 2018.

The releases plan is the following:

SETHER 1.0-alpha
Support for Ethereum testnets

Dev Tools
Setup data center community monitoring tools and provide web API console for testing and development

SETHER 1.1
Bugfix release

| Jan 2018 | Mar 2018 | Apr 2018 | May 2018 | Jun 2018 | Jul 2018 |

SETHER 1.0-beta
Support for Ethereum testnets and mainnet. Limited access will be provided to beta testers

SETHER 1.0
Production version for Ethereum nets and private Ethereum nets

SETHER 1.2
Production version for Bitcoin, Rootstock and off-chain applications

The source code for the entire Sether platform is and will be fully available on GitHub at the following link: https://github.com/mware-solutions

9   The SETH Token

The SETH token is a standard ERC20 token that can be used by multiple wallets and trading platforms.

The token will be used as the only currency for interacting with the Sether platform.

The SETH token smart contract and the crowdsale smart contract will be publicly available for auditing before the ICO launch, at the address: http://github.com/mware-solutions/ico

10 The ICO

We are committed to go live with Sether, regardless of the outcome of the ICO campaign. Hence, there will be no soft-cap, because the project will continue and will go in production.

The pre-ICO round will provide an early opportunity to loyal contributors to join in prior to the full ICO. The ICO will have a maximum duration of 4 weeks, closing the moment the tokens are sold out. During the round, the tokens are distributed in a first-come-first-served basis. Regarding the token issue timeline, the SETH tokens are planned to be issued to all contributors, as soon as their contributions are received.
This timeline may be subject to further delays in the event of any unforeseen legal, certification, or regulatory compliance roadblocks.

Up to one hundred million (100,000,000) SETH tokens will be minted. The tokens will be available at the time of the pre-sale, and will be distributed immediately.

10.1   Allocation

An initial pre-ICO will be launched on 4th Dec 2017 and the main ICO opening round is set to follow on 15th January 2018, 2pm GMT (delay due to the numerous holidays in December). The maximum crowdsale cap is set at $55 MM (fifty-five million), with accepted contribution tokens in both BTC and ETH. The total pool is fixed at 100,000,000 (100 million) tokens.

The initial price of 1 SETH will be 0.003 ETH.

The SETH tokens are intended to be allocated as follows:
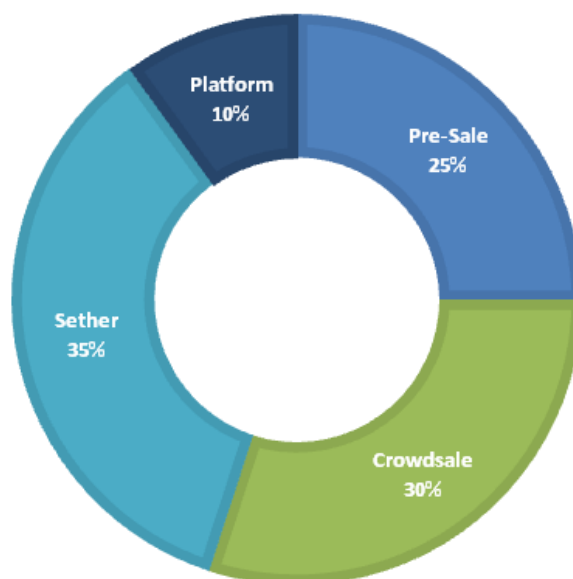
➔   25% (25,000,000) to be sold by the Company to pre-ICO purchasers.
➔   30% (30,000,000) to be sold by the Company to ICO purchasers.
➔   35% (35,000,000) reserved by the Company to incentivize community, beta testers and strategic partners. These tokens will be locked for 6 months after the end of the crowdsale. Afterwards, tokens will be diluted in 7% per year, during the next 5 years to support future steering of the project.
➔   10% (10,000,000) to be sold directly on the platform for service consumers. These tokens will be locked for 6 months after the end of the ICO.
➔   Any unsold tokens in the pre-sale will go into the crowdsale
➔   Based on the results of the public sale, at the end of the ICO, a proportional amount of tokens will be minted for the Company and the Platform to keep the balance and prevent dilution. Therefore, the community will own 55%, the Company will own 35% and the Platform will own 10% from all minted tokens.

For further information regarding our token sale, please visit the dedicated FAQ section on our website.

10.2   Pre-ICO

An initial pre-ICOs will be launched on 4th Dec 2017. The Company is pre-selling Tokens (prior to the Crowdsale) at a discount on a wholesale basis for larger volume purchasers or through authorized Affiliates.

The minimum contribution for pre-ICO:

➔   Minimum 0.1 ETH or BTC equivalent with a bonus of 40%

For information on the terms of the pre-sale please contact investor@sether.io

10.3   ICO

The main ICO opening round is set to follow on 15th January 2017, 2pm GMT. The delay is due to the fact that there are numerous holidays in December.

SETH Tokens are intended to be sold at the following rates:
- ➔   Week 1: 375 SETH tokens for 1 ETH (25% bonus)
- ➔   Week 2: 360 SETH tokens for 1 ETH (20% bonus)
- ➔   Week 3: 330 SETH tokens for 1 ETH (10% bonus)
- ➔   Week 4: 300 SETH tokens for 1 ETH (no bonus)

10.4   Usage of funds

The ICO will allow us to hire new talent, pay for hardware, pay for marketing, as well as for business and product development so that we can be the first to market with a platform that anyone can use.

