Decentralized professional data exchange

powered by Ethereum

dock.io Protocol V0.3 | Subject to change

November 26th, 2017

Table of Contents

1 Abstract

In recent years centralized platforms have revolutionized the way individuals find jobs and employment, buildup 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 monetization of this data for business purposes. Each platform wants to hold an advantage over its competitors.

The dock.io protocol aims to solve this fundamental data hoarding problem. This will be accomplished by implementing a protocol that actually encourages data exchange between platforms.

By utilizing the secure Ethereum mainchain and a token model, the dock.io protocol will allow users to take control of their data and exchange it between applications. The dock.io 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.io 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. Applications will 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 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 that percentage is projected to accelerate to 40% by 2020.

Professionals rely more than ever on centralized platforms to find work opportunities and sustain 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 announced it would restrict public API access. As a result, many businesses died and industry innovation has slowed.

We strongly believe users should own their own data and be able to easily share it across applications. We believe in innovation and aiding companies in this pursuit. We believe in a connected internet, the power of technology, and the global impact it can have to improve lives.

3 High Level Overview and Vision

3.1 Users

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

Users will input signed and verified content and choose where it is shared, thus owning their own data. They will also choose where to share data received from other platforms in their name. It is up to the user to opt-into sharing any particular data format 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 is not publishing data directly to the blockchain, but sending 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, etc.

For an application to be able to consume data from the user, they are required spend tokens to access the data. The alignment of interests between user and application is achieved via the dock.io token model. [SEE SECTION 16]

4 Possible Future Use Cases

When we consider 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.io will therefore help create future applications that are not possible today. These applications all have the same problem: data sharing. dock.io solves this problem by aligning 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.io protocol makes it possible for a third party application to receive push updates from a user about 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 state, such as upgrade a relationship in a social graph data format from acquaintance to colleague, based on external application data. This data in turn will be pushed back to the user.

4.2 In-Work Application Performance Mapper

Another example of a third party integration could be current business applications such as task managers, collaborative document writers, or even file storage applications, pushing interactions to the user through the dock.io protocol. This interaction 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.io 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.io Protocol Features

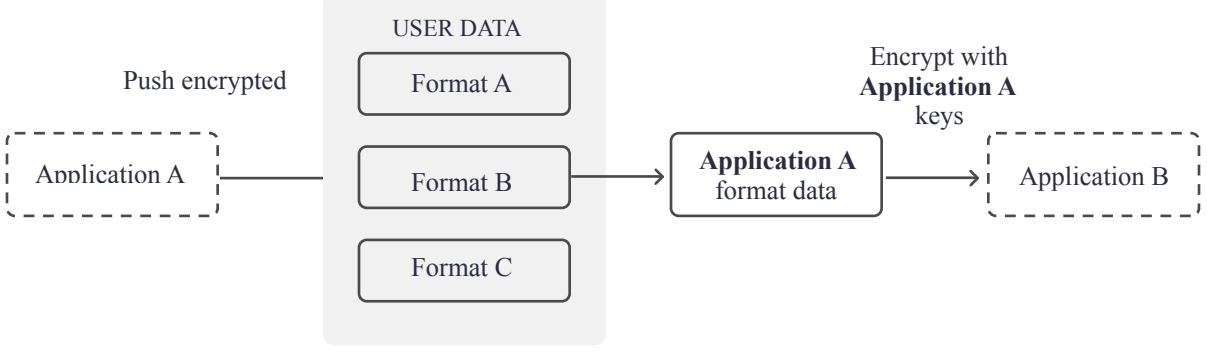
6.1 Data Exchange Model

dock.io uses the IPLD[2] specification developed for IPFS to perform content addressable data exchange.

6.2 Encrypted Secure Versioning of Data

All data is by default encrypted and 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 receivers public key. Encryption and versioning happens in the following order:

1. The user creates the delta of the data. The application always wants to have the latest version of the data. Delta's make sense because it is possible to tie this data to a specific time in the blockchain. Initial data sets without deltas would not offer that possibility, unless every time a new dataset is created it references the old version. Delta's might also reduce total bandwidth requirements for applications and nodes.
2. The unencrypted delta is used to create a hash. This hash includes the previous version of the hash as well as the most recent Ethereum block hash, the transaction, and the block hash of the previous version hash.
3. The hash of the unencrypted delta is committed as a transaction to the dock.io Ethereum Smart Contract special data format section of the user (e.g. Employment, Reviews, Git commits, education, etc).
4. The user takes the delta and encrypts it with the application's public key and includes the IPFS addressable hash of the data in the application update list section of the contract. This is part of the same transaction as before, it is needed to reduce the transaction requirement.
5. The application observes the dock.io Ethereum contract of the user and sees that a package is addressed to it. The application requests the encrypted package over IPFS from the user or an intermediary node.



