Wolk SWARMDB:
Decentralized Database Services for Web 3

October 15, 2017

Wolk Inc.

Abstract

Ethereum SWARM is an open source project of the Ethereum Foundation being adapted and extended by Wolk to support: (1) SWARMDB, a decentralized NoSQL implementation to store and retrieve id-column-value combinations in the style of Google BigTable / Apache HBase; (2) decentralized content + data marketplaces, where providers are paid for content + data delivered through SWARM. This white paper outlines Wolk's goals on both efforts. It details how the use of Ethereum-backed ERC20 Wolk tokens (token symbol: WLK), offered in a Wolk Token Sale for contributors, will be used to develop a Wolk Ecosystem where storage and bandwidth providers earn Wolk tokens for participating in this decentralized NoSQL and content/data marketplace. This ecosystem is also expected to have wide applicability for: (a) decentralized applications, who will pay to store and retrieve record content/data and (b) content/data providers, who will earn Wolk tokens for providing highly desirable content/data stored in decentralized data exchange -- without any intermediary. We outline how a decentralized index and SWARMDB interfaces support these efforts and the development of the Wolk ecosystem.

1. Wolk Mission and Core Value Proposition

Wolk's mission is to create open data systems for the world. Wolk aims to do this by adapting Ethereum SWARM, an open source project of the Ethereum Foundation. In this white paper, we describe how Wolk is adapting and extending SWARM to support the development of:

(1) SWARMDB, a decentralized NoSQL implementation for storage and retrieval of id-column-value combinations in the style of Google BigTable / Apache HBase;

(2) decentralized paid content / data marketplaces, where content of files and data in SWARMDB tables can be obtained using Wolk Tokens.

A new ETH-backed Wolk token (WLK) is used to power SWARMDB incentives for the two, above mentioned, core use cases. This whitepaper describes parameters for a Wolk Token Generation Event. [1]

Background

The Ethereum SWARM project has a functioning implementation of a Kademlia-based peer-to-peer storage protocol backed by ERC20 token-based incentives for *bandwidth* and *storage*. Files are chunked into a Merkle tree, where the top level root hash identifies the file in a collision-resistant way. Chunks are forwarded peer-to-peer to nodes where the closest nodes store the chunks using Kademlia's well-understood XOR distance metric and support $\log_2(n)$ storage and retrieval. When a Merkle tree root hash requests a file, in parallel, the SWARM requests a set of chunks, and each chunk is retrieved from SWARM nodes forwarding on chunk requests to the closest nodes that contain the chunk, passing chunks back to the original requesting node.

SWARM DB's accounting system is done with ERC20 tokens in Wolk's (currently private) SWARM, the ERC20 token is WLK (rather than ETH, as in SWARM's current POC docs/implementation). The basics of SWARM's incentivization model are as follows:

- For Bandwidth, one SWARM node compensates one of its direct peers when its balance due to the peer exceeds a specific threshold. Once compensated, the transfer is stored on the Ethereum blockchain. Each node maintains an internal count of "Checks" that can be cashed in from each of its peers. This incentivizes delivery of highly popular chunks of data where:
    - If a content/data buyer/node retrieves a lot of data from other nodes, they pay for it using WLK
    - If a content/data supplier/node delivers a lot of data, they earn WLK
- For Storage, SWARM incentivizes *long-term* storage (specifically of rarely accessed chunks) with nodes that register with the SWEAR contract that implements a WLK-based security deposit secured on the Ethereum blockchain. Registered nodes sell receipts for chunks received, where receipts are used in the challenge-response "court" system of SWINDLE contracts. A system of guardians implement the court system, enabling data uploaders to upload and disappear and effectively check whether chunks are still available by the nodes swear. When nodes cannot provide (or can) proof of custody of chunks, the guardians/judges award the deposit to the data requester (or not).

[1] Please refer to the Wolk Token Term Sheet for information on who may participate in the Token Generation Event.

While SWARM was not conceived to support record storage or incentives for providing data+content, the mature infrastructure of chunks, manifests, and token-based incentives offers a solid base for the extension of SWARM along these lines.

Earlier versions of Wolk's data exchange focused on developing only a *partially* decentralized architecture; where Wolk was an intermediary between buyer and seller and Wolk Inc would earn up to 20% of Wolk Tokens as data buyers used APIs to access data. This was not a *fully* decentralized design, because Wolk continued to act as an intermediary. Initially, we were aiming for BigTable as a cache for SWARM data, but as our design for SWARMDB developed in Summer 2017, it became clear that SWARMDB could support a completely decentralized design, and eliminate Wolk's position as an intermediary.

2. SWARMDB

Wolk's design for SWARMDB is one where decentralized file storage is repurposed to support the organization of "schemaless tables" (called "NoSQL"), where table records are indexed solely by a single primary key. Each table record can have any number of unspecified column qualifiers and columns and values associated with those columns, obeying a hierarchy of:

```
table owner ("0x7289…")
       table ("account")
              id ("jsmith1234@gmail.com")
                     column qualifier ("profile")
                            column (profile:face)
                                   value ("\0x48\0xa7\0xff...")
```

Table owners are specified by their Ethereum address (e.g. "0x7289…") interacting with a SWARMDB interface, which must hold Wolk tokens to store and retrieve rows from a SWARMDB table. Rows in the table are indexed by id (e.g. "jsmith1234@gmail.com"). A column qualifier (e.g. "profile") allows for restricting the number of column-value combinations for efficient retrieval.