A significant portion of the post-ICO funds, totaling 55% of the token total value will be allocated to the datacenter, R&D, personnel, the development of the social APIs. Around 20% of the funds will be allocated for marketing purposes and to drive platform adoption. The remaining amount will be dedicated to operational and administration expenses, partner enablement and legal fees.

The following chart represents the current expected allocation and may be adjusted as the project evolves:
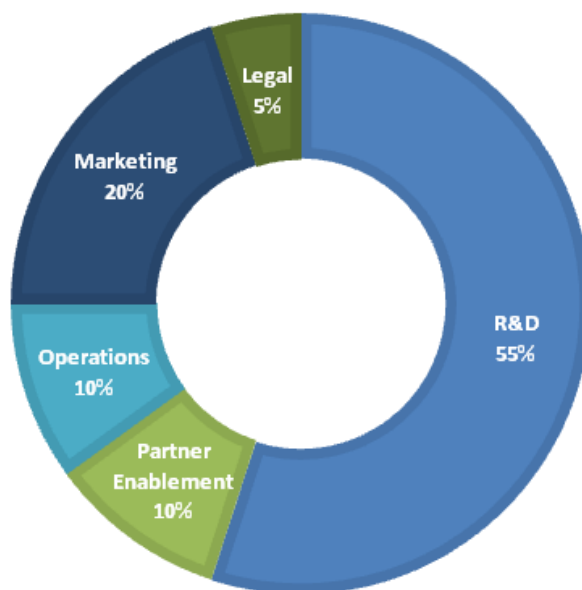
The values and percentages are:

- ➔ 55% will be used for **Research and Development**
- ➔ 10% will be used for **Partner Enablement**
- ➔ 20% will be used for **Marketing & Business Development**
- ➔ 10% will be used for **Running and Administrative** costs
- ➔ 5% will be used for Legal & Regulations

Research and Development – Developing the SETHER and all its components, hiring technical resources and establishing the infrastructure to support development.

Partner Enablement – Developing relationships with large service consumers and other strategic partnerships. Engaging possible joint ventures and incentives to drive platform adoption.

Marketing & Business Development - Developing relationships, marketing plans and cooperation strategy with large service consumers to expand the support of the SETHER platform and reach a wider audience of clients. Developing a marketing campaign targeting increased adoption of the platform by buyers.

Running & Administrative – Keeping the infrastructure running and ensuring permanent availability. Providing 24/7/365 technical support. Auxiliary personnel for the administrative department.

Legal & Regulation – Establishing foreign legal entities, complying with the ongoing regulatory framework, incorporation in other countries and other legal obligations.

10.5   Disclaimer

The Sether token sale represents a voluntary contribution towards the execution of this product vison by its current and future team.

The SETH token does not grant any direct equity stake nor profit sharing. It does not represent an ownership right or claim in the Sether ecosystem, revenues or intellectual property, either present or future. Despite Sether team's best efforts and diligence to bring this project forward, all contributors should be aware that their contributions are not refundable and accept the inherent risk of project failure at any stage of development.

This implicit risk is associated with any and all uncertainty of backing cutting-edge technologically-focused entrepreneurial projects, and can be affected by either internal or external factors that are out of the control scope of the team. Additionally, contributions may be subject to any applicable compliance regulations. The SETH token can be used in the Sether platform, with a corresponding utility value inside such ecosystem. The SETH token does not qualify as a security and does not provide a contributor equity share in the legal structure of any of the Sether entities.

10.6   ICO Security

Our system is designed to take into account recent events of wallet theft and phishing during ICO campaigns. This will be mitigated by using secure multi-sig wallets and by converting a significant amount of the ICO to fiat currency when possible.

We will also continuously monitor for phishing sites and will alert all Contributors through press releases and newsletters. We will communicate official information regarding wallet addresses only through email to registered members.

Our deflationary mechanism creates intrinsic value and a relationship between the platform growth and token value. The purpose is to mitigate the effects of earlier ICOs in which the tokens had a large demand on the first months, and then flattened, leaving the value at almost zero.

Our operating strategy includes periodical contact with Contributors through a mailing list and/or newsletter in order to keep them updated on current status of development, marketing, clients, and revenue. This strategy allows us to gain feedback from Contributors and keep them updated with the most recent news and developments.

