Dispatch Protocol: Technical White Paper

Zane Witherspoon

17th September, 2018

Abstract

The Dispatch Protocol provides the benefits of trustlessness and decentralization to application-level data creation, user-level data ownership, and business-level data analytics. The system as a whole is architected as two distributed networks that operate in unison, and individual Dispatch nodes are made up of three main components: the *Dispatch Ledger*, the *Dispatch Artifact Network (DAN)*, and the *Dispatch Virtual Machine (DVM)*.

To keep all nodes in the network in agreement on what information is stored in the Dispatch Ledger, the Dispatch Protocol implements the new Delegated Asynchronous Proof-of-Stake (DAPoS) consensus algorithm. DAPoS is a Byzantine fault tolerant delegated transaction gossip protocol that scales to the capacity of the hardware validating the transactions. *Rate-limiting* is implemented as an alternative to gas to create a system with zero transaction fees for *Stakeholders*.

While all nodes maintain a full copy of the Dispatch Ledger, every node can keep a different subset of the Dispatch Artifact Network (DAN). The DAN is comprised of a distributed network of *Farmers*, storing *Artifacts*, or Merkelized units of data. Using homomorphic encryption, data owners can opt-in to *Dispatch Guru* which enables data researchers to trustlessly query fully encrypted, distributed data for valuable insights without requiring the data owner forsake their data sovereignty.

Finally, we introduce the Dispatch Virtual Machine (DVM). To support the existing ecosystem of distributed applications (Dapps), the DVM supports nearly every opcode in the EVM (Ethereum Virtual Machine) and therefore most *Solidity* smart contracts. The DVM extends the functionality of distributed applications with the ability to create, update, distribute, and analyze Artifacts in the DAN.

Contents

1 Introduction

The Dispatch protocol is a scalable platform for Decentralized Applications (Dapps). It is built upon three core principles:

- Support the existing Dapp development eco-system,
- Scale to the transaction throughput needs of supported Dapps, and
- Enable Dapps with distributed storage, distribution, and analysis of their data.

1. Since the 2015 launch of the Ethereum network [1], over 1,818 Dapps have been developed and released on the platform [2]. We believe this interest and education in Dapp development is a good thing, and we want to enable developers who have already invested their time learning these skills to be able to create even more powerful applications. The Dispatch Virtual Machine (DVM) is designed to be backwards compatible with the Ethereum Virtual Machine (EVM) and Ethereum bytecode [3], so Ethereum and Solidity smart contracts will be able to run on Dispatch, although not all Dispatch smart contracts will be able to run on the Ethereum platform.

2. Scalability has been a prominent issue in existing distributed ledger technologies. Bitcoin validates and adds transactions to its ledger at a rate of about 7 transactions per second (tx/s)[4] , and Ethereum handles an average of about (15 tx/s)[5]. This paper introduces the novel Delegated Asynchronous Proof-of-Stake (DAPoS) algorithm, which has reached a peak throughput of around 100,000+ tx/s in optimal conditions. Of course, there is substantial overhead in incorporating the consensus into a functioning protocol, but even a 90% reduction in throughput still makes Dispatch a more scalable Dapp platform than Ethereum by several orders of magnitude.

3. We believe the future of decentralized applications will emulate the world of centralized applications we have today. We intend to learn from the way these centralized applications scale to plan for our decentralized future. Since most centralized applications at scale use far more data than could reasonably be stored in a shared ledger, Dispatch is built to support applications that utilize any quantity of data at any scale. We introduce the idea of off-chain *Artifacts* that live in the *Dispatch Artifact Network* (*DAN*), which runs parallel to the shared ledger. Dapp developers can incorporate bulky data like media content, "internet of things" data, or AI/machine learning data, into the logic of their smart contracts without sacrificing security or scalability. Finally, and arguably most important, we introduce the *Dispatch Guru* which enables the trustless analytical querying of data distributed across the DAN. Data researchers can pay data owners to query their data for analytics, without requiring the data owners to reveal their data.

2 The Ledger

2.1 World State

Dispatch protocol is a transaction-based generic state machine. Every smart contracØt or Dapp built on Dispatch has its own state, and the state of the Dispatch network is composed of all the states of all the Dapps built upon it. Together, we refer to the entirety of the Dispatch ledger as the world state; formally noted as σ. Outside of application-specific states, the Dispatch world state includes two specific state subsets: (1) Account States ($\sigma[a]$) and (2) Election States ($\sigma[e]$)

2.1.1 Account State

The fundamental native functionality of the Dispatch ledger is to maintain a state of account balances. Balances are fundamental in the maintenance of the Dispatch network and underlie core functionality like *Delegate Elections* and *Rate-Limiting*. Accounts can exist either under the control of an external actor or as an automated smart contract which executes code upon the receipt of a Transaction. The Account State is comprised of the following six fields:

Address: The 160-bit address of the account holder generated from the `keccak-256` hash [6] of an external actors private key, or in the case of a smart-contract account, the address is the Transaction hash of a contract creation transaction; formally $\sigma[a]_a$

Balance: A 128-bit scalar integer value equal to the number of Divvitos owned by this address; formally $\sigma[a]_b$.

Bookkeeper: The boolean identifier of an accounts intent to contribute the network by executing accepted Transactions. Accounts with a Bookkeeper value of 1 are eligible to be elected as Delegates. Conversely, a Delegate can remove themselves from candidacy by setting their Bookkeeper value to 0; formally $\sigma[a]_d$.