A specific *{ owner, table, id }* combination is mapped to a SWARM manifest holding the columns and values (more precisely, SWARM hashes of the values) for that id. Record-level read permissions fall into one of these 3 categories:

1. Private – the data can only be read by the owner of the data
2. Permissioned – the data can be read with permissions, using proxy re-encryption methods
3. Public – the data can be read by everyone

The full design for the permissioned case using decentralized Key Management Systems and proxy re-encryption we leave to a subsequent technical note; planned design with Wolk partner Nucypher is underway but needs coordination with SWARM's efforts to have chunk-level encryption. Concerning write permissions, the current working design is that records of the table can only be written to by the table owner. Designs to support public writes by multiple data suppliers for the same id are currently underway.

2.1 SWARMDB Interfaces

Wolk's implementation of SWARMDB exposes:

1. a command line interface
2. HTTP API
3. Go interface modelled after Google BigTable's interface.

These interfaces enabled content/data requesters and suppliers to read and write records into SWARM. The SWARMDB client connects to an authenticated Ethereum account that has WLK credited or debited to it for all gateway operations.

In these interfaces, each time content/data suppliers upload content/data onto SWARM, they may specify their bounty amount, in WLK. This bounty is the amount a requester must "pay" to access the file or record. For now, we assume optimistic retrieval of the entirety of content/data (e.g. access to less than 100% of the chunks of a file / record results in 100% of the bounty), and that failure to retrieve data are prosecuted by SWARM's court system.

When a node requester wishes to pay a bounty, the maximum WLK bounty the node requester is willing to pay is sent in the form of a "transaction receipt" representing a promise to pay up to that amount of WLK if the top level data is returned. This transaction receipt is passed peer-to-peer until the data chunk is found at some node leaf. If this node leaf has a WLK bounty attached to it specified in an earlier write operation (either for a specific file or a record's id-column) then that amount may be collected by the end node from the buyer based on the transaction receipt. This bounty for delivery is on top of the storage and bandwidth incentives on SWARM. We expect to evolve this paid content / data delivery mechanism significantly in the near future.

2.1.1 Command Line Interface

A command line interface "swarmdb" enables writing table/column/value combinations for:

- Writing Public records with column + value specified by local file, where the presence of a bounty makes the record public:

```
 swarmdb up --table account --id jsmith1234@gmail.com -column profile:photo
-file john-selfie.jpg -bounty 2.0
[ { "column": "profile:photo", "tx":
"0x0bbc266eae70eaf962eab2ddb3be48a0975771b44a1d4f7cf2e227fbf76916b4" } ]
```

- Writing Private records with column + value specified on commmand line where the lack of a bounty makes the record private.

```
Example: swarmdb up --table account --id jsmith1234@gmail.com -column
stats:uploadCount -value 723
[ { "column": "stats:uploadCount", "tx":
"0x3be48a00bbc266eae7076916b4e975371baf962eab2ddb44a1d4f7cf2e227fbf" } ]
```

- Writing Public content specified by local file, where the bounty makes the data public and the lack a table specification adds to a generic "content" table shared by all

```
Example: swarmdb up --id "Beethoven-Ode to Joy" --file ode-to-joy.mp4 -bounty
0.001
```

- Reading content of table keyed by id:

```
Example: swarmdb --table account --id jsmith1234@gmail.com
[{"column":"profile:name", "value": "john smith"},
```

```
{"column":"stats:lastUpdatedTS", "value": "1424712341"},
{"column":"stats:uploadCount", "value": "723"},
{"column":"profile:photo", "value": "0x1423..."} ]
```

2.1.2 HTTP API

The previous command line interface maps into the HTTP API calls below:

GET http://localhost:8500/swarmdb:/tbl/id/column
Retrieve from the table `tbl` a specific column `column` value for the id `id`, retrieved in JSON form

Request:
http://localhost:8500/swarmdb:/account/jsmith1234@gmail.com/profile:photo

Response:
```
[{"column":"profile:photo", "value": "0x1423"}]
```

GET http://localhost:8500/swarmdb:/tbl/id
Retrieve from the table `tbl` *all* the columns values for the id `id`, retrieved in JSON form

Request:
http://localhost:8500/swarmdb:/account/jsmith1234@gmail.com

Response:
```
[{"column":"profile:name", "value": "john smith"},
{"column":"stats:lastUpdatedTS", "value": "1424712341"},
{"column":"stats:uploadCount", "value": "723"},
{"column":"profile:photo", "value": "0x1423..."} ]
```

GET http://localhost:8500/swarmdb:/tbl/id?cq=cqfilter
Retrieve all the columns values for the id where each column is part of the column qualifier family in "cq", retrieved in JSON form.

