dock

Decentralized data exchange powered by Ethereum

Dock Protocol V0.5 | Subject to change
March 12, 2018

Table of Contents

1 Abstract

In recent years centralized platforms have revolutionized the way individuals find jobs and employment, build up a reputation, and establish professional networks. As a result, these platforms retain ownership of massive amounts of professionals' data and information that only exist within these closed networks.

Applications are disincentivized from sharing information and data because of the competitive marketplace and their reliance on data monetization for business purposes. Each platform wants to hold an advantage over its competitors.

The Dock protocol aims to solve this fundamental data hoarding problem by implementing a protocol that actually encourages data exchange between platforms.

By utilizing the secure Ethereum mainchain, data encryption, as well as a token model, the Dock protocol will allow users to take control of their data and exchange it between applications. The Dock token model will simultaneously motivate applications to exchange information by making it beneficial for both platforms involved in the transaction.

Both the users and the applications primary needs are met. Users get full control of their data and applications receive compensation for data exchange.

2 Summary

The Dock protocol is a special purpose decentralized data exchange protocol. This protocol is intended to incentivize the exchange of work experience, reviews and professional connections, and can be expanded to include any type of shareable data. In addition, the tokens serve as the proposal and voting utility to guide the evolution of development and changes to the Dock protocol. All token holders can contribute proposals and vote on changes and updates they'd like to see implemented. Applications receive tokens for data exchange. Users will have absolute control over their data through the use of a hosted third party service.

2.1 Problem Definition

The internet has changed the job market forever. The average professional person changes jobs 12 times during his or her career in the US. Freelancing as a profession grows every year. Currently, 35% of the current US workforce perform as freelancers, and this percentage is projected to accelerate to 40% by 2020.

Professionals rely more than ever on centralized platforms to find work opportunities and sustain their careers. This $200 billion industry is controlled by data monopolies including LinkedIn, Upwork, Glassdoor, and others. These platforms centralize user data, making it impossible for individuals to transfer their hard-earned experiences, reviews, and ultimately their value. As a result, the industry provides fragmented experiences, and the internet becomes less connected.

In previous years, LinkedIn provided an open API. 30,000 apps were built off this API and innovation flourished. In February 2015, LinkedIn made an announcement stating that it would restrict public API access. Consequently, many businesses have died and industry innovation has slowed.

We strongly believe users should own their personal data and be able to easily share it across applications. We believe in innovation and in aiding companies in pursuing this. We believe in a connected internet, the power of technology, and its potential global impact in improving lives.

3 High Level Overview and Vision

3.1 Users

Anyone can use the Dock protocol and easily port their data to and from participating applications. Users have access to automatically updated profile information, employment history data, and any type of platform data in an integrated fashion. Hence, they will not be bound to any single platform for their employment data, freelancing, or personal reviews.

Users will possess signed and verified content and choose where it is shared, thus owning their own data. They will also choose where to share the data received from other platforms in their name. It is up to the user to opt-into sharing any particular data with any specific application of their choosing.

3.2 Applications: Data Providers and Consumers

Any application can be a data provider. Data providers are special on-chain non-user accounts which can request to push signed data to the users' profile. It is important to note that the application does not publish data directly to the blockchain, but rather sends it to each user individually. Examples of such data would be CV updates, platform reviews and ratings, freelance transactional data, Git commits, application specific data, and so on.

For an application to be able to consume data from the user, it requires spending tokens to access the data. The alignment of interests between the user and the application is achieved via the Dock token model. [SEE SECTION 17]

3.3 Proposal and Voting Mechanism

Community participation through voting and proposal creation is crucial for the continued growth and development of Dock as a mechanism to make iterative changes and updates to the Dock protocol and network. This voting mechanism enables Dock to engage the community and user base in maintaining a system that evolves according to the needs of its users.

4 Possible Future Use Cases

When considering how much value LinkedIn as a single data silo has captured, it is easy to imagine how much more valuable an open data protocol can be. A protocol that combines, without any friction, a plethora of different types of data relevant to helping people find the right type of work, Dock will, therefore, help create future applications that are not possible today. These existing applications all have the same problem: data sharing. Dock solves this problem by aligning the interests of work-related data sharing through a crypto economic model while making the technical integration as simple as possible.

4.1 Automatic Relationship Mapper

Current applications need to be able to observe any type of interaction between users within their own application to map the relationship as a social graph or contact list. The Dock protocol makes it possible for a third-party application to receive push updates from a user's interactions on another application. In this case, the user acts as a sovereign data broker between applications. This gives the user full control over their own data and information. The third-party application can, in turn, update a state, such as upgrade a relationship in a social graph data format from an acquaintance to a colleague, based on external application data. This data, in turn, can be pushed back to the user.

4.2 In-Work Application Performance Mapper

Another example of a third-party integration could be the current business applications such as task managers, collaborative document writers, or even file storage applications, pushing interactions to the user through the Dock protocol. This interaction of based work-related data can, in turn, be used selectively by third-party applications to assess individual work history and performance. Third-party applications can create reputation scores, network centrality scores, performance reviews, or even vouch for whether an individual works at a specific company.

5 Usage of Ethereum (Reasons)

The Ethereum main chain was chosen as the preferred chain for the Dock protocol to run on due to several benefits:

1. Network effect and strong ecosystem.
2. Multiple good scaling roadmaps.
3. Easy integration with third-party smart contracts and applications.
4. ERC20[1] token standard and easy integration with wallets and exchanges.