CodeHash: The hash of the DVM code of a smart-contract account, or Ø for an account held by an external actor. This is the code that gets executed should a smart-contract address receive an interact Transaction. CodeHash is immutable and thus, unlike all other fields, cannot be changed after construction; formally denoted $\sigma[a]_c$. The code associated with the account may be denoted as $\mathbf{b}$, so that $\texttt{KEC}(\mathbf{b}) = \sigma[a]_c$.

Root: A 256-bit hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account (a mapping between 256-bit integer values), encoded into the tree as a mapping from the `Keccak 256`-bit hash of the 256-bit integer keys to the RLP-encoded[3] 256-bit integer values; formally $\sigma[a]_r$.

Name: An ascii-encoded unlimited-size byte array that operates as a human-readable username to be associated with accounts controlled by external actors or, for a smart-contract account, Ø; formally $\sigma[a]_n$.

2.1.2 Election State

The supplementary state maintained natively by the Dispatch protocol is the *Election State*; formally $\sigma[e]$. The Election State is used by Dispatch's consensus algorithm to elect the quorum of validators and determine their salary compensation. The Election State has three properties, each of which can be voted on by the Stakeholders:

Count: A scalar value representing how many of the top voted Delegates are considered officially elected; formally $\sigma[e]_n$.

Salary: A scalar value representing how many Divvitos each Delegate's account is credited for their work at the end of the next election cycle; formally $\sigma[e]_s$.

Delegates: An $\sigma[e]_n$-size array of 160-bit addresses representing the ordered list of the highest-voted Delegates this election cycle. These addresses are considered the network's official quorum for the remainder of the election cycle; formally $\sigma[e]_d$ where $\sigma[e]_{d[0]}$ is the highest voted Delegate and $\sigma[e]_{d[n]}$ is the $(n+1)^{\text{th}}$ highest voted Delegate.

2.2 Transactions

A Transaction (formally, T) is a cryptographically unmodifiable instruction to change the world state (σ), constructed by an external actor. A Transaction could instruct validators to move balance from the sender's account to a recipient's account, update the state of a smart contract, write changes to the DAN, or more. In the Dispatch network, all valid transactions are accepted into the ledger regardless of the success of the execution of that Transaction. A Transaction is considered

Value	Type Name	Description
0	TokenTransfer	Moves the Dispatch Divvy across accounts
1	Deploy	Instantiate a new smart contract on the ledger
2	Execute	Call a method of a deployed smart contract
3	Election	Update the Account's Election State votes

Table 1: Transaction Types (T_t)

valid if it is formatted correctly, and a Transaction is considered successful if its execution resulted in an update to the world state (σ).

Pulling apart the acceptance and execution of Transactions has two main benefits: (1) increased throughput for transactions that alter the asymmetric information stored in the DAN (Section 3) thanks to the distribution of work amongst Farmers, and (2) Transactions that are accepted but not successful can still be used to manage the bandwidth consumed by the sending account. Transactions have varying properties based on their type:

Type: A single byte that determines the function of the transaction. Required for all transactions; formally T_t.

From: The 160-bit address of the transaction's sender. Cannot be the same as the *to* Address. Required in all transaction types; formally T_f

To : The 160-bit address of the transaction's recipient. Required for TokenTransfer and Execute transaction types; formally T_{to}.

Value: A scalar value equal to the total amount of Divvitos to be transferred from the balance of the *from* address to the *to* address. Required for TokenTransfer, Deploy, and Execute transaction types; formally T_v.

Time: A `RFC3339` standard time signature [7] for the transaction. The transaction's time determines its order in the ledger. Required for all transaction types; formally T_{ts}

Code: The DVM bytecode that is executed each time a subsequent Execute transaction is called with a *To* (T_{to}) value that equals the *Hash* (T_h) of this Transaction. Required only for Deploy transactions; formally T_c

Abi: The Application Binary Interface for the smart contract being deployed. When applied to the DVM bytecode, the ABI defines the methods available to interact with the smart contract and their required parameters. Required only for Deploy transactions; formally T_a.

Method: An unlimited size byte array that maps to a method defined in the ABI (T_a). Required for Execution and Election transactions; formally T_m

Params: An unlimited size byte array that maps to the parameters of *Method* (T_m) as defined in the *ABI* (T_a). Required for Execution and Election transactions; formally T_p.

Hash: The 256-bit `keccak`-256 hash of the concatenation of the *Type*, *From*, *To*, *Value*, *Code*, *Abi*, *Method*, *Params*, and *Time* fields in that order. The hash is used as the unique identifier for all transactions. In Deploy transaction types, the hash is also used as the address of the deployed smart contract. Required for all transactions; formally T_h where:

$$\texttt{KEC}(T_t, T_f, T_{to}, T_v, T_c, T_a, T_m, T_p) = T_h \tag{1}$$

Signature: The signature is a 65 byte array in [R ‖ S ‖ V] format of the transaction hash (T_h) where V is 0 or 1. R, S, and V are the values used in the `secp256k1` ECDSA signature

algorithm used to recover the *from* address and prove that the transaction was created by the owner of the associated private key. Required for all transaction types; formally T_s.