11 API Documentation

11.1 Facebook

Method	Description
sth_fb_getAccessToken(key, secret)	Returns a new access token
sth_fb_getAlbumLikes(albumId) sth_fb_addAlbumLike(albumId, like)	Get people who like the album Like/Unlike the album
sth_fb_getAlbumComments(albumId) sth_fb_addAlbumComment(albumId, cmnt)	Get comments made on the album Add a comment to an album
sth_fb_getApplicationFeed(appId) sth_fb_addToApplicationFeed(appId, msg)	Get the application wall or post a message on the profile page
sth_fb_getApplicationInsights(appId)	Usage metrics for the application
sth_fb_getApplicationPosts(appId)	The application's own posts
sth_fb_getApplicationReviews(appId)	Reviews of the application
sth_fb_getCheckin(checkinId)	Represents a single visit by a user to a location
sth_fb_getCheckinComments(checkinId) sth_fb_addCheckinComment(checkinId, cmnt)	Get all comments for a checkin Post a comment for a checkin
sth_fb_getCheckinLikes(checkinId) sth_fb_addCheckinLike(checkinId, like)	Get users who liked the checkin Create a new like for the checkin
sth_fb_getComment(cmntId)	Returns a comment
sth_fb_getCommentLikes(cmntId) sth_fb_addCommentLike(cmntId, like)	Get users who liked the comment / Create a new like for the comment
sth_fb_getEvent(eventId)	Returns the information for an event, including the location, event name and which invitees plan to attend
sth_fb_getEventFeed(eventId) sth_fb_addToEventFeed(eventId, msg)	Get the event wall or post a message on the event wall
sth_fb_getEventUsers(eventId, responseType) sth_fb_rsvpEvent(eventId, responseType)	Get all the users that responded with Maybe, Attending or Declined RSVP the user for the event (with Maybe, Attending or Declined)
sth_fb_getFriendList(listId)	Get the details of a friend list, not including its members
sth_fb_getFriendListMembers(listId)	All users who are members of the friend list
sth_fb_getGroup(groupId)	Get the details of a group
sth_fb_getGroupFeed(groupId) sth_fb_addToGroupFeed(groupId, msg)	Get the group wall Post a message on the group wall
sth_fb_getGroupMembers(listId)	All users who are members of the group
sth_fb_getLink(linkId)	A link shared on a user's wall
sth_fb_getLinkComments(linkId) sth_fb_addLinkComment(linkId, cmnt)	Get all comments for the link Create a new comment for the link
sth_fb_getLinkLikes(linkId) sth_fb_addLinkLike(linkId, like)	Get users who liked the link Create a new like for the link
sth_fb_getPage(pageId)	Get the details of a page

sth_fb_getPageFeed(pageId) sth_fb_addToPageFeed(pageId, msg)	Get the page wall Post a message on the page wall
sth_fb_getPageLinks(pageId) sth_fb_addPageLink(pageId, link)	Get the links posted on the page Post a new link on the page
sth_fb_getPagePosts(pageId)	Get the page's own posts
sth_fb_getPageEvents(pageId) sth_fb_addPageEvent(pageId, event)	Get the events the page is attending Create a new event for the page
sth_fb_getPageCheckins(pageId)	Checkins made to this Place Page by the current user and his friends
sth_fb_getPhoto(photoId)	Get the details of an individual photo
sth_fb_getPhotoComments(photoId) sth_fb_addPhotoComment(photoId, cmnt)	Get all comments for the photo Create a new comment for the photo
sth_fb_getPhotoLikes(photoId) sth_fb_addPhotoLike(photoId, like)	Get users who liked the photo Create a new like for the photo
sth_fb_getPost(postId)	Get the details of an individual post
sth_fb_getPostComments(postId) sth_fb_addPostComment(postId, cmnt)	Get all comments for the post Create a new comment for the post
sth_fb_getPostLikes(postId) sth_fb_addPostLike(postId, like)	Get users who liked the post Create a new like for the post
sth_fb_getStatus(statusId)	Get a status message object
sth_fb_getStatusComments(statusId) sth_fb_addStatusComment(statusId, cmnt)	Get all comments for the status message Create a new comment for the status message
sth_fb_getStatusLikes(statusId) sth_fb_addStatusLike(statusId, like)	Get users who liked the status Create a new like for the status
sth_fb_getUser(userId)	Get the details of the user
sth_fb_getUserCheckins(userId) sth_fb_addUserCheckin(userId, checkin)	Get the places the user has checked-in to Check-in the user into a place
sth_fb_getUserEvents(userId) sth_fb_addUserEvent(userId, event)	Get the events the user is attending Create a new event for the user
sth_fb_getUserFeed(userId) sth_fb_addToUserFeed(userId, msg)	Get the user wall Post a message on the user wall
sth_fb_getUserFriends(userId)	Get the user friends
sth_fb_hasUserFrinend(friendId)	Checks if the given user is a friend of the current user
sth_fb_getUserGroups(userId)	Get the groups the user belongs to
sth_fb_getUserInterests(userId)	Get the interests listed on the user's profile
sth_fb_getUserLikes(userId)	Get all the pages the user has liked
sth_fb_hasUserLikedPage(userId, pageId)	Checks if the user likes the given page
sth_fb_getUserLinks(userId) sth_fb_addUserLink(userId, link)	Get links posted by the user Post a new link on the user's profile page
sth_fb_getUserPosts(userId) sth_fb_addUserPost(userId, post)	Get user's posts Create a new post on behalf of the user
sth_fb_getUserStatuses(userId) sth_fb_addUserStatus(userId, status)	Get the user's status updates Post a new status message
sth_fb_getVideo(videoId)	Get the details of a video
sth_fb_getVideoComments(videoId) sth_fb_addVideoComment(videoId, cmnt)	Get all comments for the video Create a new comment for the video
sth_fb_getVideoLikes(videoId) sth_fb_addVideoLike(videoId ,like)	Get users who liked the video Create a new like for the video