[2] https://ipld.io/

7 Reasoning Behind These Steps

dock.io 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 should become a standard soon.

The unencrypted delta's of the data format updates are hashed, versioned, and anchored to the Ethereum blockchain. This is done to provide data integrity and make it possible for any application receiving data format updates from any user to know whether they truly received the latest updates, and not a parallel fake history of the data. Through this method all applications can be sure that they received the true data every other application also received, without exposing unencrypted data on-chain.

By encrypting the updates to the different applications' public keys, the user remains in full control of who receives the push updates. The data itself is not exposed to the public.

8 Possibility of Public Data

Users can opt to publish specific unencrypted data formats to IPFS directly. That way anyone can receive that data, verify its integrity, and verify 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 data that is not sensitive in this fashion. 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.io protocol to IPFS is irreversibly on the internet and cannot be taken back.

9 Data Exchange Model

9.1 Chainpoint Modification

dock.io makes use of 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.io uses a modified specification of Chainpoint to not just

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

anchor updates of data formats to a single blockchain and include it in a single block, but to prove that the update was created at 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 delta) being anchored in hex string format
prevHash	The hash value of the previous hash of the blockchain transaction being referenced
currentBlock	Current, Past, or Future 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)

Chainpoint data of encrypted data being shared with specific applications is structured exactly the same. The sole difference is that the encrypted versioning chainpoint data has one more field:

non_encryptedHash	The hash value of the blockchain transaction with the targetHash of the unencrypted update.

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 it's own key, it can compare the hash value of it's decrypted update with the hash value of the unencrypted update being referenced on-chain publicly.

By including the currentBlock hash we can establish the earliest update that was generated by the user. Ethereum block hashes are unpredictable but provable, any user can know for certain 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 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.

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

9.2 Client-Side Git

All versioning of the data formats will happen client-side and off-chain. The actual versioning is not resource intensive and can be performed on any device, including a mobile device operated by the user. The first implementation will use simple GIT[4] for versioning of data formats. Later versions might include more compact versioning methods. The end-goal of the versioning functionality is to build a versioned key-value store similar to Noms[5].

Versioning, rather than publishing, of full content hashes is done to ensure correct versioning of the same data format. Any application receiving the previous version can check whether the current version truly is a continuation of the previous one. If non-delta based publishing of content hashes was used, then the applications would not be able to check for that. A whole class of applications will rely on this feature for multiple reasons.

9.3 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.io Contract State
- Vouching Signatures
- Arbitrary UTC Timestamps
- Possibly External ID Verifiers (In the case of civic, only predetermined)

No user data is stored in encrypted or unencrypted form on-chain. No application can infer the 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.