2.3 DAPoS: Delegated Asynchronous Proof-of-Stake

One of the most exciting developments out of Dispatch is the novel Delegated Asynchronous Proof-of-Stake (DAPoS) consensus algorithm [8]. DAPoS is used to ensure that all network participants maintain an identical world state. DAPoS uses elected Delegates like Dan Larimers DPoS consensus [9], but operates on the gossiping of individual transactions rather than relying on the sequential distribution of blocks. DAPoS maximizes scalability of transaction throughput by minimizing the Delegates' co-dependency. Once transaction information is evenly distributed between Delegates, each Delegate autonomously and deterministically accepts the Transaction into their chain and reports the validity of the Transaction. Work done by Delegates is redundant for the Byzantine security of the network.

2.3.1 Decentralization *vs* Scalability

There is a classic two-of-three trilemma with consensus algorithms: decentralization, scalability, and security. Without sacrificing security, we are left with an inherent tradeoff between decentralization and scalability. DAPoS is inherently more centralized than other consensus algorithms to achieve the level of scalability necessary for enterprise-level applications. Decentralization is not lost, however, since the Delegates are held accountable for their actions by their constituents, the *Stakeholders*. If the elected powers act maliciously or unreliably, the Stakeholders have the power to replace the incumbents with more trustworthy Delegates. Delegated consensus offers the scalability of a centralized system, but leaves the power of governance decentralized in the hands of the Stakeholders.

To ensure the stability of the Dispatch network at launch, Dispatch will control all of the initial Delegates. Once we believe the network to be stable and there is sufficient voter turnout, we will turn full Delegate control over to community-elected Delegates. The responsibility of Transaction validation will then fall entirely onto the community. Stakeholders are welcome to elect as many Delegates as they deem appropriate, with the consideration of scalability *vs*. decentralization. In comparable delegated systems like Steemit [10], Stakeholders have elected around 90 Delegates [11].

2.3.2 DAPoS Consensus Walkthrough

With the objective of maintaining byzantine fault tolerant consensus while maximizing transaction throughput, the trusted quorum of validators is intentionally small. Dispatch utilizes stake-based Delegate elections to select its trusted quorum (see section 2.3), although the consensus would be unaffected by using any other mode of Delegate selection (such as trusted validators in a private network or masternode validators in a public network). There is a required minimum of 5 and no maximum number of validators or Delegates. It is required that all Delegates can identify and communicate with one another, and all non-validator nodes can communicate with at least one validator.

By formal definitions, DAPoS is not a blockchain consensus protocol. There are no blocks, and it is not entirely immutable. DAPoS differentiates itself by handling individual transactions asynchronously via gossip protocol, not in lockstep. The validators in DAPoS consensus are each responsible for their own state or chain of transactions, but all functioning validators who receive all valid transactions can deterministically agree on the validity of all transactions and conclude on identical world states; stated formally as:

$$\sigma_n = V(T[0],\ T[1],\ T[2],\ ..., T[n]) \tag{2}$$

where V is the deterministic validation function being run by the delegated validators, and $T[0], T[1], T[2]T[n]$, represents all ordered transactions, valid or invalid, submitted by an external actor. Assuming incoming transactions are continuous, all validator world states should be identical before σ_{n-g}, where g is the time it takes for a transaction to be gossiped across all validators.

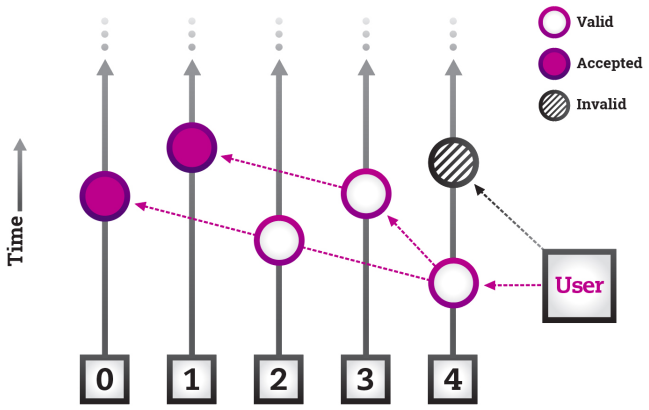

Figure 1: Delegates gossiping transactions

In the simple case of all the Delegates receiving transactions in the order they are published (eg. $T[n]_{ts} < T[n+1]_{ts}$), the Delegates will receive a transaction to determine if it is valid or not. If a transaction is invalid, then no action is taken and the transaction is ignored. If a transaction is valid, the Delegate adds it to the end of its transaction chain and gossips the transaction with its validator peers. However, because the gossiping happens asynchronously, it is very likely that a Delegate will receive two or more transactions out of order (eg. $T[n]_{ts} > T[n+1]_{ts}$). It is then up to the validator to sort the transactions by T_{ts} and re-evaluate the validity of transactions where $T_{ts} > T[n+1]_{ts}$. By sorting and validating transactions asynchronously, the transaction throughput is decoupled from blocktimes entirely and the whole of the network can scale with the validation capabilities of the validators.

DAPoS is Byzantine fault tolerant assuming a high enough gossip redundancy for every valid transaction to reach every honest node. Because of the ECDSA cryptographic signatures [12] associated with every transaction, no actor can fake the validity of a transaction. And because of the deterministic nature of V, every validator can find the same world state in an isolated environment. An unreliable validator who adds invalid transactions to their chain or does not add valid transactions to their chain will deterministically reach a different world state than the other validators and should be replaced.