11.2 Twitter

Method	Description
sth_tw_getUserMentions()	Returns the 20 most recent mentions (tweets containing a user's @screen_name) for the authenticating user. The timeline returned is the equivalent of the one seen when you view your mentions on twitter.com. This method can only return up to 800 statuses. This method will include retweets in the JSON response.
sth_tw_getUserTimeline(userId)	Returns the 20 most recent statuses posted by the authenticating user. It is also possible to request another user's timeline by using the screen_name or user_id parameter. The other user's timeline will only be visible if they are not protected, or if the authenticating user's follow request was accepted by the protected user. The timeline returned is the equivalent of the one seen when you view a user's profile on twitter.com. This method can only return up to 3,200 of a user's most recent statuses.
sth_tw_getUserHomeTimeline()	Returns a collection of the most recent Tweets and retweets posted by the authenticating user and the users they follow. The home timeline is central to how most users interact with the Twitter service. Up to 800 Tweets are obtainable on the home timeline.
sth_tw_getRetweets(tweetId)	Returns up to 100 of the first retweets of a given tweet.
sth_tw_getTweet(tweetId)	Returns a single status, specified by the id parameter below. The status's author will be returned inline.
sth_tw_addTweet(tweet)	Updates the authenticating user's status, also known as tweeting. For each update attempt, the update text is compared with the authenticating user's recent tweets. Any attempt that would result in duplication will be blocked, resulting in a 403 error. Therefore, a user cannot submit the same status twice in a row. While not rate limited by the API a user is limited in the number of tweets they can create at a time. If the number of updates posted by the user reaches the current allowed limit this method will return an HTTP 403 error.
sth_tw_addTweetWithMedia(tweet, media)	Updates the authenticating user's status and attaches media for up-load. The Tweet text will be rewritten to include the media URL(s), which will reduce the number of characters allowed in the Tweet text. If the URL(s) cannot be appended without text truncation, the tweet will be rejected and this method will return an HTTP 403 error. Important: Make sure that you're using upload.twitter.com as your host while posting statuses with media. It is strongly recommended to use SSL with this method.
sth_tw_retweet(tweetId)	Retweets a tweet. Returns the original tweet with retweet details embedded.
sth_tw_getTrendingTopics(woeId)	Returns the top 10 trending topics for a specific WOEID, if trending information is available for it.
sth_tw_getLocations()	Returns the locations that Twitter has trending topic information for.
sth_tw_getClosestLocations(locId)	Returns the locations that Twitter has trending topic information for, closest to a specified location.

sth_tw_addPlace(lat, lon, data)	Creates a new place object at the given latitude and longitude.
sth_tw_getFriends(userId)	Returns a collection of user IDs for every user the specified user is following (otherwise known as their "friends")
sth_tw_getFollowers(userId)	Returns a collection of user IDs for every user following the specified user
sth_tw_lookupFriendships(userIds)	Returns the relationships of the authenticating user to the comma-separated list of up to 100 screen_names or user_ids provided. Values for connections can be: following, following_requested, followed_by, none
sth_tw_getIncomingFriendships()	Returns the relationships of the authenticating user to the comma-separated list of up to 100 screen_names or user_ids provided. Values for connections can be: following, following_requested, followed_by, none.
sth_tw_getOutgoingFriendships()	Returns a collection of numeric IDs for every protected user for whom the authenticating user has a pending follow request.
sth_tw_getUserFriendship(userid1, userId2)	Returns detailed information about the relationship between two arbitrary users.
sth_tw_getUserDetails(userId)	Returns a variety of information about the user specified by the required user_id or screen_name parameter.
sth_tw_searchUsers(query)	Provides a simple, relevance-based search interface to public user accounts on Twitter. You can query by topical interest, full name, company name, location, or other criteria. Exact match searches are not supported. Only the first 1,000 matching results are available.
sth_tw_getLists()	Returns all lists the authenticating or specified user subscribes to, including their own
sth_tw_getListTweets(listId)	Returns tweet timeline for members of the specified list. Retweets are included by default.
sth_tw_getListMemberships(listId)	Returns the lists the specified user has been added to. If user_id or screen_name are not provided the memberships for the authenticating user are returned.
sth_tw_getListSubscribers(listId)	Returns the subscribers of the specified list. Private list subscribers will only be shown if the authenticated user owns the specified list.
sth_tw_isUserMemberOfList(userId, listId)	Check if the specified user is a member of the specified list.
sth_tw_getListMembers(listId)	Returns the members of the specified list. Private list members will only be shown if the authenticated user owns the specified list.
sth_tw_getSubscribedLists()	Obtain a collection of the lists the specified user is subscribed to. Does not include the user's own lists.