Request:
http://localhost:8500/swarmdb:/account/jsmith1234@gmail.com?cq=stats

Response:
```
[ {"column":"stats:lastUpdatedTS", "value": "1424712341"},
 {"column":"stats:uploadCount", "value": "723"} ]
```

POST http://localhost:8500/swarmdb:/tbl/id/column
The POST request writes a (table, id, column, value). The value for the column is specified in the body of the POST. SWARMDB returns the hash of the record along with the transaction receipts

Request:
http://localhost:8500/swarmdb:/account/jsmith1234@gmail.com
```
[{"column":"profile:name", "value": "john smith", "bounty": .001},
{"column":"profile:photo", "value": "0x1423...", "bounty": .01} ]
```

Response:
```
[ {"column": "profile:name", "tx":
"0x6492d8487c03c2f33a8dc83c2980e818485c3033486920e26af5554e909bc459" },
```

```
    { “column”: “profile:photo”, “tx”:
“0x0bbc266eae70eaf962eab2ddb3be48a0975771b44a1d4f7cf2e227fbf76916b4” } ]
```

POST http://localhost:8500/swarmdb:/tbl/id
The POST request expects a full JSON structure of all column and values.

2.1.3 Go Interface

The SWARMDB Go interface is modelled on Google BigTable’s interface.

1. To read a *single* row from SWARMDB, use the ReadRow method of SwarmDB.Table:

```
swarm_table := swarm_client.Open("account")
// "jsmith1234@gmail.com" is the entire row key
row, err := swarm_table.ReadRow(ctx, "jsmith1234@gmail.com")
...
```

2. To read *multiple* rows from SWARMDB, use the ReadRows method of SwarmDB.Table to get multiple rows. A RowRange specifies a virtual portion of a table. A RowFilter such as FamilyFilter limits the data that is returned to specific column families.

```
swarm_table := swarm_client.Open("account")
// Read rows starting with "jsm", but only columns with the "profile"
column qualifier
prefix_range := swarmdb.PrefixRange("jsm")
err := swarm_table.ReadRows(swarmdb_context, prefix_range,
func(swarm_row Row) bool {
   // do something with swarm_row
   return true // keep going
}, swarmdb.RowFilter(swarmdb.FamilyFilter("profile")))
...
```

3. To write a row to SWARMDB, use Mutation to build up one or more column-value operations on a table, and then use the Apply methods on the Table to execute the operations. For instance, to set a couple of cells in a table,

```
swarm_table := client.Open("account")
mut := swarm_table.NewMutation()
mut.Set("profile", "name", bigtable.Now(), []byte("John Smith"),
.001)
mut.Set("stats", "lastUpdatedTS", bigtable.Now(),
[]byte("1424712341"), 100)
err := tbl.Apply(ctx, "jsmith1234@gmail.com", mut)
…
```

The WLK Bounty is set in the Mutation record.

2.2 SWARMDB's Index

One central task of a database is for fast record lookup by a table's primary key. In SWARMDB, the challenge is to be able to quickly look up any id's data. For our POC, every value ({ owner, table, id }) is hashed using a simple ":" separator using the 256-bit SHA3 hash of "owner:table:id" e.g.
`0x728781E75735dc0962Df3a51d7Ef47E798A7107E:account:jsmith1234@gmail.com` => `2477cc8584cc61091b5cc084cdcdb45bf3c6210c263b0143f030cf7d750e894d`
...
Currently, SWARM's POC stores the mapping between ids using an Ethereum Patricia Tree where the mapping between the hashed ids
`Record 1: 2477cc8584cc61091b5cc084cdcdb45bf3c6210c263b0143f030cf7d750e894d`
=>
`SWARM HASH:`
`b5cc084cdcdb45bf3c622477cc8584cc6109110c263b0143f030cf7d750e894d`
...

To store a record (owner, table, id) in SWARM with data D:

1. Content Storage: Build Merkle tree for data D, store in SWARM, getting back swarm hash `H(D)` (based on the content D) and transaction receipt `TXR(D)`.
2. Index Storage: Compute hash `HID(owner, table, id)`, and update Patricia Tree using Kademlia routing.

To retrieve a record (owner, table, id):

1. Index Retrieval: Compute hash `HID(owner, table, id)`, and obtain SWARM hash of record from Patricia Tree via Kademlia routing
2. Content Retrieval: Build Merkle tree from SWARM hash, reconstructing missing data

When a record is updated (any column, any value for any column), the SWARM manifest changes. If a large number of updates are committed to the same record, at the gateway it is a simple optimization to batch process and only update the (owner, table, id). Another potential optimization is to keep "small" content values (much less than a chunk, ie less than 32 bytes) directly in the index in place of the SWARM hash; this can be done with the Ethereum RLP scheme.

Clearly, SWARMDB makes no attempt for consistency and so decentralized applications must only use SWARMDB for records that do not change much more often than every few seconds.