2.3.3 Bookkeepers

Dispatch utilizes the work of Bookkeepers (sometimes referred to as Learners) to help Stakeholders hold Delegates accountable for their responsibilities, and to help the Delegates distribute information to the Stakeholders efficiently. Bookkeepers are responsible for executing the transactions accepted into the ledger by the Delegates. This makes the Delegates themselves a subset of the Bookkeepers. The Bookkeepers also record the states of the Delegates, so unresponsive Delegates can be reported and replaced quickly, all while providing an audit trail of accountability in case of a Delegate fork. Bookkeepers can also help reduce administrative stress from the Delegates, like syncing the chain. Anyone can elect to run a Bookkeeper node to help the network.

Bookkeepers can each be thought of as independent network scanners that end users can query.

Decentralizing the responsibility of reporting on the ledger will eliminate the single point-of-failure stress carried by traditional network scanners. Dispatch intends to build an interface with which end users can compare the states of the various Bookkeeper accounts.

2.3.4 Delegate Elections

To ensure everyone using the Dispatch network has a say in its governance, the delegated quorum of validator nodes is elected based on stake-based voting. Delegates are the most important actors to the consensus and security of the Dispatch network, so they should be chosen carefully. We fully encourage the community to elect Delegates that have shown their commitment and interest in helping the network by participating as a Bookkeeper, announcing their candidacy on a public forum, and revealing their validator machine hardware specs.

Stakeholders are entitled to one vote per full Divvy, rounded down. Stakeholders cast their votes as transactions of the Election type. Votes are submitted to the election as a percentage of account balance ($\sigma[a]_b$). All Stakeholders can vote on the three properties of the Election State ($\sigma[e]$): the number of Delegates, the ordered list of preferred Delegates, and the salary of the elected Delegates. Delegate election cycles happen on the hour every hour, where p is the calculated median of all the Election transactions' Delegate count votes. At the launch of the network, while Dispatch controls all of the nodes, Dispatch will set the salary rate to 1×10^8 Divvitos (or 1 Divvy/hour) and remove our stake in the Delegate salary as the Stakeholder voter turnout increases.

The number of Delegates ($\sigma[e]_n$) and the Delegate salary ($\sigma[e]_s$) is determined as the simple median of submitted votes. Both values have a maximum rate of change. The Delegate count has a minimum of 5 and can change by a maximum of 10% each election cycle; formally:

$$(\sigma[e]_n)' = \begin{cases} 5 & \text{if } p < \lfloor 0.4 \times \sigma[e]_n \rfloor \ \&\& \ p < 5 \\ \lfloor 0.9 \times \sigma[e]_n \rfloor & \text{if } p < (0.9 \times \sigma[e]_n) \ \&\& \ p > 5 \\ \lceil 1.1 \times \sigma[e]_n \rceil & \text{if } p > (1.1 \times \sigma[e]_n) \\ \lfloor p \rfloor & \text{otherwise} \end{cases} \tag{3}$$

where p is the calculated median of all the Election transactions Delegate count votes.

Unlike the Delegate count, the salary has no minimum, but can only change by a maximum of 0.1% each election cycle; formally:

$$(\sigma[e]_s)' = \begin{cases} \lfloor 0.999 \times \sigma[e]_s \rfloor & \text{if } p < (0.999 \times \sigma[e]_s) \\ \lceil 1.001 \times \sigma[e]_s \rceil & \text{if } p > (1.001 \times \sigma[e]_s) \\ \lfloor p \rfloor & \text{otherwise} \end{cases} \tag{4}$$

where p is the calculated median of all the Election transactions salary votes.

Once the number of Delegates has been determined, Dispatch network implements a "Single Transferable Vote" (STV) method of election [13]. Specifically, it uses a Droop Quota [14] with a Meek Rule method of counting for determining which nodes will become Delegates. The $\sigma[e]_{n'}$ sized ranked list of Delegate ballots are multiplied by the voters balance ($\lfloor \sigma[a]_b \rfloor$), so the results are calculated as "one ballot per share". The Droop Quota is then used to calculate the number of votes a Delegate candidate needs to win a position. The first choice votes are tallied, and if a candidate has enough votes to secure a position they are considered elected as the top choice Delegate. Votes for the lowest ranked candidates are then reallocated to the voters' second choice candidates, and candidates with enough votes to meet the Droop Quota are considered elected. The cycle of vote reallocation repeats until enough candidates have enough votes to be considered elected.

2.4 Eliminating Transaction Fees

Dispatch believes transaction fees seriously inhibit the adoption of a wide variety of applications, so the Dispatch protocol is a decentralized application infrastructure without transaction

fees of any kind. Transaction fees exist in most systems to solve for two problems: (1) validator compensation and (2) preventing network spam. Compensating Delegates with time-based salary instead of transaction-based commission disincentivizes price-gouging in times of high network traffic. To prevent the network spam usually inhibited by transaction fees in other systems, Dispatch implements stake-based rate-limiting. Stake in the network is weighed as balance ownership of the system's native currency, the *Divvy*, and bandwidth is efficiently monitored and allocated by *dynamic fractional reserves*.

2.4.1 The Divvy Token

Arguably the most fundamental resource of the Dispatch network is transaction bandwidth. The Dispatch Protocol allocates this resource based on the ownership of the native token, the *Divvy*. Dispatch is launching the network with a genesis of 18 billion Divvies, each divisible up to eight decimal places. Stakeholders vote on the salaries of the delegates and, in turn, the rate of inflation of the Divvy. The smallest unit that makes up the Divvy is referred to as the *Divvito*, so:

$$10^8 \text{ Divvitos} = 1 \text{ Divvy} \tag{5}$$

The value of the Divvy is directly related to the real-world usage of the network, so attackers cannot spam the network without a proportional investment in the network's bandwidth. What users decide to do with their share of the bandwidth is their own decision, but tying the network's bandwidth to the value of the token should disincentivize behavior that would ultimately be harmful to the network.