[1] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md

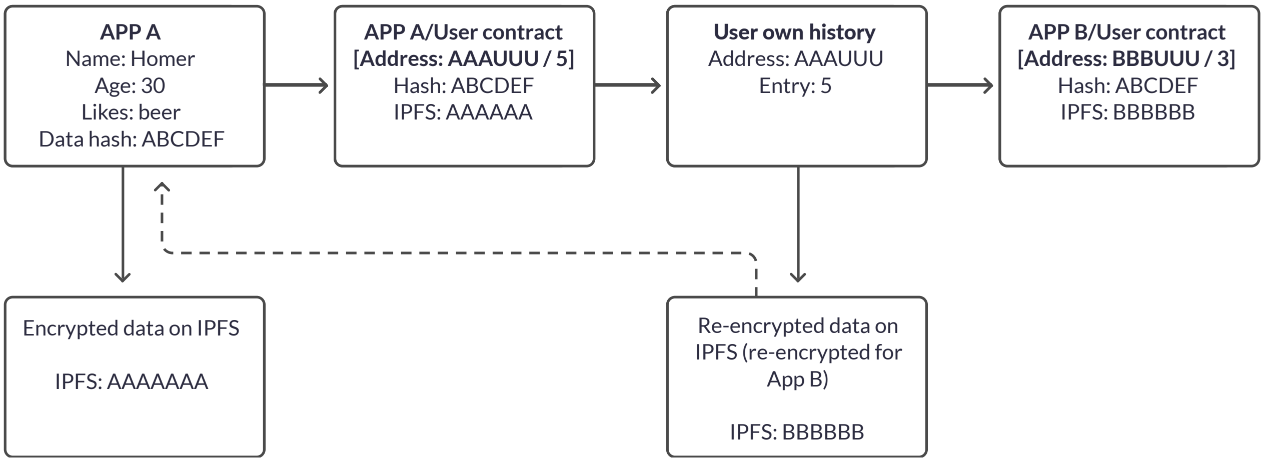
6 Core Dock Protocol Features

6.1 Data Exchange Model

Dock uses the IPLD[2] specification developed for IPFS to perform content addressable data exchange. Furthermore, users form an ad-hock publish-subscribe (pubsub) network between each other.

All content is IPLD addressable with the content within being by default encrypted with the receivers on-chain stored public key. There is no need for a direct exchange of the packet between the user and the application because the encrypted data packages are also partially replicated by other nodes. These nodes cannot decipher what is stored in the packages as they have been encrypted with the destiny application's public key. IPFS makes it possible to discover and resolve the integrity of these packages without the originator of the package needing to be online. If the originator is online, however, the discovery of the data should be considerably faster. Originators of the data need to be online only if the data being pushed by them is fresh and has never been published to the network before.

6.2 Encrypted Secure Snapshots of Data



All data is by default encrypted, and it is only decrypted by the parties exchanging the data. The user selectively decides to perform push updates for any data format to a specific application by first encrypting the content with the receiver's public key. Encryption and versioning happen in the following order:

1. The user stores the snapshot hash of the data created or shared by an application and pushed to the user. The application always wants to have the latest version of the data. Snapshot hashes make sense because it is possible to tie this data to a specific time in the blockchain without needing the entire history to have useful data.

[2] https://ipld.io/

2. The unencrypted snapshot is used to create a hash. This hash includes the previous version of the hash as well as the most recent Ethereum transaction and the block hash of the previous snapshot hash.
3. The user takes the snapshot hash and encrypts it with the application's public key and includes the IPFS addressable hash of the data in the application-user pair smart contract.
4. The application observes the Dock Ethereum application-user pair contract and recognizes that a package is addressed to it. The application requests the encrypted package over IPFS from the user or an intermediary node.

6.3 Voting Utility

The voting protocol is dependent on IPFS where the data is stored and will hold the hash of the IPFS data store. The data stored includes the proposal, the answer choices, the start and end time, author and vote identifiers. MetaMask will be used to verify whether someone is qualified to cast a vote based on the number of dock tokens held in the wallet and the contract will calculate the number of votes that can be attributed to that person.

7 Reasoning Behind These Steps

Dock makes use of IPFS to be fully content addressable and as compatible as possible with other Ethereum ecosystem applications and smart contracts. As of this writing, IPFS adoption within Ethereum smart contracts is picking up and its trajectory indicates that it is going to become a standard soon.

Data is anchored to a newly created application-user pair contract to which only the specific application and user have write capability to. This contract is effectively an updatable, verifiable, symmetric, hash-exchange channel for the application and user. Snapshot versioning can be publicly verified through this channel.

By encrypting the updates to the different applications' public keys, the user remains in full control of who receives and can interpret the push updates, even though the IPFS hashes and, therefore, content are publicly available. The data itself is not exposed to the public in unencrypted form. The public only knows that a push update happened.

8 Possibility of Public Data

Users can opt to publish specific unencrypted data formats to IPFS directly. That way anyone can receive that data, validate its integrity, and verify whether it is the latest version, without having to make any data requests to the user. We envision that users will opt to publish some core insensitive data in this manner. Such data could include but is not limited to:

1. First name and last name

2. Location
3. Education
4. Licenses and Certifications
5. Public contact information such as work email

Users need to be aware that any data published in this fashion through the Dock protocol to IPFS is irreversibly on the internet and cannot be taken back.