A key new concept (developed by Viktor Tron [SWARM project lead]) that must be implemented in the SWARMDB roadmap is the representation of a *provable object traversal*. To incentivize long-term storage of manifest encoding embedded in the decentralized index, we require a guarantee of storage of manifest encoding analogous to the SWEAR (guarantee of storage of a chunk) that is also paired with the SWINDLE. When an index mapping is stored in the SWARM, a transaction receipt is stored just as with SWEAR. The transaction receipt can be mapped onto a "provable object traversal", a sequence of manifest encodings. The network protocol for delegated traversal and POT proof response is being developed in further detail.

Data on-boarders may specify a WLK "bounty" for each record. This bounty is kept in the manifest in our current POC.

3. Wolk Ecosystem

To support open data and content sharing Wolk will develop the Wolk Ecosystem:

- Wolk Inc., a for-profit Delaware corporation, promoting the development of a decentralized database service
- Content/Data Suppliers, who onboard content + data and earn WLK
- Content/Data Buyers, who pay WLK to access content + data using the SWARMDB interfaces
- Storage and Bandwidth Providers, who run SWARMDB nodes and earn WLK
- Decentralized Apps, who pay WLK in exchange for decentralized NoSQL storage

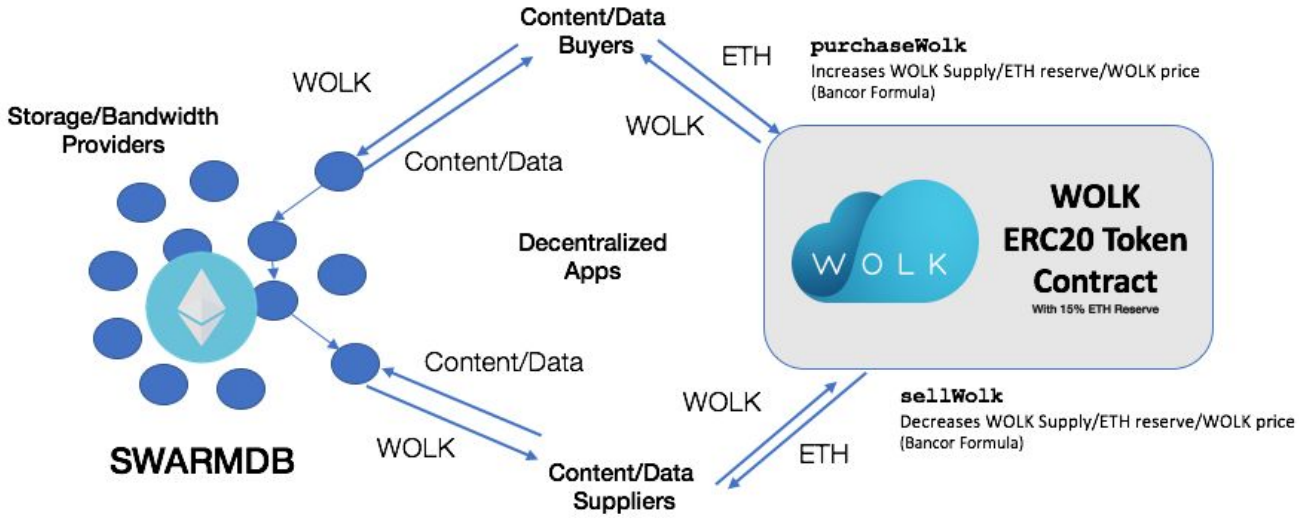
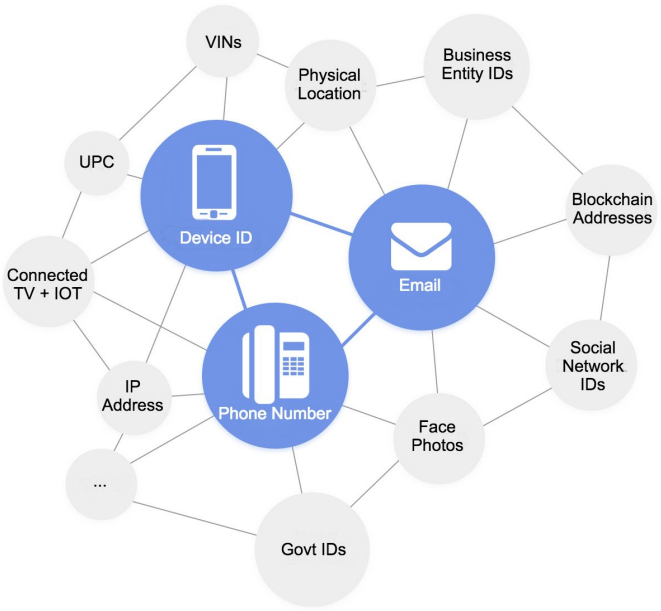


Figure 1. Data + Monetary flow in Wolk Ecosystem.

The monetary flow is that "real money (ETH)" from decentralized applications and content/data buyers will flow into the network of storage + bandwidth providers and the content/data suppliers.