2.4.2 Bandwidth in Hertz

Hertz is the unit of measurement for bandwidth consumed by a transaction. Hertz-per-share price is based on the maximum validation bandwidth of the Delegates and, to maximize cost predictability of the network, should remain a relatively stable value. Instead of changing the price of transactions in times of high network traffic, Dispatch implements a variable time-to-reimbursement. This ensures that under most conditions, having some number of Divvitos will always entitle the holder to some number of transactions. To incentivize users to wait to send transactions until the network usage is under-capacity, time-to-reimbursement will be quicker when network-wide traffic is low. Conversely, the time-to-reimbursement will be greater when the network-wide traffic is high. These dynamic fractional reserves optimize to fully utilize network bandwidth at all times. Since each transaction could be a dramatically different amount of work process, network traffic is measured by the amount of hertz used over time. The hertz cost is defined with the following equation:

$$\text{time-to-reimbursement } = 24 \text{ hours} \times \left(\frac{7 \times \text{count(dayHz)}}{\text{count(weekHz)}} \right) \tag{6}$$

When the current days traffic rises higher than the weekly average, the time-to-reimbursement rises to mitigate traffic spikes. The base price of hertz exists to counter volatility in the price of Divvy and is set by the Delegates. Each Delegate gets one vote on the base price, and the final price is determined by taking the median of the Delegates' votes rounded down. Delegates who keep the Hertz price high to reduce their own workload should be voted out by the Stakeholders and replaced with a Delegate whose base hertz price matches what the Stakeholders find ideal.

Before a valid transaction is executed by the Bookkeepers, the sending account's remaining hertz is calculated by subtracting the outstanding hertz spent by the account from the accounts

balance. If the sending account has spent more in hertz han they have balance in their account, the transaction is considered invalid, and it is not gossipped. If in the execution of the transaction, the amount of Divvitos spent on bandwidth exceeds the amount remaining in the accounts balance, an `out_of_bandwidth` error is returned. Delegates will record the valid transaction as unsuccessful, and the remaining balance spent on bandwidth is considered spent.

In the case that a Transaction's $T_v > 0$, or the transaction is a TokenTransfer type, the transaction value T_v must be greater than or equal to the bandwidth cost of that transaction. The hertz cost is considered spent by the *To* account (T_{to}) to prevent the repeated transferring of the same tokens.

3 Dispatch Artifact Network (DAN)

3.1 Artifacts

Artifacts are the distributed data objects stored in the Dispatch Artifact Network; formally A. All Artifacts stored in the DAN are referenced in the Dispatch ledger by their Merkle hash [16]. Artifacts can be sharded and encrypted for the security of private data. Artifacts come in two types: structured and *BLOBs* (Binary Large Objects). A structured Artifact has several entries with defined properties, similar to rows in a SQL database. A BLOB Artifact could be a document, *.csv* file, VR asset, software program, a side chain, or any other arbitrary data with a defined format. While both Artifact types benefit from the decentralized algorithms that govern the DAN, structured Artifacts also benefit from the additional analytical capabilities of the *Dispatch Guru* 3.5.

The governance of each Artifact is defined by its Uploader in its associated smart contract. Therefore an Artifact can be defined as a property of an Account States stored memory, or formally:

$$A \not\subseteq \sigma[a]_r \tag{7}$$

To the world state (σ), this initialization smart contract is the Artifact's creation step. The smart contract is executed in the DVM (Dispatch Virtual Machine) to determine permissions around the *reading*, *updating*, *deleting*, *distributing*, *storing*, and *analysis* of that Artifact. Because the ledger cannot govern what users do with the DAN data, the distribution of Artifacts is governed by algorithms outside of the DVM.

3.2 Actors & Roles

There are three roles a node can assume participating in the DAN: the Uploader, Downloader, and Farmer. The roles are not mutually exclusive; in fact, a nodes role in a given interaction is likely to depend on the specific Artifact and algorithm the node is participating in.

Uploaders Uploaders seed the DAN with Artifacts. Uploaders deploy Artifacts via smart contracts and serve those Artifacts to the Farmers and Downloaders. The number and price of Farmers enlisted is up to the Uploader, allowing the Uploader flexibility between affordability and availability.

Downloaders Downloaders are the Artifact consumers and the most common role in the DAN. Downloaders receive their Artifacts from Farmers or Uploaders. Downloaders typically pay the Farmers for their bandwidth in transmitting the Artifact. If the Artifact is encrypted, the Downloader must acquire the encryption key from the Uploader by telling them which Farmer their Artifact came from. Once the Downloader has the Artifact, they can then act as Farmers and serve the Artifact themselves for the bandwidth reward.

Farmers The Farmers are extremely important to the scalability of the Dispatch Protocol. Farmers store Artifacts for Uploaders, distribute Artifacts to Downloaders, and execute analytics against their Artifacts for the Guru. Farmers are typically compensated for their storage by the Uploaders and their bandwidth by the Downloaders. On the launch of the network, Dispatch will be offering free farming to help seed the DAN.

3.3 Artifact Security and Availability

Programmable governance of Artifacts enables incredible flexibility around the security and availability of Artifacts stored in the DAN. *Encrypting and sharding* are both optional security measures an Uploader can choose to implement, while other functionality is implemented in the DAN by default.