11.3 Google+

Method	Description
sth_gp_listUserActivities(userId)	List all of the activities in the specified collection for a particular user
sth_gp_getActivity(activityId)	Get the details of an activity
sth_gp_searchActivities(query)	Search public activities
sth_gp_getActivityComments(activityId)	List all of the comments for an activity.
sth_gp_getComment(commentId)	Get the details of a comment
sth_gp_getPerson(personId)	Get a person's profile.
sth_gp_searchProfiles(query)	Search all public profiles
sth_gp_getActivityPeople(activityId)	List all of the people for a particular activity

11.4 YouTube

Method	Description
sth_ytb_getVideo(videoId)	Retrieves information about a single video
sth_ytb_addVideoRating(videoId, rating)	Adds a rating to a video
sth_ytb_getVideoComments(videoId) sth_ytb_addVideoComment(videoId,cmnt)	Returns a list of comments for a given video Add a comment to the given video
sth_ytb_getVideoResponses(videoId) sth_ytb_addVideoResponse(videoId,response)	Returns a list of video responses for a given video / Adds a video response to the given video
sth_ytb_getRelatedVideos(videoId)	Returns a list of related videos for a given video
sth_ytb_getUserProfile(userName)	Returns a user's profile
sth_ytb_getUserSubscriptions(userId) sth_ytb_addUserSubscription(userId, object)	Returns a list of the subscriptions created by a user Creates a subscription
sth_ytb_getUserUploads(userId)	Returns a feed containing all of the videos uploaded by a specific user
sth_ytb_getUserLiveEvents(userId)	Retrieves a feed listing a specific user's live-streaming events
sth_ytb_getUserEvents(userId)	Returns a user's activity feed
sth_ytb_getUserEvents(set⟨userId⟩)	Returns an activity feed for one or more users
sth_ytb_getUserFriendEvents(userId)	Returns an activity feed for the logged-in user's friends

11.5 Instagram

Method	Description
sth_igm_getUser(userId)	Get basic information about a user.
sth_igm_getSelfFeed()	Get the authenticated user's feed.
sth_igm_getRecentMedia(userId)	Get the most recent media published by a user.
sth_igm_getSelfMediaLikes()	Get the authenticated user's list of media they've liked.
sth_igm_getUserFollows(userId)	Get the list of users this user follows.
sth_igm_getUserFollowedBy(userId)	Get the list of users this user is followed by
sth_igm_getMedia(mediaId)	Get information about a media object.
sth_igm_getPopularMedia()	Get a list of what media is most popular at the moment.
sth_igm_searchMedia(query)	Search for media objects
sth_igm_getMediaComments(mediaId) sth_igm_addMediaComment(mediaId, cmnt)	Get a full list of comments on a media Create a comment on a media
sth_igm_getMediaLikes(mediaId) sth_igm_addMediaLike(mediaId, like)	Get a list of users who have liked this media Set a like on this media by the currently authenticated user
sth_idm_getRecentMedia(tagName)	Get a list of recently tagged media.
sth_igm_searchTags(query)	Search for tags by name.
sth_igm_getTag(tagId)	Get information about a tag object.
sth_idm_getLocationMedia(locationId)	Get a list of recent media objects from a given location.
sth_idm_getLocation(locationId)	Get information about a location.
sth_igm_searchLocations(query)	Search for a location by name and geographic coordinate.
sth_idm_getSubscriptions() sth_idm_addSubscription(subscription)	Returns your current subscriptions Creates a new subscription

11.6 LinkedIn

Method	Description
sth_lin_getBasicProfile()	Retrieve basic profile data for the authenticated user
sth_lin_getCompanyUpdates(companyId)	Get a company's updates
sth_lin_getComments(updateId)	Get comments for a specific company update
sth_lin_getLikes(updateId)	Get likes for a specific company update
sth_lin_getCompanyProfile(companyId)	Get a company's profile (expanded)
sth_lin_getCompanyFollowers(companyId)	Get a company's followers, by segment
sth_lin_getCompanyStats(companyId)	Get statistics for a company page
sth_lin_addShare(data)	Share via a comment containing a URL or with specific values

11.7 Pinterest

Method	Description
sth_pin_getBoard(boardId)	Retrieve information about a Board
sth_pin_getBoardPins(boardId)	Retrieve the Pins on a Board
sth_pin_getUser(userId)	Return a user's information
sth_pin_getUserBoards()	Return the logged in user's Boards
sth_pin_getUserBoardFollows()	Get the Boards that the logged in user follows
sth_pin_followBoard(boardId)	Follow a Board
sth_pin_getUserFollows()	Return the users that the logged in user follows
sth_pin_followUser(userId)	Follow a user
sth_pin_searchBoards(query)	Search the logged in user's Boards
sth_pin_searchPins(query)	Search the logged in user's Pins
sth_pin_addPin(data)	Create a Pin
sth_pin_getPin(pinId)	Return information about a Pin

11.8 Reddit

Method	Description
sth_red_getLinkComments(linkId)	Returns information about a link, with comments, given its id
sth_red_getUserDetails(userName)	Retrieves information from a user's "about me" page, including karma totals
sth_red_getUserReddits()	Returns information about the subreddits the currently logged-in user subscribes to.
sth_red_addStory(data)	Submits a link/story.
sth_red_vote(objId, vote)	Casts or rescinds a vote on a story/comment.
sth_red_getStoryTitle(url)	Returns the title for a story, given its URL.
sth_red_marknsfw(objId)	Marks a thing as NSFW.
sth_red_unmarknsfw(objId)	Unmarks a thing as NSFW.
sth_red_savePost(postId)	Saves a post
sth_red_composeMessage(data)	Handles message composition
sth_red_getMessages()	Retrieves messages.
sth_red_addComment(data)	Posts a comment
sth_red_searchSubreddits(query)	Searches for subreddits with the given query.
sth_red_getSubredditFlairs(redId)	Returns the flair assignments of a subreddit.
sth_red_setSubredFlair(redId, flair)	Sets or clears a user's flair in a subreddit.