9 Data Exchange Model

9.1 Chainpoint Modification

Dock utilizes the open-source standard for anchoring data to blockchains called Chainpoint[3]. Chainpoint was originally developed for the Bitcoin blockchain, and only for anchoring data to a single chain. Dock uses a modified specification of Chainpoint which enables not just anchoring updates of data formats to a single blockchain and including it in a single block, but, in addition, makes it possible to prove that the update was created at the time of a specific block as well. This is achieved by making the following Chainpoint blockchain receipt modifications (changes in gray):

@context	The JSONLD Context of the document
type	The type of Chainpoint Receipt being described
targetHash	The hash value (of the unencrypted snapshot) being anchored in hex string format
prevHash	The hash value of the previous hash of the blockchain transaction being referenced
currentBlock	Most current Ethereum block hash being referenced at the time of the update
merkleRoot	The merkle root of the tree in hex string format
proof	An array of hash objects connecting targetHash to merkleRoot
anchors	An array of methods employed to anchor data to blockchain(s)
Ipfs address	Address of the encrypted file

Chainpoint data of encrypted data being shared with specific applications is structured exactly the same.

[3] https://tierion.com/chainpoint

The user is the only party selectively sharing data with the application in the user-application contract. He/she is the only party that knows what in fact the latest update of the data is. The application can only know what the latest data is depending on what the user wants to share. It is important for the application receiving the encrypted data to know that it is receiving the latest version being tracked publicly on-chain. Once the application decrypts its update with its own key, it can use it freely. After sharing the data with an application, the user cannot control whether the application further shares the data with third parties.

By including the currentBlock hash we can establish the earliest that update that could have been generated by the user. It does not make it possible to define the exact time something was created. Ethereum block hashes are unpredictable but provable, any user can know with certainty that the update was not shared publicly before the time of the block creation.

The prevHash is the hash of the Ethereum transaction which included the previous connected update. This way any user can now prove that a certain update happened after another one, and is not a fork of an older update. The prevHash field is important for applications which do not permanently connect to the Ethereum main network but still want to share relevant anchoring data. Hence, when they decide to check the anchoring to Ethereum they can prove it at any given time.

For more information on the Chainpoint protocol standard see the Chainpoint white paper[4].

9.2 On-Chain Data

On-chain data is solely chainpoint anchoring data which includes:

- Chainpoint Versioning
- Content Hashes
- Merkle Roots
- Merkle Proofs
- Git Hashes
- Ethereum Dock Contract StatePossibly External ID Verifiers (In the case of civic, only predetermined)

No user data is stored in an encrypted or unencrypted form on-chain. No application can infer the content of the snapshot hashes from on-chain data. Applications can download the encrypted versioned on-chain data from the IPFS hash pointers. They won't be able to make sense of it unless the user creates a newly encrypted version specifically for that application. This is how encryption of updates is used for application data access control.

The user can choose to publish unencrypted updates and data to IPFS directly. Applications can then directly download those updates from IPFS without a prior permission from the user.

[4] https://github.com/chainpoint/whitepaper/blob/master/chainpoint_white_paper.pdf

9.3 Off-Chain Data

Off-chain data includes all data format information, including prior updates and snapshots, in both encrypted and unencrypted form. As long as the user opts to share data selectively with more than one application, the stored encrypted data should require more storage than the unencrypted data. Performing this client-side encryption and chainpoint anchoring to each relevant user-application contract requires both additional storage on IPFS as well as slightly higher computational resources on the client side. Nonetheless, for the majority of users, these operations and storage requirements should be met even on mobile devices.

Depending on semantics, all IPFS addressable data is off-chain, but the content hashes through which the data can be found are all published publicly and in hashed snapshot form on-chain. Any underlying data published to the IPFS network overall in this fashion is just as accessible as on-chain data.

10 User-Application Contract Pairs

Each user generates a new contract for keeping track of the status between him/herself and any application. This makes it possible to push separate updates to different applications all at once on-chain. Additionally, the attribution of who pushed what is very simple, and any 3rd party applications can easily identify which applications are interacting with any particular user, therefore proving that any data they might receive is, in fact, coming from that application or user.

Only the application and the user have write access to their mutual contract. This also makes it simple to keep track of the tit-for-tat data and payment exchange between applications and the user's contract with the application. Enforcement of the stepwise relationship becomes relatively trivial on the contract.

The main benefit is that we can accommodate Ethereum’s (reasonable) limitation of one state transition per block, per contract. By having user-application pair contracts, each party can initiate multiple state transitions/data updates per block.

11 On-Chain Data Attribution Verification

Applications receiving data from users will know if any data format was originally pushed from another application to the application-user contract. Applications can both sign the pushed content itself within the IPFS packet, as well as the version hash the user publishes to the user-application contract. Only when the data format within the IPFS package has been signed, the user can manually withhold the content signature of the application, however, in this case the application might not sign the version hash on-chain in the next round.

Applications can have various levels of verification requirements. Loose verification requirements might only require the data from the user. Mid-level requirements would require a content signature by the other application. Strong verification requirements would include on-chain signing of the version hash the user pushed to the blockchain.

12 On-Chain Vouching and Signaling

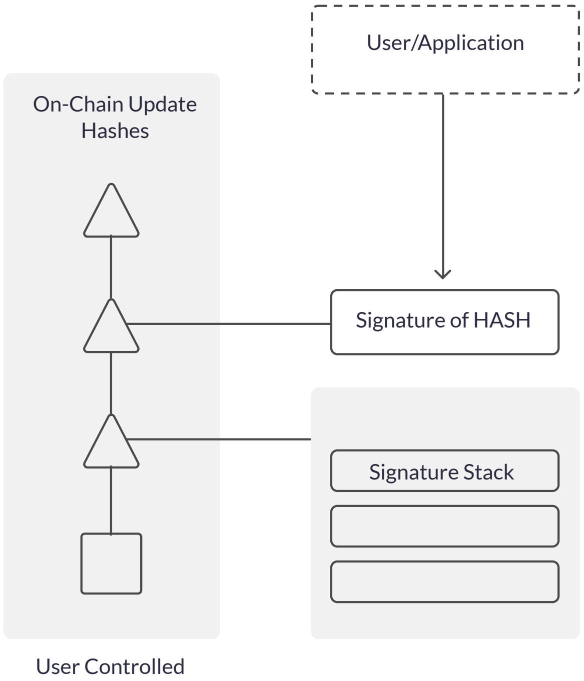
Any account can sign on-chain data format snapshot hashes of any other account. By doing so, one signals to the rest of the network that the content of that version hash is meaningful and correct. Vouching signatures created in this fashion appear in a unilateral user contract as a repository of vouchers by that user, while pointing to the signed transaction content. Use cases include, but are not limited to:

1. Employers vouching for the truthfulness of their employees' work history.
2. Schools vouching for the truthfulness of their students' credentials.
3. Students of a school vouching for the integrity of another student.
4. Identity services vouching for their citizens' identity information.
5. Applications vouching for the truthfulness of an application related data format.

Vouching itself does not verify the authenticity of the underlying data. Instead, a secondary trust/reputation layer needs to be used to assess how good a signal any individual signature is.

Users have no control over who can sign and publish signatures to the blockchain as any user can vouch for any published content and publish it on his/her contract. Signing and publishing of signatures of versions/data are fully at the discretion of the signing party.

For data compression purposes, only the Ethereum transaction hashes of the to be verified snapshot/data will be signed by any entity. This ensures signature output will remain both small enough and verifiable.



13 Public Name Registry

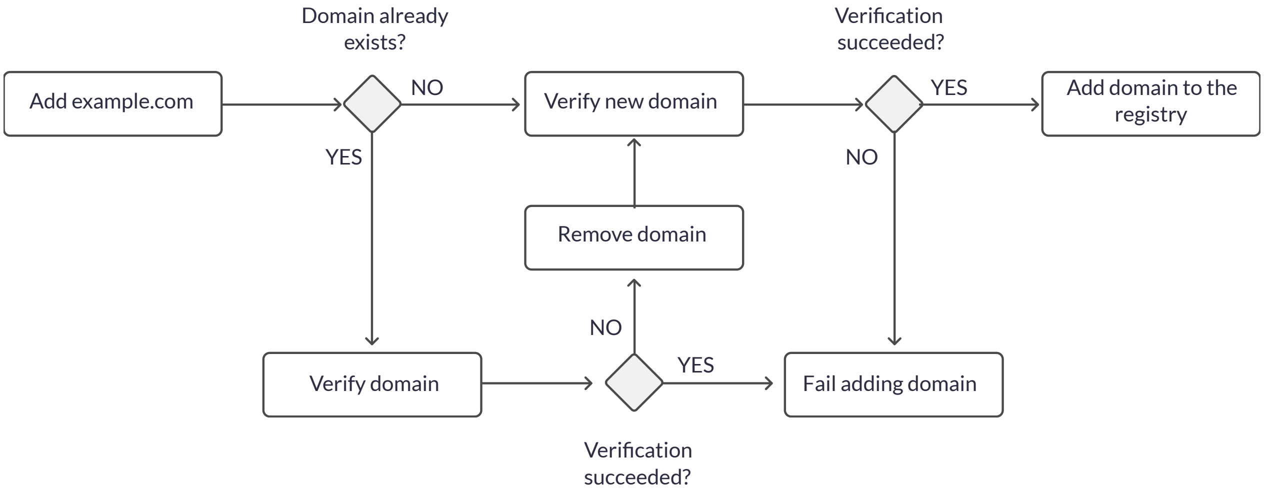
The public name registry will be the last part to be developed by Dock as we expect broader domain association methods to emerge soon in the Ethereum community. We will actively monitor the community for such developments and decide which method makes the most sense for domain name association with Ethereum addresses.

In order to facilitate simple application authentication of existing services and improve user experience, Dock will introduce an additional service to allow anyone to map the domain name to the Ethereum address. This public service will allow domain owners to associate their domain(s) with the Ethereum address they use to publish and sign data. Internally this will be a two-way mapping between the Ethereum address and a domain name. Internally the public name registry is an Ethereum smart contract with the well-known Ethereum address.

13.1 Name Verification

In order to verify domain names, one should first add a proper file under the domain namespace. For instance, if one claims that the domain example.com corresponds to the Ethereum address of "1234abcd", he/she should create a file under "example.com/1234abcd" with contents of "1234abcd". Next step would be to invoke from a given Ethereum address contract method with the desired domain name. This step involves the use of the oracle that will perform http or dns request resolution and validate domain authority.

There can be only one Ethereum address associated with the domain name. Thus, if one wants to change the address for the domain, he/she has to remove the original verification file first. Upon adding the new address, the registry contract will try to revalidate the existing domain and will remove it if verification fails.



There are several options on how data can be verified:

1. HTTP url: A file within the domain namespace with the same name and content as the Ethereum address
2. DNS record: A TXT record with the name and the value of the Ethereum address: "1234abcd.example.com. TXT 1234abcd"

More options can be added at a later time. It is possible to use more than one verification method for a domain. In this case, at least one valid check is required to consider the domain verified.

13.2 Reverse Address Lookup

Because mapping is two-sided, it is possible to lookup an Ethereum address for a given domain name. And because only verified domain names are stored in the registry, one can be sure that there will be only one valid domain stored.

14 Data Types and Standard Formats

Dock data formats are essentially the equivalent of microformats[5] for Ethereum smart contracts. Any type of data formats with any type of content is possible on Dock. The web microformats community already has a resume based microformat example that is universally machine readable[6]. Based on our team's background at remote.com, we will commence with the example of a resume and work history data format. The Dock proposed microformat model is content addressable via IPFS and anchored via the modified Chainpoint approach. Data formats are not prescribed in the Dock Ethereum smart contract. Users can choose to create, snapshot, share and use whatever data formats the community settles on. Data formats are only useful to users as long as a large number of applications accepts them.