3.3.1 Kademlia DHT

The Kademlia Distributed Hash Table (KDHT) is a hashtable data structure that sits on every node in the Dispatch network. The Kademlia DHT ensures the quick and efficient discovery of Artifacts stored in the DAN. The KDHT maps Artifact addresses to physical IP addresses with a search efficiency of $O(\log(n))$, where n is the number of Artifacts in the network. In addition to the search for Artifacts, the Kademlia DHT can also help nodes send transactions directly to Delegates given the Election State $\sigma[e]$.

3.3.2 Proof-of-Replication (PoRep)

Proof-of-Replication (PoRep) exists to disincentivize Sybil attacks and gaming of the system by those who would otherwise impersonate multiple Farmers hosting the same file. The solution, outlined Filecoins white paper [17], is for the Uploader to provide each Farmer with a differently encoded version of the Artifact using appropriately slow encoding process such that:

$$T_{\text{pass}} < T_{\text{decode}}(A) + T_{\text{encode}}(A) \tag{8}$$

where T_{pass} is the time a Farmer has to respond to a challenge. When each Farmer has a different encoding of the same Artifact, and the time to de-encode and re-encode an Artifact is greater than the time the farmer has to respond to a challenge, the Farmers then have to store separate copies of the Artifact for each Farmer they are claiming to be.

3.3.3 Proof of Retrievability (PoRet)

Proof of Retrievability (PoRet)[18] ensures the availability of an Artifact by a Farmer. Farmers are compensated for their storage by Uploaders based on duration of their storage agreement. At the time of the agreement, a number of PoR challenges is agreed upon by both the Farmer and the Uploader, and the full contract payment is put into escrow to be released upon completion of PoR challenges or at the agreement's expiration. The Uploader can challenge the Farmer any time during the agreement's duration. Each passed PoR challenge releases r tokens to the Farmer, and each failed PoR test releases r tokens to the Uploader, where:

$$r = \frac{\text{Total storage payment}}{\text{Number of PoRet challenges}} \tag{9}$$

To initiate a PoRet challenge, the Uploader must provide the Farmer with the index of a small subset of the Artifact. Upon the receipt of a challenge from the Uploader, the Farmer has fixed amount of time to respond to the test with the hash of a specific subset of the Artifact.

$$PoR\ Response = \texttt{KEC}(A[subset]) \tag{10}$$

If no response is provided, the Farmer is presumed offline, the test reward is returned to the Uploader, and optionally triggers the fail-safe redistribution of the Artifact to a new Farmer to ensure availability. Otherwise, the Uploader posts their own PoRet response. If the hashes match, the Farmer passing the test is given their reward. If the Farmer and Uploader's hash do not match, neither party can be trusted because both have an economic incentive to lie about what the true hash is. It then becomes the decision of the other owners of the artifact (Farmers or Downloaders) as to which party is telling the truth. In the case of a tie, the reward goes to the Uploader.

3.4 Artifact Storage and Distribution

At the core of the Dispatch Artifact Network is the decentralized distribution of data. More fundamental still is the ability for any useful data to be *stored* and *distributed*. The DAN handles data distribution differently depending on the actors participating and the on-chain, governing rules of the Artifact. The *Storage Orderbook* defines the agreements between Farmers and Uploaders. The *Multi-Party MiH* protocol defines the procedure for distribution between Farmers and Downloaders.

3.4.1 Storage Orderbook

The agreements between Uploaders and Farmers are recorded in the *Storage Orderbook*. Farmers publish storage offers on the Ledger defining three parameters: (1) their maximum available storage capacity, (2) their maximum storage duration, and (3) their minimum cost. When an Uploader decides to enlist the storage capabilities of a Farmer, they accept the storage offer with the Artifact Merkle hash (or the Merkle hash of the subset of the Artifact the Farmer will be responsible for storing), the size of the data they want stored, and the duration. The Farmer then confirms the agreement, and the Uploader sends the Artifact or shard to the Farmer via the Make-it-Happen (MiH) Protocol.

3.4.2 Multi-Party Make-it-Happen (MiH)

Make-it-Happen (MiH) is the protocol for distributing Artifacts to Downloaders from Farmers. MiH both incentivizes Farmers and Downloaders to act honestly in a situation where both parties have an incentive to lie. Additionally, MiH updates the world state (σ) to include the new Downloader of an Artifact. In the case where Farmers are charging Downloaders for bandwidth in the exchange of an Artifact, both parties have a reason to lie. The Farmer could collect the Downloader's payment without transferring the Artifact, and the Downloader could receive the Artifact without paying the Farmer for their bandwidth. MiH provides trust to the larger consensus group that neither party is dishonest, as dishonesty works against the self-interests of both parties engaged in MiH.

In the MiH protocol both participants store Dispatch tokens in excess of the actual cost of the transfer in escrow, outside of the reach of either party for the extent of the exchange. How the Artifact exchange then happens or how many attempts it takes then becomes irrelevant. Once the tokens are locked, the burden of recovering those tokens rests solely with the Farmer and Downloader. Once the Artifact exchange is complete, the payment is transferred to the Farmer, and the escrow security deposit is returned to both parties. Because a timed release of the escrow would incentivize one of the parties to act dishonestly, tokens put in a MiH escrow remain there until the transfer happens. This solution is fault-tolerant to Farmer or Downloader downtime and non-malicious failures. Downloaders should avoid going into transfers with Farmers still in an exchange. In the case that the Farmer has lost their copy of the content, such as a corrupted hard

drive, and has no way of retrieving it, the Farmer can release the Downloaders tokens and close the MiH exchange by burning their own deposit.