11.9 Tumblr

Method	Description
sth_tbr_getBlogInfo(blogId)	This method returns general information about the blog, such as the title, number of posts, and other high-level data.
sth_tbr_getBlogFollowers(blogId)	Retrieves a blog's followers.
sth_tbr_getBlogPosts(blogId)	Retrieves published posts.
sth_tbr_newBlog(data)	Creates a new blog post
sth_tbr_reblog(postId)	Reblogs a post.
sth_tbr_getUserLikes()	Retrieve the liked posts of the authenticated user
sth_tbr_getUserFollowing()	Retrieve the blogs followed by the authenticated user
sth_tbr_followBlog(blogId)	Follow a blog
sth_tbr_unFollowBlog(blogId)	Unfollow a blog

11.10 Flickr

Method	Description
sth_fkr_getUserComments()	Returns a list of recent activity on photos commented on by the calling user. Do not poll this method more than once an hour.
sth_fkr_getUserPhotos()	Returns a list of recent activity on photos commented on by the calling user. Do not poll this method more than once an hour.
sth_fkr_getBlogList()	Get a list of configured blogs for the calling user.
sth_fkr_postBlogPhoto(blogId, photo)	Posts a photo to a blog.
sth_fkr_getCollectionIfo(collId)	Returns information for a single collection. Currently can only be called by the collection owner, this may change.
sth_fkr_getContacts()	Get a list of contacts for the calling user.
sth_fkr_addPhoto(galleryId, photo)	Add a photo to a gallery.
sth_fkr_createGallery(data)	Create a new gallery for the calling user.
sth_fkr_getGallery(id)	Returns information about a gallery.
sth_fkr_getGalleries()	Return the list of galleries created by a user. Sorted from newest to oldest.
sth_fkr_getGalleryByPhoto(photoId)	Return the list of galleries to which a photo has been added. Galleries are returned sorted by date which the photo was added to the gallery.
sth_fkr_getGalleryPhotos(galleryId)	Return the list of photos for a gallery.
sth_fkr_getGroupInfo(groupId)	Get information about a group.
sth_fkr_searchGroups(query)	Search for groups. 18+ groups will only be returned for authenticated calls where the authenticated user is over 18.
sth_fkr_getGroupMembers(groupId)	Get a list of the members of a group. The call must be signed on behalf of a Flickr member, and the ability to see the group membership will be determined by the Flickr member's group privileges.
sth_fkr_addGroupPhoto(groupid, photo)	Add a photo to a group's pool.
sth_fkr_removeGroupPhoto(groupid, photo)	Remove a photo from a group pool.

sth_fkr_getInterestingness()	Returns the list of interesting photos for the most recent day or a user-specified date.
sth_fkr_findUserByEmail(email)	Return a user's NSID, given their email address
sth_fkr_findUserByUsername(userName)	Return a user's NSID, given their username.
sth_fkr_getUserInfo(userId)	Get information about a user.
sth_fkr_getUserPhotos(userId)	Return photos from the given user's photostream. Only photos visible to the calling user will be returned.
sth_fkr_getPhotosOf(userId)	Returns a list of photos containing a particular Flickr member.
sth_fkr_getUserPublicGroups(userId)	Returns the list of public groups a user is a member of.
sth_fkr_getUserPublicPhotos(userId)	Get a list of public photos for the given user.
sth_fkr_addTags(photoId, tags)	Add tags to a photo.
sth_fkr_getPhotoContexts(photoId)	Returns all visible sets and pools the photo belongs to.
sth_fkr_getContactsPhotos(userId)	Fetch a list of recent photos from the calling users' contacts.
sth_fkr_getContactsPublicPhotos(userId)	Fetch a list of recent public photos from a users' contacts.
sth_fkr_getPhotoEXIF(photoId)	Retrieves a list of EXIF/TIFF/GPS tags for a given photo. The calling user must have permission to view the photo.
sth_fkr_getPhotoFavorites(photoId)	Returns the list of people who have favorited a given photo.
sth_fkr_getPhotoInfo(photoId)	Get information about a photo. The calling user must have permission to view the photo.
sth_fkr_getRecentlyUpdatedPhotos()	Return a list of your photos that have been recently created or which have been recently modified. Recently modified may mean that the photo's metadata (title, description, tags) may have been changed or a comment has been added (or just modified somehow :-)
sth_fkr_searchPhotos(query)	Return a list of photos matching some criteria. Only photos visible to the calling user will be returned. To return private or semi-private photos, the caller must be authenticated with 'read' permissions, and have permission to view the photos. Unauthenticated calls will only return public photos.
sth_fkr_addPhotoComment(photoId, cmnt)	Add comment to a photo as the currently authenticated user.
sth_fkr_getPhotoComments(photoId)	Returns the comments for a photo.
sth_fkr_getPhotoGeolocation(photoId)	Get the geo data (latitude and longitude and the accuracy level) for a photo.
sth_fkr_getPhotoPersons(photoId)	Get a list of people in a given photo.
sth_fkr_getPhotoSet(id)	Gets information about a photoset.
sth_fkr_addPhotoSet(data)	Create a new photoset for the calling user.
sth_fkr_addPhotoToSet(photoId, setId)	Add a photo to the end of an existing photoset.
sth_fkr_getPhotoSetComments(photoSetId)	Returns the comments for a photoset.
sth_fkr_addPhotoSetComment(photoSetId, cmnt)	Add a comment to a photoset.
sth_fkr_findPlace(query)	Return a list of place IDs for a query string.
sth_fkr_findPlaceByLatLon(lat, lon)	Return a place ID for a latitude, longitude and accuracy triple.
sth_fkr_getPlace(placeId)	Get information about a place.
sth_fkr_getPlaceByUrl(placeId)	Lookup information about a place, by its flickr.com/places URL.
sth_fkr_getTopPlaces()	Return the top 100 most geotagged places for a day.
sth_fkr_getPlacesFoUser(userId)	Return a list of the top 100 unique places clustered by a given placetype for a user.
sth_fkr_getTagsForPlace(placeId)	Return a list of the top 100 unique tags for a Flickr Places or Where on Earth (WOEID)
sth_fkr_getPhotoStats(photoId)	Get the number of views, comments and favorites on a photo for a given date.
sth_fkr_getPopularPhotos()	List the photos with the most views, comments or favorites.
sth_fkr_getTotalViews(userId)	Get the overall view counts for an account.