3.1 Data Suppliers and Buyers

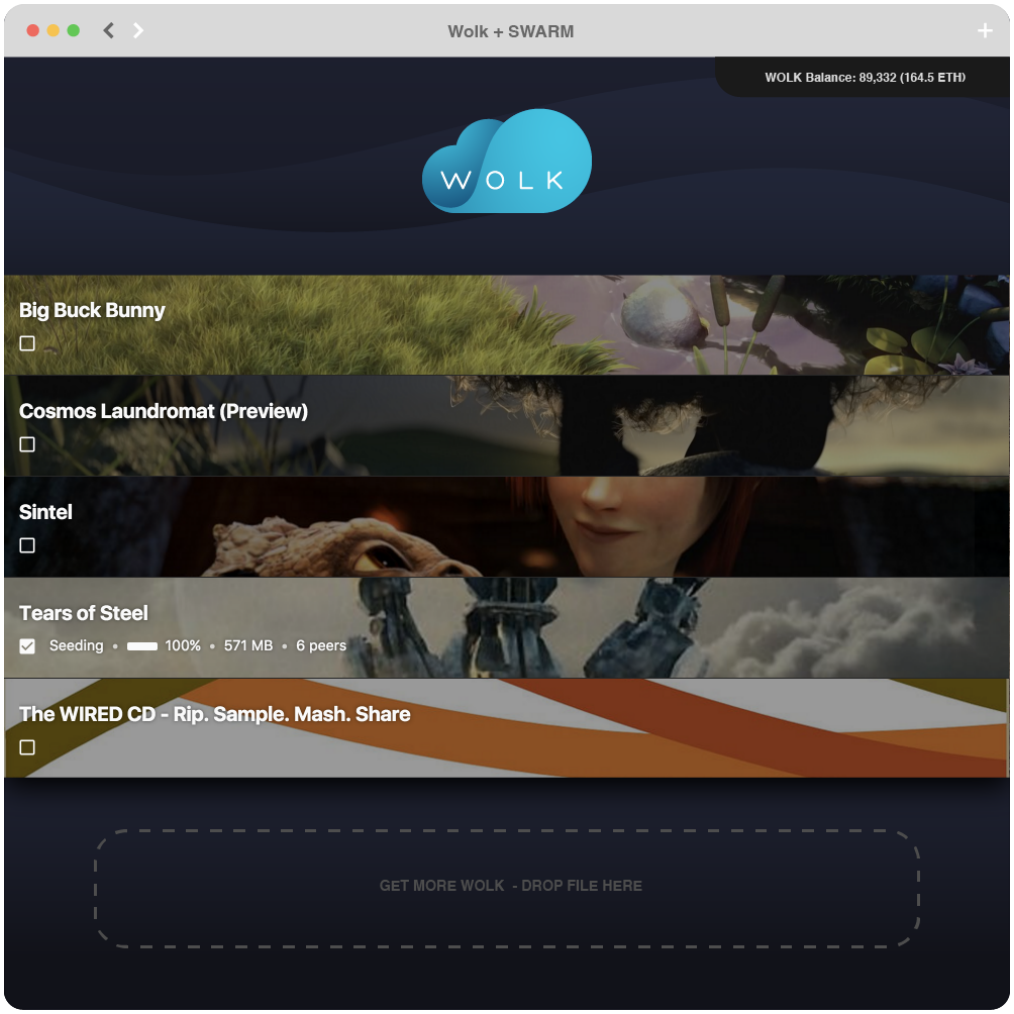
Wolk Inc. supports the onboarding of many ID spaces in SWARMDB about IDs in a wide variety of ID spaces, starting with emails, phone numbers and mobile device IDs:



For each ID space, standardized attributes will have own columns in public SWARMDB tables. Data suppliers will use one of SWARMDB interfaces to onboard their data. Content suppliers may do the same. As buyers access the data, sellers accumulate WLK tokens. By allowing WLK to be exchanged for ETH using a 5% ETH-reserve, Wolk provides data sellers a potentially liquid market for their data, so long as ETH itself is liquid. In addition, the WLK token is an ERC20 token, so with sufficient transaction volume and subject to the transfer restrictions described below, it will potentially be tradeable on a number of exchanges and supported by multiple cryptocurrency wallets, allowing data suppliers to monetize their WLK token, although Wolk is not involved in these exchanges and cannot guarantee they will allow transfers of WLK tokens.

3.2 Storage/Bandwidth Providers, Content Sharing

To stimulate and support widespread adoption of a large SWARM, Wolk will release a consumer-friendly, Electron-based implementation of a SWARM client based off an open source BitTorrent client (c.f. WebTorrent) that enable consumers to earn WLK for content sharing and providing storage.



Preliminary SWARMDB client

Unlike BitTorrent clients, SWARM clients have the advantages of being highly censorship-resistant, monetization-oriented (because WLK can earn storage providers ETH), and privacy conscious (because data of a file / record is not stored at any one specific node).

3.3 Decentralized Application Publishers

Decentralized application users or developers need a large scale SWARM network. Wolk is conversing with early beta candidates who need to fully decentralize their applications with SWARMDB.