In the case that there are many Farmers hosting an Artifact, the Downloader can go into a *Multi-Party MiH* exchange with any number of Farmers. The many-to-one nature of the Artifact transfer significantly improves the download times of larger Artifacts, and because Artifacts are formatted as Merkel trees, the Downloader has complete transparency into both the quality and quantity of data transferred by each Farmer and can compensate them for their bandwidth accordingly.

3.4.3 Update Deltas

Updating Artifacts in the DAN presents a unique challenge where data is distributed across any number of Farmers with any quantity of available storage capacity. It cannot be assumed that Farmers of an Artifact will have the capacity to accept a large update to the Artifact they are storing, nor can it be assumed that all updates will lower the size of the Artifact. The DAN therefore considers all updates to be considered additions in the form of Artifact *Deltas*; formally Δ.

A Delta defines the exact differences between an Artifact (A) and its updated version (A'). An Artifact can have any number of Deltas, but they must be ordered. Deltas have no size limitation. An Artifact can be considered *updated* when it has been combined with all of its Deltas in order; formally:

$$A' = \mathbf{U}(A, \Delta_1, \Delta_2, ...\Delta_n) \tag{11}$$

where n is the number of Deltas associated with the Artifact and $\mathbf{U}$ is the deterministic Artifact update function. The distribution access to Deltas can be defined in the Artifacts initialization smart contract.

3.5 Zero-Knowledge Analytics: Dispatch Guru

Arguably the most powerful tool in the Dispatch Protocol is the *Dispatch Guru*, which enables analytic queries across any of the structured Artifacts distributed in the DAN without sacrificing data ownership. A data creator (like a Facebook user or a pharmaceutical research company) can earn rewards for enabling third party access to analytics of their data, without needing to reveal or transfer ownership of any of the underlying data itself. A data researcher (like an advertising agency or a pharmaceutical research company) can pay for the extraction of data analytics from a wide variety of data and verify the integrity of the results without ever having to see the data itself. The Dispatch Gurus benefits of data sovereignty and distributed data analytics are powered by a combination of *Homomorphic Hidings*[19] and *Proxy Re-encryption*[20].

By encrypting Artifacts stored in the DAN using a Homomorphic encryption scheme, operations are able to be conducted on the encrypted data to produce an encrypted answer. As a very simple example, Homomorphic encryption allows for computations such as:

$$\text{ENCRYPT}(x) + \text{ENCRYPT}(y) = \textit{encrypted\_query} \tag{12}$$

$$\text{DECRYPT}(\textit{encrypted\_query}) = q \tag{13}$$

$$\text{where} \quad x + y = q \tag{14}$$

Then without giving the data researcher the decryption key of the original Artifact, the *encrypted_query* is re-encrypted by any party (likely the Artifact's Farmers) with a re-encryption key provided by the Uploader based on the original Artifact encryption key and the public key

of the data researcher. The data researcher can then decrypt the answer to their query, q, with their own private key given the re-encrypted *encrypted_query*. The answer is trustless given the user can verify that there is some encryption key, k, that can turn q into *encrypted_query*. The final step would be the aggregation of all the query results returned by all the data owners into a single result by the data researcher. Thus, the Dispatch Convoy can deliver reliable, trustless query results across any amount of distributed data stored by any number of data owners.

4 Dispatch Virtual Machine (DVM) and Smart Contracts

4.1 Transaction Execution in the DVM

The DVM is what ties together the Dispatch ledger and the DAN. Transactions are executed by Bookkeepers and relevant Farmers following the acceptance of transactions into the ledger by the Delegates. Transactions change the world state in the state-transition function, which varies depending on the type of transaction sent; formally:

$$\sigma_{t+1} \equiv \mathbf{Y}(\sigma_t, T) \tag{15}$$

where $\mathbf{Y}$ is the state-transition function. $\mathbf{Y}$ can allow for arbitrary computation, while the data in T allows the storage of any quantity of arbitrary data, either on-chain or in the DAN. The state-transition function is deterministic and can be considered either *successful* or *unsuccessful*. A transaction can be *unsuccessful* at the code level (as defined in a smart contract) or at the system level (eg. *out_of_bandwidth error*).

Part of the scalability of the Dispatch protocol comes from the separation of responsibilities between the Bookkeepers of the ledger and the Farmers of the DAN. All Bookkeepers are responsible for transactions that update the world state, but the bulky operations that create, update, distribute, and analyze Artifacts are the responsibilities of the Farmers of the network. There are minimal DAN operations that can trustlessly measure work done off-chain, so the onus often falls on the Downloaders to apply Deltas to Artifacts.

4.2 EVM Integration

To capture the value of the existing Dapp ecosystem, the Dispatch Protocol is designed to be backwards compatible with the Ethereum Virtual Machine (EVM). This means that EVM bytecode is acceptable in the DVM even though DVM bytecode will not necessarily be acceptable by the EVM. The DVM supports the interface for all of the opcodes defined in the Ethereum Yellow Paper [3] (even though some of the results are nonsensical in Dispatch's architecture) and adds a new set to refer to operations on the DAN.

4.3 DVM Extension

The DVM supports the interface of every opcode defined in the EVM. Some modifications have been made that provide alternative but compatible functionality. Examples include :

0x41 (COINBASE)- The Dispatch protocol uses a delegated consensus mechanism that does not have block beneficiaries. Instead this value is replaced with the hash of the ordered list of Delegate addresses.