11.11 VKontakte

Method	Description
sth_vk_getUser(userId)	Returns detailed information on users.
sth_vk_getFollowers(userId)	Returns a list of IDs of followers of the user in question, sorted by date added, most recent first.
sth_vk_getNearbyUsers(lat, lon, radius)	Indexes current user location and returns nearby users.
sth_vk_getUserSubscriptions(userId)	Returns a list of IDs of users and public pages followed by the user.
sth_vk_searchUsers(query)	Returns a list of users matching the search criteria.
sth_vk_getAppFriendsList()	Return friends list for requests and invites in current app.
sth_vk_getBoardComments(topic_id)	Returns a list of comments on a topic on a community's discussion board.
sth_vk_addBoardComment(topic_id, cmnt)	Adds a comment on a topic on a community's discussion board.
sth_vk_addTopic(data)	Creates a new topic on a community's discussion board.
sth_vk_userIsFriendWith(set<user_id>)	Checks the current user's friendship status with other specified users. Also returns information specifying whether there is an outgoing or incoming friend request (new follower).
sth_vk_getFriendLists(userId)	Returns a list of the user's friend lists.
sth_vk_getMutualFriends(userId, targetUserId)	Returns a list of user IDs of the mutual friends of two users.
sth_vk_getOnlineFriends(userId)	Returns a list of user IDs of a user's friends who are online.
sth_vk_searchFiends(query)	Returns a list of friends matching the search criteria.
sth_vk_getUserGroups(userId)	Returns a list of the communities to which a user belongs.
sth_vk_getGroupMembers(groupId)	Returns a list of community members.
sth_vk_joinGroup(groupId)	With this method you can join the group or public page, and also confirm your participation in an event.
sth_vk_groupInvite(groupId, userId)	Allows to invite friends to the community.
sth_vk_getGroup(groupId)	Returns information about communities by their IDs.
sth_vk_getGroupInvites(groupId)	Returns a list of invitations to join communities and events.
sth_vk_getLikes(type, objId)	Returns a list of IDs of users who added the specified object to their Likes list.
sth_vk_isLiked(userId, type, objId)	Checks for the object in the Likes list of the specified user.
sth_vk_addLike(userId, type, objId)	Adds the specified object to the Likes list of the current user.
sth_vk_getNewsfeed()	Returns data required to show newsfeed for the current user.
sth_vk_getNewsfeedComments(filters)	Returns a list of comments in the current user's newsfeed.
sth_vk_getFollowedNewsfeeds()	Returns a list of newsfeeds followed by the current user.
sth_vk_searchNewsfeed(query)	Returns search results by statuses.
sth_vk_unsubscribeFromNewsfeed(feedId)	Unsubscribes the current user from specified newsfeeds.
sth_vk_getPage(pageId)	Returns information about a wiki page.
sth_vk_getPhoto(photoId)	Returns information about photos by their IDs.
sth_vk_getPhotoComments(photoId)	Returns a list of comments on a photo.
sth_vk_getPhotoTags(photoId)	Returns a list of tags on a photo.
sth_vk_addPhotoToWall(objId, photo)	Saves a photo to a user's or community's wall after being uploaded.
sth_vk_getUserPhotos(userId)	Returns a list of photos in which a user is tagged.