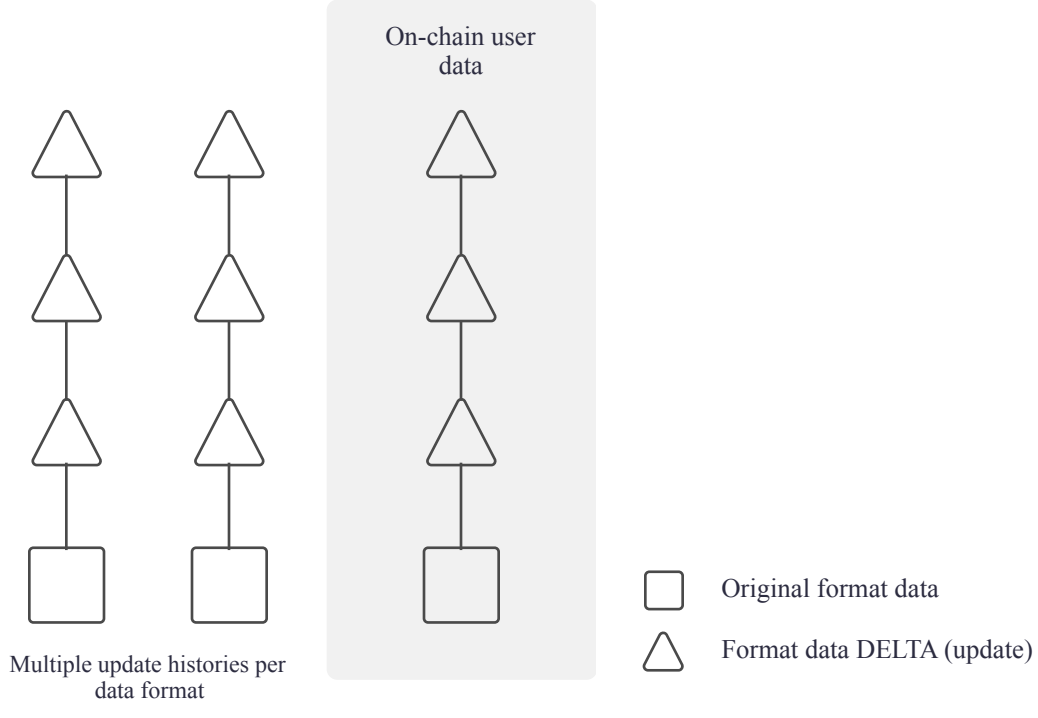
The user can chose to publish unencrypted updates and data to IPFS directly. Applications can then directly download those updates from IPFS without the prior permission from the user.

[4] https://git-scm.com/
[5] https://github.com/attic-labs/noms/blob/master/README.md

9.4 Off-Chain Data

Off-chain data includes all data format information, including prior updates and versions, 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 versioning and encryption for every application requires both additional storage as well as computational resources. 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 versioned form on-chain. Any underlying data published to the IPFS network overall in this fashion, is just as accessible as on-chain data. Users can opt to make their data IPFS addressable and versioned on-chain, but not share the underlying data on the IPFS network. This way the integrity of the data can be proven even if the data itself was never published publicly.



10 On-Chain Data Attribution Verification

Applications receiving data from users will also know if any data format was originally pushed from another application to the user's profile. Applications can both sign the pushed content itself as well as the version hash the user publishes to his/her smart contract profile. The user can manually withhold content signatures, but then the application might not sign the version hash on-chain.

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.

11 On-Chain Vouching and Signaling

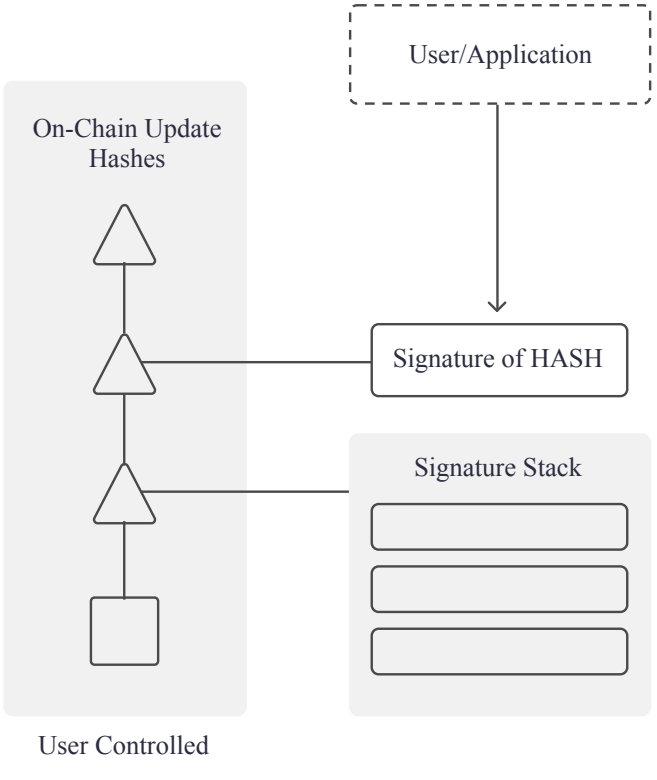
Any account can sign on-chain data format version hashes of any other account. By doing so, one signals to the rest of the network that the content of that version hash is meaningful and correct. Use cases include, but are not limited to:

1. Employers vouching for the truthfulness of their employees' work history.
2. Schools vouching for the truthfulness of their student's 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. Signing and publishing of signatures of versions/data is fully at the discretion of the signing party.