14.1 Initial Format Types

Because mapping is two-sided, it is possible to lookup an Ethereum address for a given domain name. And because only verified domain names are stored in the registry, one can be sure that there will be only one valid domain stored.

Dock will initially create a number of ready-made data formats. All data formats are opt-in and voluntary for all users and applications. Theoretically, non-employment related data formats can be used with the Dock approach as well. Data format examples include, but are not limited to:

1. Employment history format (resume)
2. Social Graph (mutually signed contacts)

[5] http://microformats.org/wiki/Main_Page

[6] http://microformats.org/wiki/hresume

3. Licenses (relevant certifications/licenses)
4. Education (education history)
5. Application specific formats

All format data consists of key-value fields. Keys can be nested. It is up to the format data creator, such as the application, to determine whether the actual data is stored in JSON, XML, CSV, or another format.

Format types cannot be updated by a central party. This implies that if an application wants to alter something in the structure of a data format which it established previously, it cannot arbitrarily tell clients to change the format. The clients need to opt-into updating their format as well. This is partially why data format signaling is very important, otherwise, data formats would have too much power over the network. The greater the inertia to change over time, the larger the Dock network becomes.

We predict that clients will both do signaling as well as upgrade data formats semi-automatically. Ultimately stored data is only as useful as long as applications accept the format. We have refrained from including an auto-update format method. We have seen that historically this has been used by closed source API and software vendors as a way to keep 3rd party applications from relying on their format, while still being able to claim that their data/service is openly available.

14.2 Data Format Signaling

We believe that one major reason why digital formats take a long time to become industry standards is that users of these standards have no way of signaling to one another what they use or prefer. There is currently no good way to converge rapidly on a single preferred standard. Therefore, we introduce data format signaling as a convergence mechanism in Dock.

Applications which are also token holders are able to signal on their contracts which data formats they accept. This is done via a simple list with data format names. This is not a voting method, but rather a public signaling method. The signals are non-binding and do not have any effect on others in any way enforced by the contract. Other applications and users can scan the Ethereum chain and Dock application-user contracts for signals as well as for the number of tokens associated with their addresses. It is up to the applications and users to modify their signaling accordingly.

15 Snapshots of Individual Key-Value Pairs Within Format Data

The smallest atomic unit within a data format that can be versioned is a key-value pair. There is no upper bound to how large a set of these pairs can be. If a large number of pairs requires frequent updating then we advise to snapshot the individual pairs.

15.1 Rare Number of Updates

If the number of updates is sparse, then we advise to only snapshot the overall data formats and have the snapshot hashes be merkle roots. This way, the individual pairs are the merkle leafs. That would make it possible to selectively share down to individual pairs, while still verifying to any application that the pair is part of an up to date version hash on the Ethereum blockchain.

16 Experience Format Example

Tag	Data
Type	Experience event
Summary	Computer scientists
Employer	Remote.com, Inc.
Location	San Francisco: 37.7749° N, 122.4194° W
Start (UNIX time)	1511377952
End (UNIX time)	1577836800

Multiple experience formats combined can constitute an employment history format.

17 Token Model

Today, applications are naturally incentivized to hoard data about their users. User data is most often non-accessible to the user or third-party applications.

More malicious data strategies even include user data gathering, open ecosystem integration in return for data from third-party applications, and then shutting down third-party application data access. This is an example of Microsoft's internal strategy of "embrace, extend, extinguish[7]". Linkedin has also utilized similar tactics.

Even if an application was not malicious in this manner, other applications would take advantage of its data and outcompete them by not providing data in return. This is a common problem that can be solved with token-based modeling.

17.1 Long-term Trust Building

The token's utility allows applications to exchange any data with each other, with the user controlling sharing of their data. It makes long-term trust on the protocol possible. Users become brokers of their own data. Applications are incentivized to keep exchanging data with one another in the long term. Users can be reasonably sure who has which of their data as they opt-in to sharing specific data with specific applications.

17.2 Token Utility

Beyond the token being used as a means of exchange within the Dock contract system, it is also possible for token holders to vote in a regular poll on what next features should be developed for the Dock contract system.

This is important because Dock is intended to both give sovereignty over their data back to the user, as well as incentivize application ecosystem collaboration. The token is intended to solve the Commons problem of data sharing for the web as a whole. Therefore, adding a polling mechanism is a way to have on-chain governance of this new commons.

17.3 FIAT-Denominated Fee Oracle

Token costs, as well as any exit fees, will be denominated in FIAT. It is also possible that a future stable cryptocurrency could be used such as Basecoin, Dai etc., should one reach market maturity and reasonable long-term, real-world stability. This is important because users and applications will be able to predict their long-term costs. A side effect of this approach is that if the token price on the open market goes up, proportionally fewer tokens will be bought by users and applications. If the price goes down, more tokens will be bought by users and applications on the open market. In the long term this should give a lot of stability and predictability to the exchange rate of the token.

[7] http://www.economist.com/node/298112

17.4 Token Incentivization

Tokens will be used to incentivize applications to share their data with users and other applications. Users will have control over which applications can access and update their data. It is important to note that tokens will not be used to incentivize users to share data with applications.

It is imperative to motivate applications as opposed to users due to how data platforms currently operate. Applications do not share data because their peers and competitors can use it against them to create a competitive advantage. This is a major problem in today’s data landscape.

Once this is understood, it becomes clear that a token incentive model that encourages applications to share data, not hoard it, is absolutely necessary. In order for a model like this to work properly, the users must have full control of their own data.