0x44 (DIFFICULTY)- Block difficulty is always 0.

0x45 (GASLIMIT)- Instead, returns the amount of Hertz consumed at the given time interval.

and many more opcodes are updates with minimal changes such as mapping Ethereums gas to Dispatchs bandwidth measurement, Hertz.

Much more interesting than the update opcodes are the entirely new set referring to the DAN *operations*. These new opcodes start with the prefix *0xd0*.

0xd0 (ARTIFACT)- Returns the Merkle hash of the accounts Artifact.
0xd1 (ARTIFACTSIZE)- Returns the size in bytes of the accounts Artifact.
0xd2 (ARTIFACTSTRUCTURE)- Returns 0 for BLOB (Binary Large Object) Artifacts and 1 for structured Artifacts.
0xd3 (ARTIFACTENCRYPT)- Defined at time of account initialization. Returns 0 for unencrypted Artifacts and 1 for encrypted Artifacts.
0xd4 (READARTIFACT)- Formally declares the address of a new Downloader on the ledger.
0xd5 (UPDATEARTIFACT)- Saves the hash of a new Artifact Delta (Δ) to the ledger.
0xd6 (GURU)- Saves the requested steps of a Guru query to the ledger for Farmers to execute against the accounts Artifact and Deltas.

4.4 High-Level Languages

The first major language supported by the Dispatch protocol is *Solidity*. Solidity and the *solc* compiler [21] will compile down to the EVM bytecode that utilizes a majority of the functionality of the DVM. Dispatch intends to develop a compiler for a new high-level language called *Solidity++* which will include all the base functionality of Solidity as well as the additional Artifact-related functionality.

Eventually, Dispatch intends to develop a DVM bytecode back end for the LLVM compiler [22]. The LLVM compiler supports a wide variety of front-end languages ranging from Fortran to Python. The wide extension of supported smart-contract languages should enable a wide range of developers from all backgrounds to start developing their own DVM Dapps.

References

[1] Harm, Julianne, et al. Ethereum vs. Bitcoin. *Economist*, Creighton University, 2018, www.economist.com/sites/default/files/creighton_university_kraken_case_study.pdf.

[2] Yedlin, Rob. App Statistics. *State of the Apps*, 10 Sept. 2018, www.stateofthedapps.com/stats.

[3] Wood, Gavin. Ethereum: A Secure Decentralized Generalized Transaction Ledger. Ethereum.org, 2018, www.ethereum.github.io/yellowpaper/paper.pdf.

[4] Croman, Kyle, et al. On Scaling Decentralized Blockchains. *Financial Cryptography and Data Security 2016*, International Financial Cryptography Association, 2016, www.fc16.ifca.ai/bitcoin/papers/CDE+16.pdf.

[5] "Etherscan: The Ethereum Block Explorer." 2018, www.etherscan.io/

[6] Bertoni, Guido, et al. "KECCAK." 2017, www.keccak.team/keccak.html

[7] Newman, C. Date and Time on the Internet: Timestamps. *IETF Tools*, The Internet Society, July 2002, www.tools.ietf.org/html/rfc3339.

[8] Dispatch Protocol: Introduction to DAPoS. *Dispatch Labs*, 20 Apr. 2018, www.github.com/dispatchlabs/TechnicalDocs/blob/master/Introduction%20to%20DAPoS.pdf.

[9] Delegated Proof-of-Stake Consensus. *BitShares*, www.bitshares.org/technology/delegated-proof-of-stake-consensus/.

[10] Steem Whitepaper. *Steem*, June 2018,www.steem.io/steem-whitepaper.pdf.

[11] Witness Voting. *Steemit*, 10 Sept. 2018, www.steemit.com/~witnesses.

[12] Blake, Ian F., et al. *Advances in Elliptic Curve Cryptography*. ser. 317, Cambridge University Press, 2005.

[13] Rosenstiel, Colin, and James Woodward-Nutt. *How to Conduct an Election by the Single Transferable Vote*. 1997, www.rosenstiel.co.uk/stvrules/.

[14] Droop, Henry Richmond. *On the Political and Social Effects of Different Methods of Electing Representatives*. London, 1869.

[15] Hill, D., et al. Single Transferable Vote by Meeks Method. *Department of Internal Affairs*, New Zealand Government , 13 May 1986.

[16] Merkle, R. C.. "A Digital Signature Based on a Conventional Encryption Function". *Advances in Cryptology CRYPTO '87*, 1988.

[17] Filecoin: A Decentralized Storage Network. *Filecoin*, Protocol Labs, 19 July 2017, www.filecoin.io/filecoin.pdf.

[18] Vorick, David, and Luke Champine. Sia: Simple Decentralized Storage. *Sia*, Nebulous Inc., 29 Nov. 2014, www.sia.tech/sia.pdf.

[19] Gentry, Craig. A Fully Homomorphic Encryption System. *Applied Cryptography Group*, Stanford University, Sept. 2009, crypto.stanford.edu/craig/craig-thesis.pdf.

[20] Blaze, Matt, et al. Divertible Protocols and Atomic Proxy Cryptography. *Advances in Cryptology*, edited by Kaisa Nyberg, Springer Berlin Heidelberg, 1998, pp. 127144.

[21] *Solidity*. Ethereum, 10 Sept. 2018, www.github.com/ethereum/solidity.

[22] *The LLVM Compiler Infrastructure*. The LLVM Foundation, July 2018, www.llvm.org/.