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



12 Data Types and Standard Formats

dock.io data formats are essentially the equivalent of microformats[6] for Ethereum smart contracts. The web microformats community already has a resume based microformat example that is universally machine readable[7]. The dock.io proposed microformat model is also content addressable via IPFS, versioned via GIT, and timestamped and anchored via the modified Chainpoint approach. Data formats are not prescribed in the dock.io Ethereum smart contract. Users can choose to create, version, share, and use whatever microformats the community settles on. Data formats are only useful to users as long as a large number of applications accepts them.

12.1 Initial Format Types

dock.io 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.io approach as well. Data format examples include, but are not limited to:

1. Employment history format (resume)
2. Social Graph (mutually signed contacts)
3. Licenses (relevant certifications/licenses)
4. Education (education history)
5. Application specific formats

All format data consists of “tags” and “data” fields. Tags 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 means that if an application wants to change something to 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.

We imagine that clients will do this automatically as the formats and the 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.

13 Versioning of Individual Tags Within Format Data

The smallest atomic unit within a data format that can be versioned is a “tag” and “data” pair. The smallest any single data format can be is a single “tag” and “data” pair. There is no upper

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

bound to how large a set of these pairs can be. If a large number of pairs requires updating frequently then we advise to version the individual pairs.

13.1 Rare Number of Updates

If the number of updates is rare, then we advise to only version the data formats overall and have the version 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.

14 Experience Format Example

Tag	Data
Type	Experience event
Summary	Computer scientists
Employer	Dock.io 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.

15 Token Model

Today, applications are naturally incentivized to hoard data on their users. User data is 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[8]". Linkedin has also utilized similar tactics.

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

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.

15.1 Token Utility

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 in the protocol possible. Applications are incentivized to keep exchanging data with one another long term.

15.2 FIAT-Denominated Fee Oracle

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

15.3 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.

15.4 Reasoning Behind Not Including Users in the Token Economy

If tokens were used to encourage users to share their data it could lead to a number of different issues. Users could send out faulty data in an effort to maximize their profits. Users could also practice deceitful tactics like faking accounts and playing the system as a means of 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 an everyday basis. By incentivizing applications and not users, the dock.io 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.io protocol. The first time this data has been accessed, the payment will be sent to the application that is attributed to that data. Every time afterwards that the data is accessed by other applications, the fee paid by the other applications is burned.

For example, imagine a user sends data first created by Application A to Application B. The dock.io 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.io protocol.

15.5 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 changed. The percentage of payment for associated data is changeable through an onchain 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 unusable.

15.6 Token Burn

There are a number of reasons why the burning of dock.io tokens makes sense:

1. There should be a cost for applications to acquire user data.
2. At the same time, users should not receive those tokens as incentive to avoid malicious behaviors.
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.

15.7 Game Theoretic Effects

The dock.io protocol represents a two-sided market[10]. This means that one side, in this case the users, 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. All current major data platforms do not offer complete control. Since users won't 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.

16 Possible Issues and Hurdles

16.1 On-Chain Scaling and Transaction Flooding

The dock.io 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. Ethereum scaling efforts intend to solve this problem. Regardless, any type of global state blockchain system will be prone to transaction floating issues.

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

Theoretically, the dock.io 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.

16.2 Need for ETH for Gas by Users

The dock.io 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.io tokens in order to use the dock.io protocol. Applications need both in order to send and execute transactions. Users need both so they can send version data of their latest user related data.

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

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

16.3 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 chain would be dedicated solely to the dock.io 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.

16.4 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.

16.5 Token User Experience

End-users might not want to take on the cognitive load of buying a small amount of tokens, setting up their wallet and data application, manually inputting data, setting up connections, and approving applications. This is required for the user 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.

16.6 Centralized Exchange Rate Oracle Censorship

The centralized oracle providing token-FIAT life exchange rates to the dock.io 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.io 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.

16.7 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, it is up to them what they do with it. The user can stop sharing new data with the application sharing with non-permitted applications.

This is both a positive and a negative. It can be beneficial, allowing users the option to share certain updates or basic data in a public manner. This means the version hash which they commit to the dock.io Ethereum smart contract would be committed to IPFS itself by the user in 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 the user. There are many application use cases that show this can be useful for the applications and the users.

However, this can also be a negative. 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.

17 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.io, 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 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 - Senior 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 Remote.

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.

18 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