A drawback of this is that users should not value their data in terms of the money they receive. The user data is almost always disproportionally more valuable to the application than to the user since applications create added value from it. If we simply created a marketplace where users can sell their data to applications for money, it would be a race to the bottom. Instead, in Dock users will value their data as is, but will not receive money in return for it. There will be a price for applications to pay, and applications which share data with the user will be paid, but not the user itself. This is a new type of market mechanism which we believe will be extremely important to test. This is Dock’s core contribution to cryptoeconomics.

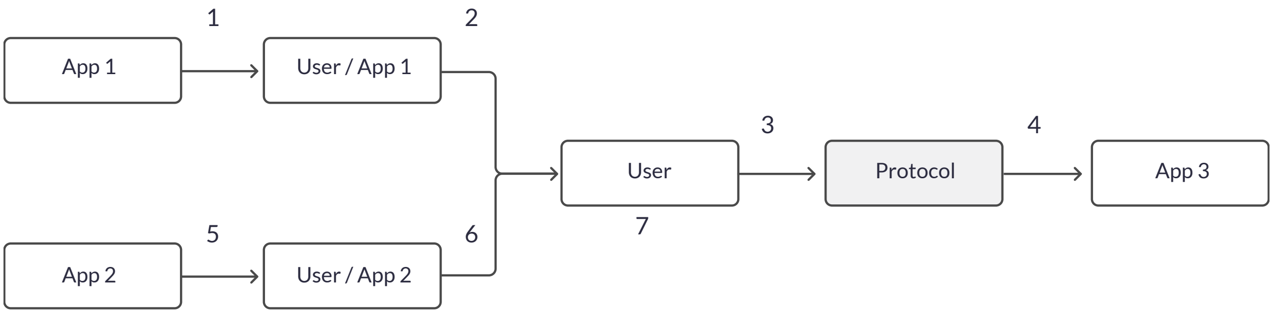
17.5 Reasoning Behind Not Including Users in the Data Exchange Economy

If tokens were used to encourage users to share their data, it could lead to various issues. Users could send out faulty data in an effort to maximize their profits. Additionally, users could also practice deceitful tactics like faking accounts and playing the system as a mean of attaining income.

Users should not be burdened with worrying about the value of their data. If users were included in the token economy, they would have to consider microtransactions and payments involving their data on a daily basis. By incentivizing applications and not users, the Dock protocol will make the user experience as easy as possible. This action will also discourage malicious tactics.

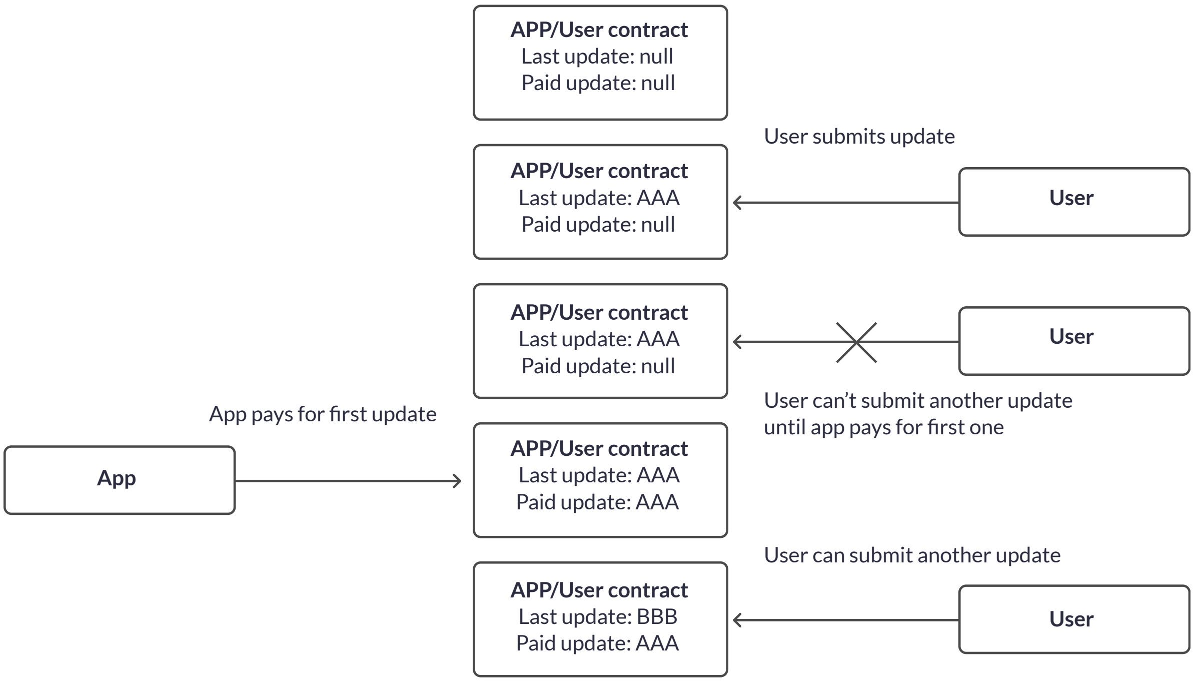
Applications will provide compensation to other applications for accessing users’ data. Users, however, will not pay applications for their own data. Instead, any time a user sends data that is attributed to another application, a payment will be triggered within the Dock protocol. This payment is not necessarily simply rerouted tokens. Rather, it might include re-creating burned tokens, without ever surpassing the original amount created initially.