3.4 Wolk Team

Wolk Inc core priorities in the first 18-24 months are to:

- develop commercially viable SWARM database services for decentralized applications, contribute to the Ethereum SWARM open source project
- support a community of SWARM storage providers, content and data buyers, and content and data suppliers

- host SWARM nodes alongside and support guardianship and ensure storage goals are being met. Successful large scale deployment will consist of:
    - Less than 10^{-6} loss for low scale chunk data storage
    - Demonstration of SWEAR incentive structure working with court system in practice
    - Resistance to data loss

The Wolk Core Team consists of:

- Sourabh Niyogi, 45, Wolk CEO
- Sonia Gonzalez, 38, Wolk Inc GM
- Rodney Witcher, 37, Wolk VP Business Development
- Michael Chung, Wolk Product Manager and Protocol Developer
- Mayumi Matsumoto, Wolk Data Scientist and Protocol Designer
- Alina Chu, Wolk Data Scientist & Quantitative Analyst
- Bruce Han, Wolk Software Engineer
- David Gentzel, Wolk Protocol Developer

Please visit our site team page here: https://www.wolk.com/#team

Wolk's Roadmap anticipates milestones in multiple related threads:

1. SWARM database services development: permissioning development (proxy re-encryption, decentralized key management services), node incentive research, blockchain connectivity with very high throughput / low latency mechanism (c.f. Plasma).
2. BETA Decentralized Apps utilizing SWARM DB in specific use cases and providing initial feedback
3. commercial storage

4. Wolk Token Economics

4.1 Wolk Token Sale

Wolk will distribute up to 17,500,000 Wolk tokens (symbol "WLK") in a Wolk Token Sale from October 15, 2017 (block # 4370000) until October 31, 2017 @ 11:59pm PST. U.S. purchasers and non-U.S. purchasers must register to buy WLK through the Wolk website and once approved may send Ethereum to the Wolk Token Contract Address listed below.

The price of WLK Price in ETH:

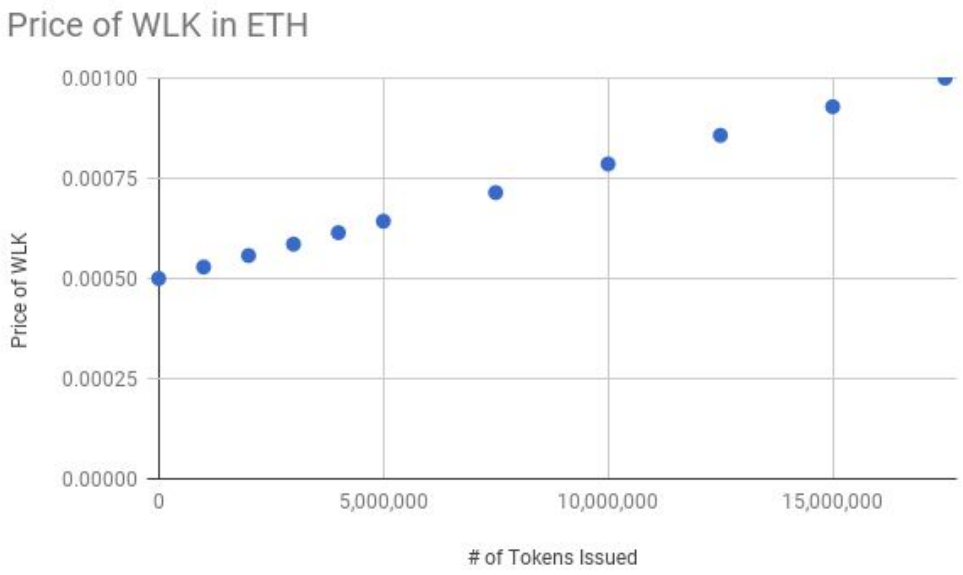
$$.0005 * (1 + \frac{T}{17{,}500{,}000})$$

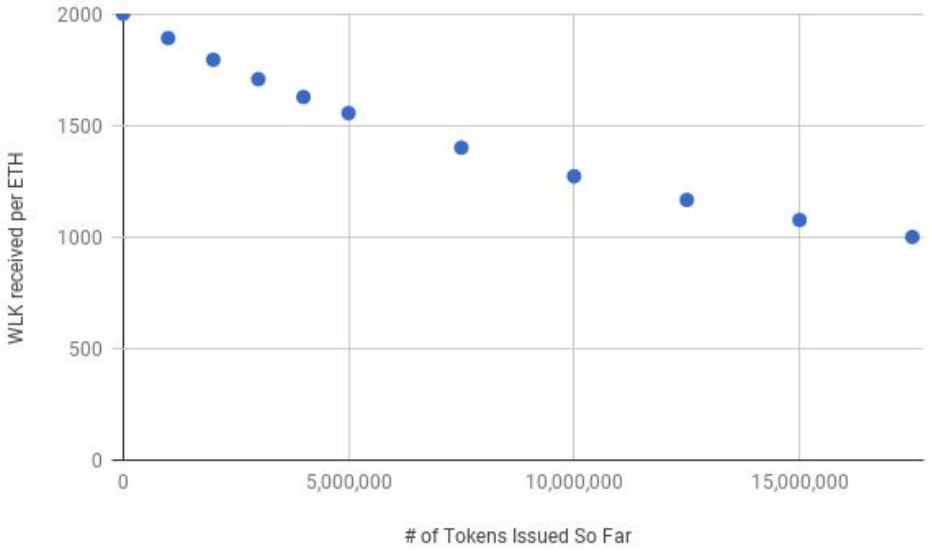
where T is the # of Tokens issued at the time ETH is sent into the contract.