sth_vk_addPhotoTag(photoId, tag)	Adds a tag on the photo.
sth_vk_addPhotoComment(photoId, cmnt)	Adds a new comment on the photo.
sth_vk_searchPhotos(query)	Returns a list of photos.
sth_vk_addPlace(place)	Adds a new location to the location database.
sth_vk_getPlace(placeId)	Returns information about locations by their IDs.
sth_vk_checkin(placeId)	Checks a user in at the specified location.
sth_vk_getCheckins(userId, params)	Returns a list of user check-ins at locations according to the set parameters.
sth_vk_searchPlaces(query)	Returns a list of locations that match the search criteria.
sth_vk_addPoll(poll)	Creates polls that can be attached to the users' or communities' posts.
sth_vk_getPollVotes(pollId)	Returns a list of IDs of users who selected specific answers in the poll.
sth_vk_getPoll(pollId)	Returns detailed information about a poll by its ID.
sth_vk_addPollVote(pollId, vote)	Adds the current user's vote to the selected answer in the poll.
sth_vk_getStatus(userId, groupId)	Returns data required to show the status of a user or community.
sth_vk_setStatus(groupId, text)	Sets a new status for the current user.
sth_vk_getVideo(set⟨videoId⟩)	Returns detailed information about videos.
sth_vk_addVideo(data)	Adds a video to a user or community page.
sth_vk_getVideoComments(videoId)	Returns a list of comments on a video.
sth_vk_searchVideos(query)	Returns a list of videos under the set search criterion.
sth_vk_getWall(wallId)	Returns a list of posts on a user wall or community wall.
sth_vk_getWallComments(postId)	Returns a list of comments on a post on a user wall or community wall.
sth_vk_pinWall(postId)	Pins the post on wall.
sth_vk_searchWall(userId)	Allows to search posts on user or community walls.
sth_vk_addWallPost(userId, message)	Adds a new post on a user wall or community wall. Can also be used to publish suggested or scheduled posts.
sth_vk_repostOnWall(object, groupId, userId)	Reposts (copies) an object to a user wall or community wall.

11.12 Analytics

Method	Description
sth_ana_getSentiment(message)	Computes the sentiment of the message (Positive, Negative or Neutral)
sth_ana_getSentiment(dict, message)	omputes the sentiment of the status message (Positive, Negative or Neutral) based on a given dictionary of words and phrases together with their impact in the sentiment arithmetics
sth_ana_getClusters(set<usersWithFriends>)	Computes communities of users and their friends
stn_ana_getReach(obldj, network)	Calculates the reach of a given object on the target social network
sth_ana_extractGraph(network, objectId)	Generates a graph of objects and relationships for a given object from a given social network
sth_ana_getBehaviour(query, graph)	Gives the results of a custom Cypher Query executed on a given graph
sth_ana_getInfluencers(graph, params)	Returns the influencers of a certain group based on parameters like relationship constraints, entity types constraints etc.
sth_ana_getPageRank(graph)	Computes pagerank for all nodes in the graph
sth_ana_getBetweennessCentrality(graph)	Computes betweenness centrality for all nodes in the graph
sth_ana_getDegreeCentrality(graph)	Computes degree centrality for all nodes in the graph
sth_ana_findPath(objId, targetObjId)	Returns the nodes that must be traversed to reach the targetObject, starting from objId
sth_ana_getHeatMap(graph)	Returns clusters of features and their temperature on the given graph
stn_ana_getGeoClusters(graph)	Returns clusters of features on the given graph
sth_ana_extractEntities(objId, network)	Extract Named Entities from the target objectId found on a social network. Named Entities can be Locations, Persons, Organizations, Emails, Phone Numbers, Zip Codes and IP addresses.
sth_ana_newWatchlist(set<objId>, notification-Type)	Creates a watchlist from the given social network objects and choose how to receive notifications when monitored objects change.
sth_ana_addToWatchlist(listId, objId)	Adds a new object to a watchlist
sth_ana_removeFromWatchlist(listId, objId)	Removes an object from a watchlist
sth_ana_createKeywordMonitor(set<words>, networks)	Creates a list of keywords to monitor on the selected target social networks
sth_ana_addKeyword(monitorId, keyword)	Adds a new word to an existent Monitor list
sth_ana_removeKeyword(monitorId, keyword)	Removes a keyword from an existent Monitor list

SETHER

MEET THE COMPANY

MWARE is a company founded in 2012 and has enjoyed a rapid expansion, reaching now over 50 employees. It started as a tehchnology company focusing of middleware stacks and strategically moved into the AI and big data sector three years ago.

Our Big Connect big data platform and our Sponge data collection platform are used throughout the world by many key stakeholders to gather and analyze structured and unstructured data from all immaginable sources.

We have successfully establised partnerships with all major big data technology leaders and have delivered projects to government agencies, intelligence and defense structures, banking, telecom and major retail customers.

We took a strategic decision to outsource part of our technology stack for the benefit of the blockchain community, to build an amazing social development platform for all blockchain developers.

We believe that the blockchain will have a major impact on how humanity will evolve and we aim to be a key player in the area.

Check out our big data solutions on http://www.m-ware.eu

Ovidiu Oancea
CEO

ovidiu.oancea@m-ware.eu

Flavius Burca
CTO

flavius.burca@m-ware.eu

Cristi Savescu
COO

cristi.savescu@m-ware.eu

MWARE SOLUTIONS
Str Popa Savu, nr 77, Bucharest, Romania
+40 31 428 2612

contact@sether.io
investment@sether.io

sether.io