For example, imagine a user sends data first created by Application A to Application B. The Dock protocol verifies this transaction and triggers a payment to be sent from Application B to Application A, which would be Application B's payment to access the data. When Application C accesses the same data, the tokens paid by Application C are burned or removed from the total supply of tokens on the Dock protocol.



1. App pays for consumed data
2. Contract proxies payment to user
3. User notifies protocol to pay for data usage by App 1
4. Protocol pays to the app that created data
5. Another app consumes same data
6. Contact proxies payment to user
7. User sees that this data was already consumed and discard the transaction

When a user is sharing his/her data, he/she submits an update to a smart contract designated for the receiving application. The contract is tracking hashes of the last updates submitted by the user as well as the hash of the last update that application consumed and paid for. The user can only push an update if the application has paid for the previous one. This way we can ensure that in order to continuously receive updates, application will be forced to pay tokens. This is a similar tit-for-tat relationship as in Bittorrent, but enforced by a smart contract. This also ensures that applications always consume all updates and have constant data updates.



17.6 Fee Model

Using this model, applications will indirectly pay other applications to access their data. However, applications will not be able to control which other applications receive this data. Users will remain in ultimate control of their own data.

The fee model for associated data exchange is not final and can be modified. The percentage of the payment for associated data is changeable through an on-chain percentage mechanism.

If an application pays a flat rate fee to a user for their data and no third-party application data is included, 100% of the fee would be burned and therefore it would be unusable.

17.7 Token Burn

There are a number of reasons why burning Dock tokens is an effective mechanism:

1. There should be a cost for applications to acquire user data.
2. Simultaneously, users should not receive those tokens as an incentive to avoid malicious behavior.
3. General token holders will benefit from the total supply of the outstanding tokens decreasing due to the burning mechanism.
4. Any percentage of token burn will act as an anti-spam and anti-DDoS mechanism for the smart contract.

17.8 Game Theoretic Effects

The Dock protocol represents a two-sided market[8]. However, Dock demonstrates a unique dynamic in that the user side of the marketplace does not have any monetary incentive to interact with the product or their own data.

The real value that users will receive from the system is ultimate control over their own data and data portability. This is a major benefit for users. No current major data platforms offer complete control over personal data. Since users will not be motivated by monetary gains, they will be impartial towards other applications and the quality of the data.

We believe this will lead to users choosing the best data providers and applications. Thus, the overall quality of data should increase during the cycle of the protocol.

17.9 Participation Through Voting

Users who are token holders of Dock will be able to participate in the evolution of the protocol and network by voting on updates to the protocol to ensure that the individual users are represented in the token mechanics.

[8] https://www.aeaweb.org/articles?id=10.1257/jep.23.3.125

When a new proposal is published, DOCK token holders will be able to use their tokens to enable their right to vote on the new proposal. Only token holders will be able to vote with each token representing one vote. A number of tokens will be required to submit to the application for the length of the voting period, at the end of which the results of the vote will be finalized based on the majority of votes. There will also be a minimum number of votes and participants required in order for the results of a vote to be finalized and the vote will be disqualified if there is less than the minimum amount of participation. It is important to mention that tokens are not locked during the voting.

18 Possible Issues and Hurdles

18.1 On-Chain Scaling and Transaction Flooding

The Dock protocol will need a relatively high transaction throughput for the smart contract to operate with such a large number of users and applications.

Currently, Ethereum can not support the required number of transactions for such a global system. The Ethereum community momentarily has scaling efforts intended to solve this problem. Regardless, any type of global state blockchain system will be prone to transaction flooding issues.

We have witnessed similar issues in the past. One example is fake transactions being generated on the Bitcoin network as an attack method.

Theoretically, the Dock protocol could experience a denial of service attack. The difference, however, is that the mainchain will require a number of tokens. Performing such an attack over a long period of time would prove to be extremely costly.

18.2 Need for ETH for Gas by Users

The Dock smart contract and protocol are reliant on the Ethereum mainchain. The Ethereum smart contracts on the mainchain rely on Ether to pay the gas requirement for running smart contracts.

Both users and applications need Ethereum and Dock tokens in order to use the Dock protocol. Applications need both in order to send and execute transactions. Users need both so they can send snapshot data of their latest user-related data.

This complicates matters for the users and applications and makes using the Dock protocol more difficult. These issues can be mitigated through the use of background automation, hosted wallets, and hosted services.

18.3 Transition to Standalone Chain

It is possible that the scaling efforts of the Ethereum mainchain do not produce adequate results. It is also possible that transactions on the Ethereum mainchain become too expensive for the average transaction moving forward. In this event, we reserve the right to create a new standalone global chain. This is something we are strongly against, however, it will be made use of in the case no other choice is left.

This chain would be dedicated solely to the Dock protocol and smart contract. Procedures will be put in place to ensure the transition is as smooth as possible.

The best method for such a transition would be a publicly known snapshot date. All applications and clients of the users would be upgraded to a client that switches over to the mainchain at a certain snapshot date.

18.4 Long-Con Game by Data Providers

Data providers can theoretically set up a large number of fake user accounts. These accounts would then be filled with meaningless data.

Applications may unknowingly pay for this faulty data. This would allow these data providers to steal tokens from other accounts. However, it would be quickly detected by other applications that the data is false, rendering this strategy useless. Applications would at some point detect that data coming from certain users is useless and would simply stop paying for updates.

These accounts would also have to go through regular registration/signup process in each application they want to steal tokens from. Many applications already have means to prevent such malicious behavior.

18.5 Token User Experience

End-users might not want to take on the cognitive load of buying a small amount of Ether, setting up their wallet and data application, and approving applications. This is required in order for the users to be sovereign and have control over their own data and tokens. Therefore, we believe that similar to Bitcoin mainstream adoption, the majority of users will opt into performing all these tasks automatically through a hosted third party service.