As the token sale progresses, the price of each WLK token increases from

Initial:	.0005 ETH	when T = 0	(ie 2000 WLK per ETH)
Maximum:	.0001 ETH	when T = 17,500,000	(ie 1000 WLK per ETH)

This is shown in the following 2 graphs:





The Wolk Token sale will end when either the maximum number of WLK is sold (17,500,000 WLK) or the Token Sale End Date/Time has been reached (October 31, 2017 11:59pm PST).

An additional 50,000,000 WLK is created and held by Wolk Inc. when the token sale is finalized, so immediately after the Token Sale there may be up to 67,500,000 WLK in circulation.

Key parameters:

- Token Sale Minimum: 1 WLK
- Token Sale Maximum: 17.5MM WLK
- Token Sale Start Date/Time: Block # 4370000 [October 15, 2017 10:00pm PST]
- Token Sale End Date/Time: October 31, 2017 11:59pm PST
- Wolk Token Contract Address:*
  0xf6b55acbbc49f4524aa48d19281a9a77c54de10f [ENS: wolkinc.eth]

***To be safe, always double check the Wolk Token contract address displayed on Wolk.com.**

4.2 Post-sale Economics

After the Token Sale, data buyers and suppliers may trade tokens freely on exchanges that allow such trades, subject to compliance with the restrictions on transfers described in Section 4.3.

Wolk has partnered with Bancor to offer a Token Changer. Wolk offers 2 smart contract functions - "purchaseWolk" and "sellWolk" - to exchange ETH for WLK and WLK for ETH freely, compensated from a 5% ETH reserve in the Wolk Token Contract. The implementation of purchaseWolk and sellWolk utilizes the “Bancor” formulas of calculatePurchaseReturn and calculateSaleReturn (see http://bancor.network white paper) where 5% of ETH will be kept in reserve to support the liquidity of WOLK directly:

- data buyers will be able to buy WLK by sending ETH to the "purchaseWolk" function of the WLK Token Contract, which will use the ETH:WLK exchange rate internally to the contract along with the calculatePurchaseReturn formula to determine the amount of WLK to transfer to the buyer.

The ETH contributed goes into a "ETH reserve" and the prevailing ETH:WLK exchange rate is automatically updated. Buyers who obtain WLK then use SWARMDB interfaces to obtain data from SWARM;

- data sellers will be able to sell WLK by interacting with the "sellWolk" function of the WLK Token Contract, specifying the amount of WLK they wish to sell in exchange for ETH. Using the ETH:WLK exchange rate internal to the contract as well as the calculateSaleReturn formula, ETH is reduced in the reserve and the ETH:WLK exchange rate is automatically updated.

The total number of tokens therefore can increase from data buyers calling purchaseWolk and can decrease from data sellers calling sellWolk, driving WLK's value up (via purchaseWolk) and down (via sellWolk) programmatically. Example 1 shows the token economics of how contributors can turn ETH to WLK via purchaseWolk and convert WLK back to ETH via sellWolk after the Token Generation Event is completed.

The actual price of WLK is calculated as follows:

- R - Reserved ETH Balance in WLK Token Contract
- S – Current Total Supply of WLK Tokens
- F - Reserve Ratio, which is fixed at 5%

T = WLK received in exchange for ETH E sent to purchaseWolk, given R, S and F

$$T = S\left((1 + \frac{E}{R})^{F} - 1\right)$$

E = ETH received in exchange for T WLK in sellWolk, given R, S and F

$$E = R\left(1 - \sqrt[F]{1 - \frac{T}{S}}\right)$$

The derivations of the Bancor formulas are found here.

4.3 Restrictions on Transfers

The Wolk Tokens are being offered in reliance upon exemptions from registration under the Securities Act of 1933 ("Securities Act"). Therefore, unless the tokens are used in commercial transactions using the Wolk services, Wolk tokens may not be transferred within the United States or to a "U.S. person" unless such transfer is made to an "accredited investor," in compliance with applicable securities laws, and may only be transferred in a transaction outside the United States to non-U.S. persons, unless and until Wolk reasonably determines and notifies holders that the tokens are not securities and freely tradeable. Any transfer made in violation of these provisions will be void.

Based on anticipated use of the Wolk tokens, Wolk believes that over time, the Tokens will reasonably be treated as non-securities for purposes of U.S. law, at which point Wolk will notify holders of the finding and that the Wolk Tokens are freely tradeable. There is no guarantee that this will occur.

4.4 Use of Proceeds

ETH raised in the Token Sale will be distributed as follows:

- 5% - supports liquidity of Wolk tokens directly via purchaseWolk, sellWolk functions directly in the token contract, which increase this reserve programmatically

- 75% - supports Wolk Inc in developing SWARMDB and the Wolk Ecosystem and costs around marketing, business development, operations, etc.
- 20% - will support a “Wolk Ecosystem Development Fund,” which will support content + data suppliers onboarding by Wolk Inc.

4.5 Wolk Inc Sustainability / Follow on Offerings

Wolk Inc developing the Wolk content + data ecosystem and the SWARM database services. Wolk Inc sustainability is derived from:

- the Wolk tokens earned from hosting SWARM nodes.
- the 75% of ETH from the Token Generation Event (see 4.4 above)
- Professional services provided to publishers for the onboarding and support of their data

4.6 Wolk Token Sale FAQs

What do Wolk Tokens represent?

Wolk Tokens represents a decentralized virtual currency represented in specialized Ethereum smart contracts following an "ERC20" standard. Wolk Tokens enable participants to interact with Wolk’s SWARM database services, as storage, content and data is obtained with Wolk Tokens as a payment mechanism. Content + data suppliers and storage + bandwidth providers are compensated in Wolk Tokens for their highly valuable content, data, storage and bandwidth.

Are you distributing Wolk Tokens in the Token Sale?

Yes. Wolk Inc. is offering to distribute up to 17,500,000 Wolk Tokens in aggregate. See section 4.1-4.5 for additional information on token economics.

How do I obtain Wolk Tokens in the Token Generation Event?

To obtain Wolk Tokens, you must e-sign a Purchase Agreement and send ETH to the Wolk Token Contract from your Ethereum Wallet. Other cryptocurrencies must be exchanged for ETH to obtain Wolk tokens.

What is the value of Wolk Token? Is Wolk transferable?

During the Token Sale, the value of a Wolk Token is determined by the formula described in section 4.1. Afterwards, buyers and sellers of Wolk Tokens can potentially exchange tokens freely on cryptocurrency exchanges, subject to the restrictions described above. In addition, the 5% ETH reserve held in the Wolk Token Smart Contract enables systematic exchange of Wolk Tokens for ETH and vice versa with publicly visible exchange rates.

Wolk Tokens are transferable, subject to the restrictions described above; they can be sold to buyers who require Wolk to procure storage, data and content. All transactions conducted are verifiable and secured on the public Ethereum blockchain.

Do Wolk Tokens represent ownership of Wolk Inc?

No. WLK does not represent any ownership rights in Wolk Inc and does not represent participation in Wolk Inc.

When will the Wolk token sale happen?

The Wolk Token Sale will be conducted between October 15, 2017 and October 31, 2017.

How will Wolk store ETH?

Wolk will use the standard Ethereum multisig wallet to store ETH.

For additional information, see our Term Sheet on Wolk Tokens and our Token Generation Event.

4.7 Communications

The primary communications vehicle for Wolk is at https://wolk.com -- no other communications vehicle is considered official at this time.

Wolk maintains a Twitter account at https://twitter.com/WolkInc, encourages Twitter users to follow Wolk **@WolkInc**.

4.8 Open Source

The Wolk Token Contract is available on https://github.com/wolktoken/token and has been posted at wolkinc.eth. All code for SWARMDB will be made open source.

5. Risks and Risk Mitigation

Wolk advises on the following risks:

I. Insufficient Demand for SWARMDB Services. While Wolk is optimistic that decentralized apps will need database services, usage of decentralized applications may be slower to develop than anticipated.

II. Insufficient liquidity for Wolk Tokens. WLK is a token for a storage, content, and data for a small community. A smaller number of participants implies lower volumes of WLK transactions, which may result in higher volatility of WLK.

III. Privacy and local regulations may pose unforeseen limits on Wolk. While Wolk will abide by industry standard privacy principles and control measures, many local markets operate with different principles and the restrictions may pose limits on what can be done in decentralized storage or on the blockchain structure.

IV. Data copying and fraud protection measures are being actively developed. Data buyers may copy data, reducing the ability for data suppliers to fully monetize their data because of the actions of fraudulent actors. Wolk has developed basic countermeasures for bad actors but there can be no assurances that the measures will be fully complete.

V. Transaction throughput may be insufficient to support extraordinarily high SWARMDB demand. However, it is possible that higher throughputs will be necessary and significant investments will need to be made to improve Ethereum SWARM technology.

VI. Currently, WLK is an Ethereum-backed token. Should further developments in the cryptocurrency ecosystem enable the handling of higher throughput, lower latency software architecture, Wolk wishes to remain nimble enough to transition over to better blockchain technology.

Wolk will use all available resources to reduce risks but outside of data buyers and suppliers participating, only accredited investors and investors comfortable with the above openly stated risks should hold Wolk tokens. For additional information on risks associated with purchasing Wolk Tokens, see our Term Sheet on Wolk Tokens and our Token Generation Event and our Wolk Token Risk Disclosures.