18.6 Centralized Exchange Rate Oracle Censorship

The centralized oracle providing token-FIAT life exchange rates to the Dock smart contract is a possible central failure point.

The following scenarios are technically possible:

1. The proxied exchange rate received by centralized exchanges is bad, and therefore the oracle relays bad data.
2. The oracle is down, making the Dock smart contract rely on the last known exchange rate for fee calculation.
3. The oracle is shut down by hackers or authorities.
4. The oracle is hacked and relays false exchange rate data.

18.7 Applications Can Share Decrypted Data

Applications can decrypt and share user-related data with applications not approved by the user. Once an application receives user-related data, there is no restriction in regards to how the data is used. The user can stop sharing new data with the application sharing with non-permitted applications. While users always opt-in to any data sharing, they can opt-out of that relationship at any time by simply not updating the user-application pair contract anymore.

This is both positive and negative. It can be beneficial, allowing users the option to share certain updates of basic data in a public manner. This means the snapshot hash which they commit to the Dock user-application pair contract would be committed to IPFS itself by the user in an unencrypted format.

This allows anyone to discover the data on IPFS itself and make use of it without having to ask permission, notify the user, or pay tokens. There are many application use cases that show this can be useful for the applications and the users.

However, this can also have a negative side. Wrongly configured clients could possibly leak data this way by publishing unencrypted data to IPFS.

Once a user publishes unencrypted data to the IPFS they can not return or delete the data. It is forever in the public domain. This could create issues for certain user groups.

19 Team

Nick Macario - Co-Founder
Previously founder of branded.me
Nick is a multi-time founder and technology executive with an exit. His previous company was branded.me, a professional networking site which grew to millions of users and was recognized for many awards and publications.

Elina Cadouri - Co-Founder
Previously Co-founder Outsource.com
Elina is a multi-time founder and veteran marketplace operator. Her previous company was Outsource.com, a freelance marketplace transacting tens of thousands of jobs and millions of dollars in revenue.

Stenli Duka - CTO
Previously @ MotiveMetrics
Stenli is a seasoned full-stack engineer and has led multiple teams. His passion is machine learning, natural language processing and complex algorithms. He previously spent time at Motive Metrics as the Director of Engineering.

Evgeniy Zabolotniy - Lead Blockchain Engineer
Previously @ branded.me
Evgeniy brings over 10 years of experience as a systems architect, developer and security specialist. Previously he worked with Nick at branded.me as part of the founding team, and has been a core team member for over 3 years.

Todd Scheuring - Head of Design
Previously @ Honeybook
Todd brings over 10 years of design and interactive development experience. Prior to Dock, he was a senior UX designer for Honeybook and led design for the network and growth teams.

Jeffry Harrison - Director of Partnerships
Previously @ The Muse
Jeffrey brings years of industry experience in both sales and recruiting. Prior to Dock he was a Team Lead at The Muse where he helped grow the sales team from 3 to over 40, and was responsible for some of the largest deals the company achieved.

Gabriel Moncarz - Data Scientist
Previously @ Sabre Corp
Gabriel brings over 10 years experience as a Data Scientist, Computer Engineer, Magister in Quantitative Finance and Magister in Data Mining, Big Data and Knowledge Discovery. Previously he was with Sabre Corp.

Fausto Woelflin - Senior Engineer
Previously @ Ampush
Fausto is a senior engineer who develops python-based microservices with Elasticsearch, Flask, Cassandra and Docker. He previously was with Ampush.

Piotr Swies - Blockchain Engineer
Previously @ Hunted Hive
Piotr is a software engineer, architect and full-stack developer with over 10 years of experience, skilled in designing and implementing distributed systems with microservices architecture. He was previously with Hunted Hive.

Sergey Ermakovich - Lead Frontend Developer
Previously @ branded.me
Sergey brings 10 years of experience, specializing in front-end web development, reactive functional programming (RFP), and UI architecture. Sergey comes from branded.me and has been with the team for over 3 years.

Samuel Hellawell - Senior Frontend Developer
Previously @ branded.me
Samuel is a senior engineer with a passion for developing quality applications, games and websites. He's started and sold his own company, and was part of the founding team at branded.me prior to Dock.

Maciek BodekSenior - Frontend Developer
Previously @ Shortlist.co
Maciek is a frontend engineer with ten years of experience crafting web products and complex interfaces ranging from portfolio microsites to agile SaaS startups.

20 External References

1. Vogelsteller, F., Buterin, V. (2015, November 19). *ERC-20 Token Standard* Retrieved from: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
2. *IPLD* Retrieved from: https://ipld.io/
3. Vaughan, W., Bukowski, J., Wilkinson, S. (2016, June 29). *Chainpoint: A scalable protocol for anchoring data in the blockchain and generating blackchain receipts* Retrieved from: https://tierion.com/chainpoint
4. *Git* Retrieved from: https://git-scm.com/
5. *Noms* Retrieved from: https://github.com/attic-labs/noms/blob/master/README.md
6. *Microformats Wiki* Retrieved from: http://microformats.org/wiki/Main_Page
7. King, R., Celik, T., Jones, G. *hResume* Retrieved from: http://microformats.org/wiki/hresume
8. *Deadly Embrace* (2000, March 30). Retrieved from: http://www.economist.com/node/298112
9. Bertani, T. *Understanding* Oracles (2016, February 18). Retrieved from: https://blog.oraclize.it/understanding-oracles-99055c9c9f7b
10. Rysman, M. (2009). *The Economics of Two-Sided Markets* Retrieved from: https://www.aeaweb.org/articles?id=10.1257/jep.23.3.125